NAME: OMAR USMAN UPPAL

CMS ID: 203515

ASSIGNMENT 3

Machine Learning/Artificial Intelligence

DATASET:

The dataset we are using is NOTMNIST. It is designed to look like classic MNIST dataset however it is more complex than MNIST.

DATASET was made by public by **Yaroslav Bulatov**. He extracted glyphs from publicly available fonts.

STRUCTURE OF THE DATASET:

DATASET consists of two folders: “notMNIST_small” and “notMNIST_large”.

Each folder consists of 10 sub folders. The name of the sub folder is the label for the images inside the sub folders. The names of the 10 subfolders in each folder are first 10 English alphabets **(“A” to “J”).**

Images inside the sub folders are ***.png** format with some **blank images** also. There are 52,910 and 1872 images in each sub folder of notMNIST_large and notMNIST_small respectively.

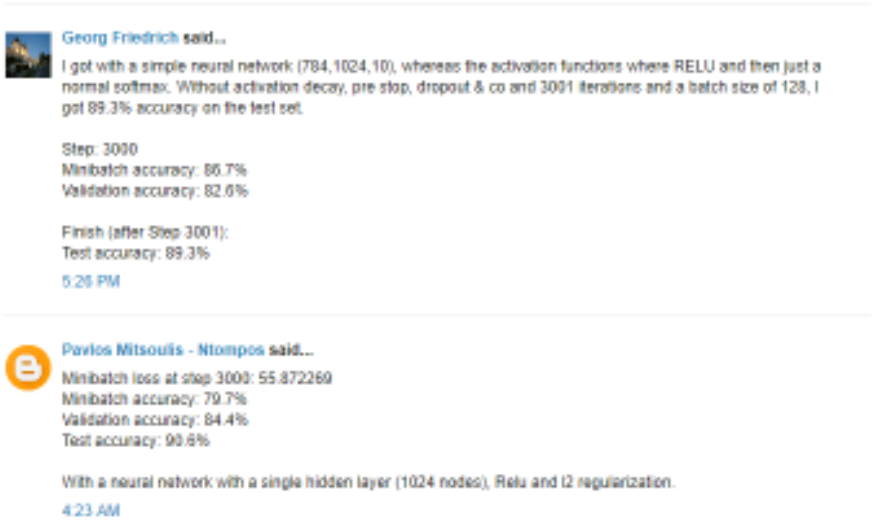
First of all, I will explain the results of this exercise. Next I will explain the methodology that is used for this assignment.

RESULTS:

Observations and explanations:

Few important observations are:

- Cost function decreases till it reaches the minimum value. This was observed in all the results. The decrease in the cost function shows that our system is learning in the model and moving in the right direction.
- In all the cases training data accuracy was less than test data accuracy.
    - Possible reasons are that there are samples in the training set that are difficult to predict as similar kinds of images are present in different sub folders of "notMNIST_large". I could not visually check the images but there are images that do not look like any alphabet. It is possible that sometimes the system predicts them as one class and sometimes other.
    - I also explored the dataset link http://yaroslavvb.blogspot.com/2011/09/notmnist-dataset.html and certain comments show that same thing is observed by others also.
    - We have used sample of 950,000 images or 195,000 images out of 500,000 images. We may need to add more samples as the samples were selected randomly. This I will try with more training time available.
    -



- Validation set accuracy was plotted to check if overfitting is occurring or not. It seems that no overfitting is occurring as validation accuracy is increasing with each EPOCH.
- Validation accuracy and its graph almost follows the training data accuracy.
- Regularization and dropout algorithms are not used but it is expected that it will increase the test accuracy.
- Weights and biases are initialized randomly.

TEST RESULTS:

RESULT 1: Test Samples: 10100, Training Samples: 95000, Learning Rate=0.01, EPOCH=15

```
Train Accuracy: 87.2% (95000 samples)
Validation Accuracy: 81.7% (5000 samples)
Test Accuracy: 88.8% (10100 samples)
```

```
plt.plot(epch_cost)
plt.show()
```

