Machine Learning (CSC 246): Project 2 Writeup

Catherine Giugno

20 March 2022

1 Training Recommendations

1.1 Reccommendations for Effective Learning Rate, Number of Epochs, and Batch Size

Table 1: Effective Learning Rate, Number of Epochs and Batch Size

Parameter	**Value**
Learning Rate	e^{-1}
Number of Epochs	500
Batch Size	10

This yielded a Neural Network with a final accuracy of 95.998365% and final F1 score of 0.740331 on the Development Dataset.

1.2 Accuracy and F1 Data for the Training Set and Development Set over each Epoch

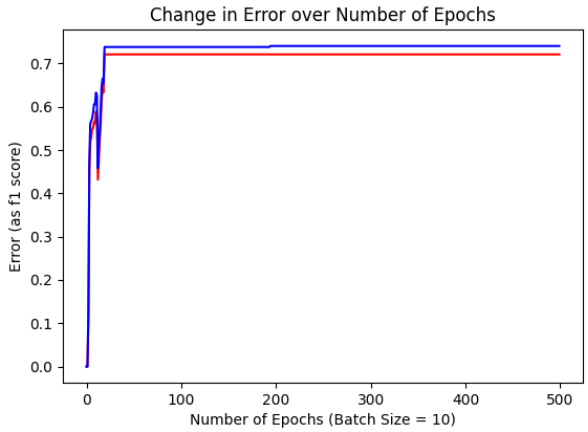

Figure 1: F1 Score for Neural Network produced with Learning Rate e^{-1} and Batch Size 10 over 500 Epoches; Red = F1 score for Training Data; Blue = F1 Score for Test Data

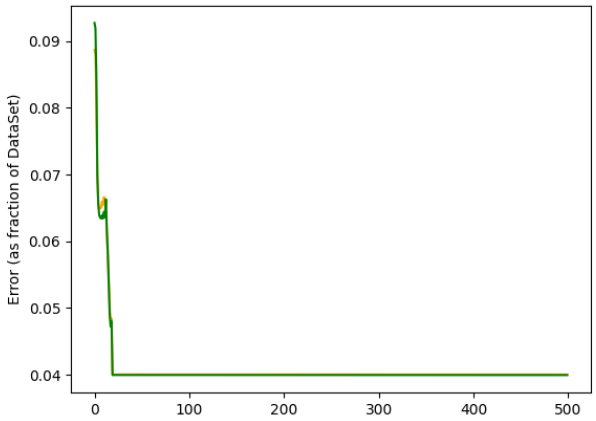

Figure 2: L1 Loss for Neural Network produced with Learning Rate e^{-1} and Batch Size 10 over 500 Epoches; Orange = Error for Training Data; Green = Error for Test Data

1.3 Expected Length of Time to Train

On my device (an HP Spectre x360 with a i7 Core Processor and 16.0 GB), the training process took approximately 4 hours.

2 Usage Instructions

As with any python project, the first step is to open the Command Prompt/Terminal, navigate to the project folder, and run the program in python, using the command:

python NeuralNetwork.py

The terminal will then print a welcome message and allow you to choose which functionality (*Inference Engine* or *Neural Network Trainer*) that you desire to use. You can also enter the command 'quit' to quit.

Welcome to the Terminal!
Please pick a functionality:
1. Train a new Neural Network (input '1')
2. Run the Inference Engine (input: '2')
Or enter 'quit' to quit the Terminal.

If the input '1', '2' or 'quit' is not entered, then the program will re-prompt the user.
Otherwise:
1. If the user entered "1", the program will prompt the user to enter the parameters for a new Neural Network (covered in subsection *How to Train a New Model*).
2. If the user entered "2", the program will load in the best Neural Network model, and prompt the user for an example for it to classify (covered in subsection *How to Use the Inference Engine*.
3. If the user entered 'quit', the program will quit.

2.1 How to Train a New Model

Once you are in the Training Program, you will be greeted with a welcome message and a prompt for input.

Welcome to the Training Program!
Choose your method of Neural Network initialization:
1. Read in from file (input '1')
2. Start with a brand new Neural Network (input '2')
Or enter 'quit' to quit the Training Program.