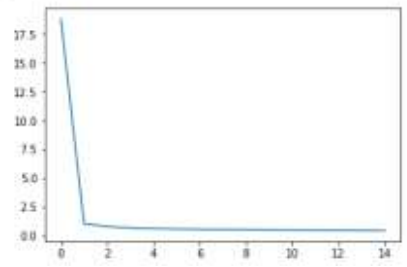


Figure 1 COST vs EPOCH

These are the first results of the system with 95000 training samples. First graph shows the cost function and we can see that cost is decreasing and accuracy is increasing. There is one downward spike in the accuracy function but then there is a sharp rise also. Training Accuracy and Test accuracy are approximately the same.

```
plt.plot(train_accuracy_trend)
plt.show()
```

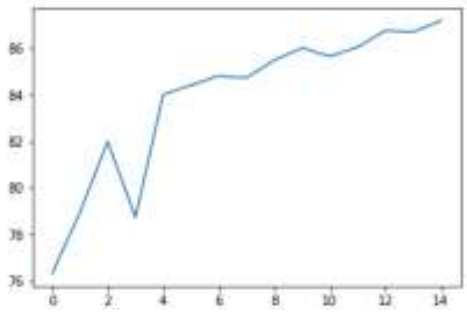
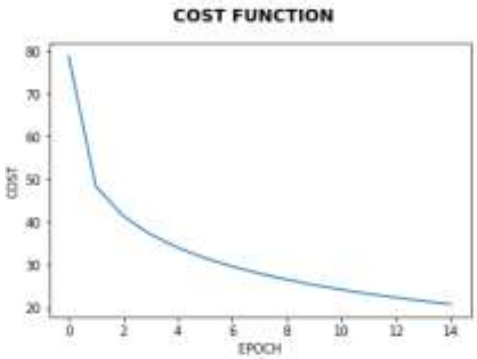


Figure 2: Training Accuracy

RESULT 2: Test Samples: 10100, Training Samples: 195000, Learning Rate=0.01, EPOCH=15

```
Train Accuracy: 79.8% (195000 samples)
Validation Accuracy: 79.4% (5000 samples)
Test Accuracy: 87.8% (10100 samples)
```



The result show there is gradual decrease in the cost function while training accuracy is increasing. This is a good decrease. We have increased the number of training samples for this experiment. All other variables are the same

In the next results we will also plot validation data accuracy to show if our model is overfitting or not.

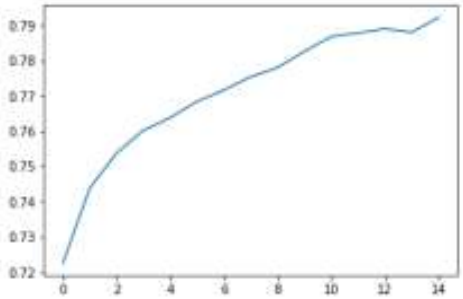


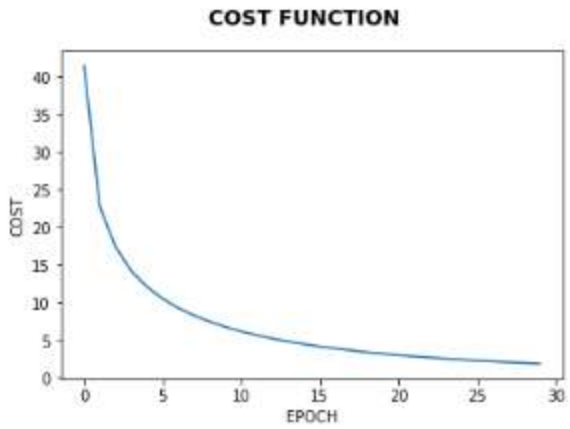
Figure 3 Training Accuracy

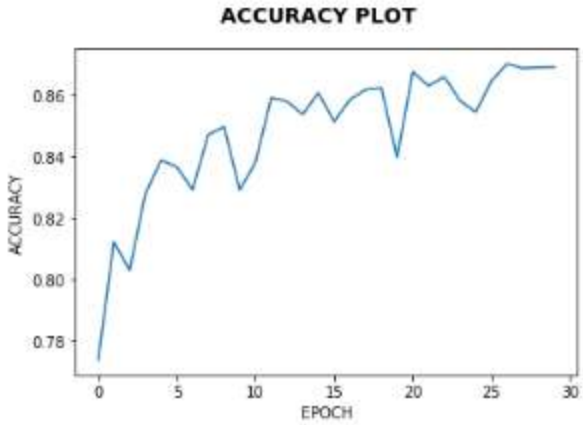
RESULT 3: Test Samples: 10100, Training Samples: 195000, Learning Rate=0.01, EPOCH=30