If the input "1", "2", or "quit" is not entered (i.e., no choice of initialization is made), then the program will reprompt the user. If "quit" is entered, the Training Program will end and the user will be returned to the *Terminal*

program and again prompted for which functionality (*Traning Program* or *Inference Engine*) that they wish to use.
Otherwise:
1. The user will be prompted for a filepath to read the Neural Network structure from.

> Input a filepath to read the Neural Network from:

An appropriate file structure is:
a. A full filepath to a particular file (ex., for a dataset, *linearSmoke*):

```
C:\Users\15854\Documents\UR\Year 4\Spring 2022\CSC 246\
pro2.1\dataProject2\data\linearSmoke
```

b. The relative filepath to a particular file, specifying its placement in a subdirectory of the folder NeuralNetwork.py is located in (ex., for the dataset, *linear smoke*, and the same filepath above)

```
\dataProject2\data\linearSmoke
```

If you wish to use my best Neural Network, input filepath:

> NNModel

Alternatively, you could use *NNCopy*, which I left in for paranoia's sake. If a valid filepath is read in, a Neural Network will be built from the file the path references.
As a side node, "quit" can also be entered here to quit the Training Program.
2. The user will be prompted to specify a number of hidden layers for their Neural Network:

> Input an integer number of Hidden Layers that you desire your network to :

As mentioned above, this input should be an integer value. For example, 1 would be approrpriate, but 1.1, 1 1 1 and *a* would not be. If an inappropriate value is entered, the user will be taken back to the beginning of the *Training Program*, and, once again, asked to enter their choice of Neural Network initialization method. If 'quit' is entered, the *Training Program* will end.
If the input value is appropriate, then, for every Hidden Layer the Neural Network is expected to have, the user will be prompted to enter the number of nodes that the Neural Network should have in that layer:

> Input the integer number of hidden nodes you wish to have in this layer:

As mentioned above, this input should be an integer value. For example, 1 would be approrpriate, but 1.1, 1 1 1 and *a* would not be. If an inappropriate value is entered at any point, the user will be taken back to the beginning of the *Training Program*, and, once again, asked to enter their choice of Neural Network initialization method. If 'quit' is entered at any point, the *Training Program* will end.
When the number and organization of the Hidden Layers has been made, a Neural Network will be built according to those specifications.
After the Neural Network has been initialized—whether via a filepath or user input—the program will prompt the user for a learning rate to train the Network with:

> Now, input the power of e that you wish the learning rate to be (ex. -1):

The input here should be an integer value as well. For example, 1 would be approrpriate, but 1.1, 1 1 1 and *a* would not be. If an inappropriate value is entered, the user will be taken back to the beginning of the *Training Program*, and, once again, asked to enter their choice of Neural Network initialization method. If 'quit' is entered, the *Training Program* will end.
The program will then prompt the user for a batch size to train the Network with:

> Now input the integer number that you wish the size of the batches to be:

The input here should also be an integer value. For example, 1 would be approrpriate, but 1.1, 1 1 1 and *a* would not be. If an inappropriate value is entered, the user will be taken back to the beginning of the *Training Program*, and, once again, asked to enter their choice of Neural Network initialization method. If 'quit' is entered, the *Training Program* will end.
Finally, the user is prompted for a number of epoches to train the Network for:

> Finally, input the integer number of epochs that you wish the Neural Network to train for:

The input here, like the others, should be an integer value. For example, 1 would be approrpriate, but 1.1, 1 1 1 and *a* would not be. If an inappropriate value is entered, the user will be taken back to the beginning of the *Training Program*, and, once again, asked to enter their choice of Neural Network initialization method. If 'quit' is entered, the *Training Program*

will end.
After all the parameters have been decided upon, the program will read in the data from "htru2.train" and "htru2.dev", and train the initialized Neural Network on the data according to the specified parameters.
The program will print each epoch as it begins. This is a sanity measure, in order to let you know how far your training program has actually progressed, since you began it.
When the network has finished training, it will output two graphs. The first shows the $f1$ score for the Neural Network on both the training and development data, for every epoch. The second shows the *accuracy*, calculated as the percentage of examples the Neural Network got correct on both the training and development data, for every epoch.

2.2 How to Use the Inference Engine

Once you are in the Inference Engine, you will be greeted with a welcome message and a prompt for input.