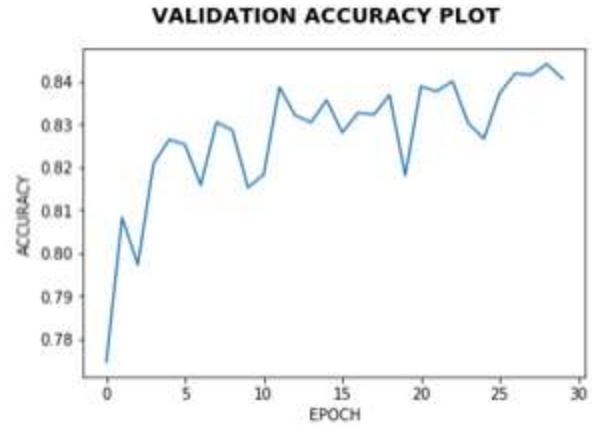
```
Training Accuracy:86.89641 %

Validation Accuracy: 84.06 %

Test Accuracy: 90.732676 %
```







This time we incease the EPOCH also. We saw in Result 2 that accuracy was still increasing at the last EPOCH. We also increased the learning rate.

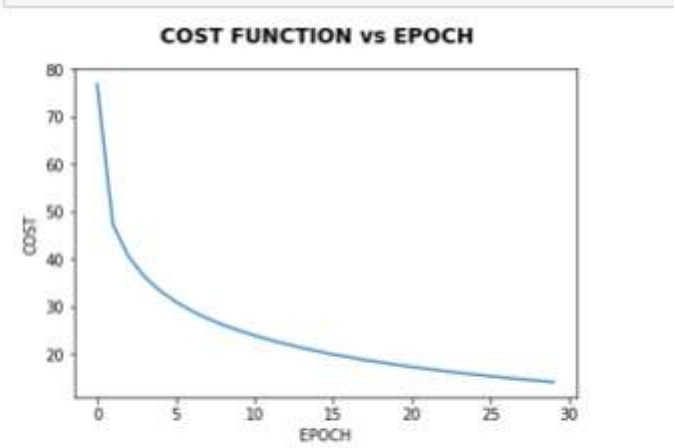
In the result we can see the spikes in the accuracy plots. The possible reaons looks like learning rate is high and so system is trying to get to minimum level of cost function. and

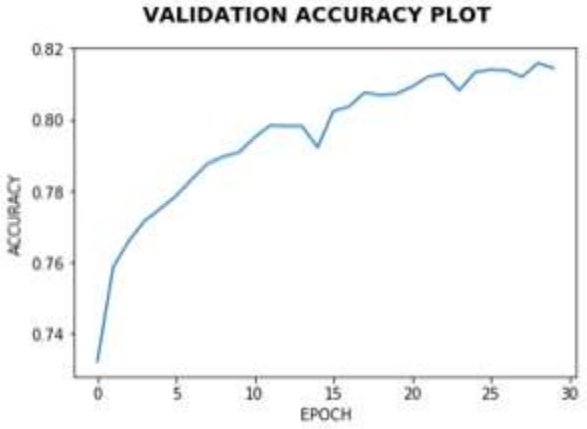
RESULT 2: Test Samples: 10100, Training Samples: 195000, Learning Rate=0.001, EPOCH=30

```
Training Accuracy: 82.00153 %

Validation Accuracy: 81.44 %

Test Accuracy: 87.821785 %
```