> Welcome to the Inference Engine!
> Input a filepath to read the Neural Network from:

An appropriate file structure is:
a. A full filepath to a particular file (ex., for a dataset, *linearSmoke*):

```
C:\Users\15854\Documents\UR\Year 4\Spring 2022\CSC 246\
pro2.1\dataProject2\data\linearSmoke
```

b. The relative filepath to a particular file, specifying its placement in a subdirectory of the folder NeuralNetwork.py is located in (ex., for the dataset, *linear smoke*, and the same filepath above)

```
\dataProject2\data\linearSmoke
```

If you wish to use my best Neural Network, input filepath:

> NNModel

Alternatively, you could use *NNCopy*, which I left in for paranoia's sake.
Next, the Inference Engine will prompt you to enter a input parameters:

> Please enter the input to an example you wish to test.
> Preferably, do this by writing the input values on a single line
> with a single space between each value. Ex:
> 1 2 3 4
> Input:

So the input should be a list of numerical values, equal in quantity to the values in the actual dataset. For the htru2 data, 8 values would be appropriate. For example,

> 32.921875 33.67180974 4.655360766 24.0367114 37.99916388 65.91602415 1.70029174 1.822946206

would be an valid input. However,

> 32.921875 33.67180974 4.655360766 24.0367114 37.99916388

would not be an valid input because it has too few values. Additionally,

> a b c d e f g h

would not be an valid input because it has letters, not numbers, as input values. If an input is encountered that is invalid because it does not satisfy one or both of these conditions, the program will print an error and re-prompt for input.
After the input values are provided to the program, the program will prompt the user for an output value (i.e., the correct classification).

> Now enter the output as a 0 or 1. Ex:
> 1
> Additionally, remember you can enter 'QUIT' to quit at any time.
> Output:

The input should be a single number, '1' or' 0', to represent the binary classification of the input data. For example, 1 and 0 are both valid, but *a* and 1 0 1 1 are not valid, because neither is a single numerical value. If an input that is invalid because it is not a single numerical value is encountered, the program will print an error and re-prompt for input.
After the input and output values are entered, the program will attempt to calculate a classification, given the input. This classification will be printed, along with the output value, and the difference between them.
As the prompt mentions, entering 'quit' will quit the Inference Engine at any time.

3 What I Learned

During one of our classes this past week, Professor Purtee mentioned offhand that the hardest part of this project would be implementing the backpropagation algorithm. He was very, *very* right.

Even after learning backpropagation in class, and working through related homework problems, it took putting the equation into code for me to really understand how it worked. It was an interesting process, and the way I like to think of the it is this—I started out the project with an idea of backprop that was vague or incomplete in certain areas. Although I attempted to translate that idea into code—many times—I would run a dataset on it, and from the results, realize I was missing something. As I returned to my code to debug, I found myself mentally tracing the algorithm in my head and comparing it to what I saw in the lecture and textbook. When there was a discrepancy between my relatively vague idea of backprop and the 'real' backprop (or more frustratingly, a missed parenthesis or incorrect array index in what should have been proper code), I had to sort out exactly how and why my initial implementation of the equation was wrong. The more I did that, the more exact and complete, my idea of backprop became. So, by the time I finished coding, and had a proper backpropagation algorithm, I had gone through the algorithm itself several times, and I hope, nearly memorized how it should work!

A specific example of an area of backprop (or really, Neural Net structure in general) that I found challenging to implement was the Softmax Activation function in the output layer. When I found the softmax equation's derivative in our second Homework Problem Set, I didn't put much thought into what each variable—'y_j', 'y_k', etc.—meant. I understood the equation on a mathematical level, but not a conceptual one.

Coding softmax activations for each output node, however, meant that the values for each variable weren't just a mathematical given. They were a part of a *actual* Neural Network. I had to return to the notes and textbook several times to understand how performing a Softmax Activation would actually work. Thankfully, I was remembered to something Professor Purtee mentioned in lecture, when he was describing how a softmax function worked– that it first calculated a value (y_k) from each output node's incoming connections, then summed them over the output layer, and iterated over every output node again, dividing by the sum of all y_j. In other words, the function had to iterate over each output node twice, first to find y_k and second to divide by the sum of all y_j.

This was the key for me! Though the granular details would come later, knowing that softmax, at its most basic, was an "average" that needed the y_j for every connection before it could truly be calculated gave me the conceptual start necessary to solve the problem.