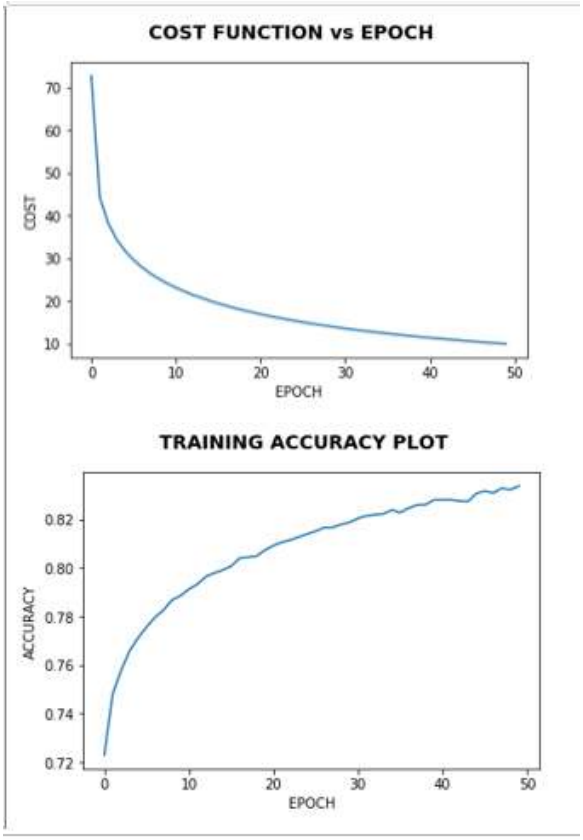
We saw the fluctuation in accuracy, therefore we make learning rate smaller. These are the better results and show that overfitting is not happening in this case as training accuracy and validation accuracy are both increasing..

RESULT 5: Test Samples: 10100, Training Samples: 195000, Learning Rate=0.001, EPOCH=50

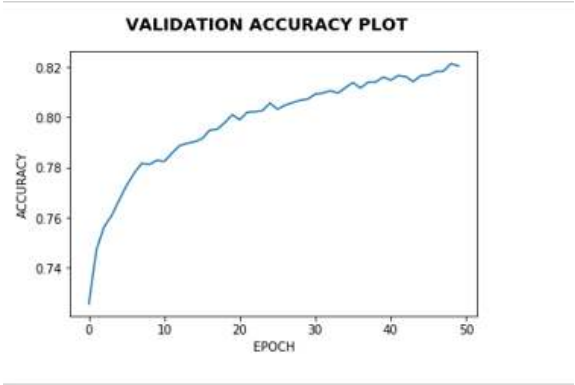
```
Training Accuracy: 83.379486 %

Validation Accuracy: 82.04 %

Test Accuracy: 88.9406 %
```



These are the final results with EPOCH=50. The result shows that even if EPOCHs are increased the training accuracy show increasing trend and so does the validation data accuracy.



HOW THE DATA IS READ:

Image files are read in jupyter notebook using "os" library. The two main functions of library used are **"os.listdir(folder name)" –** provides the list of directories and files inside the folder.

"os.path.join(folder name , file)" – joins the folder name and filename to make a path.

Using these functions, we can read all the image files inside the notebook.

METHODOLOGY OF THE WORK:

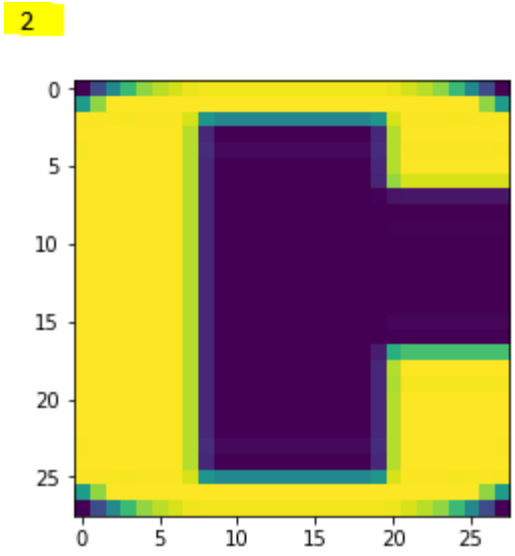
My methodology involves 2 steps.

1) Step 1: Loading the image files in the notebook and preprocess it for Logistic Regression using Tensorflow.
2) Step 2: Applying the Multinomial Logistic Regression on the loaded dataset.

Step 1: My own work

I performed following steps to read and normalize the data.

1) Reference work file for this step is "Tasks1-step1.py".
2) Using **for loop** and **"os"** library commands I was able to read the images and labels separately in the notebook. I used **"imageo"** library to read the image and **"matplotlib"** library to display the image. Once image is displayed than the label of that index is printed and it match the image which meant the image and labels are correctly loaded.
3) Normalization all the pixel values are divided by 255. All values range between 0 and 1.
4) "notMNIST_small" is used as "**test dataset**" while "notMNIST_large" was split between training and validation dataset using the **index split** of **numpy arrays.**
5) Each image read has shape of (28x28). I used numpy.reshape(arguments) function to make flat structure of (784,).

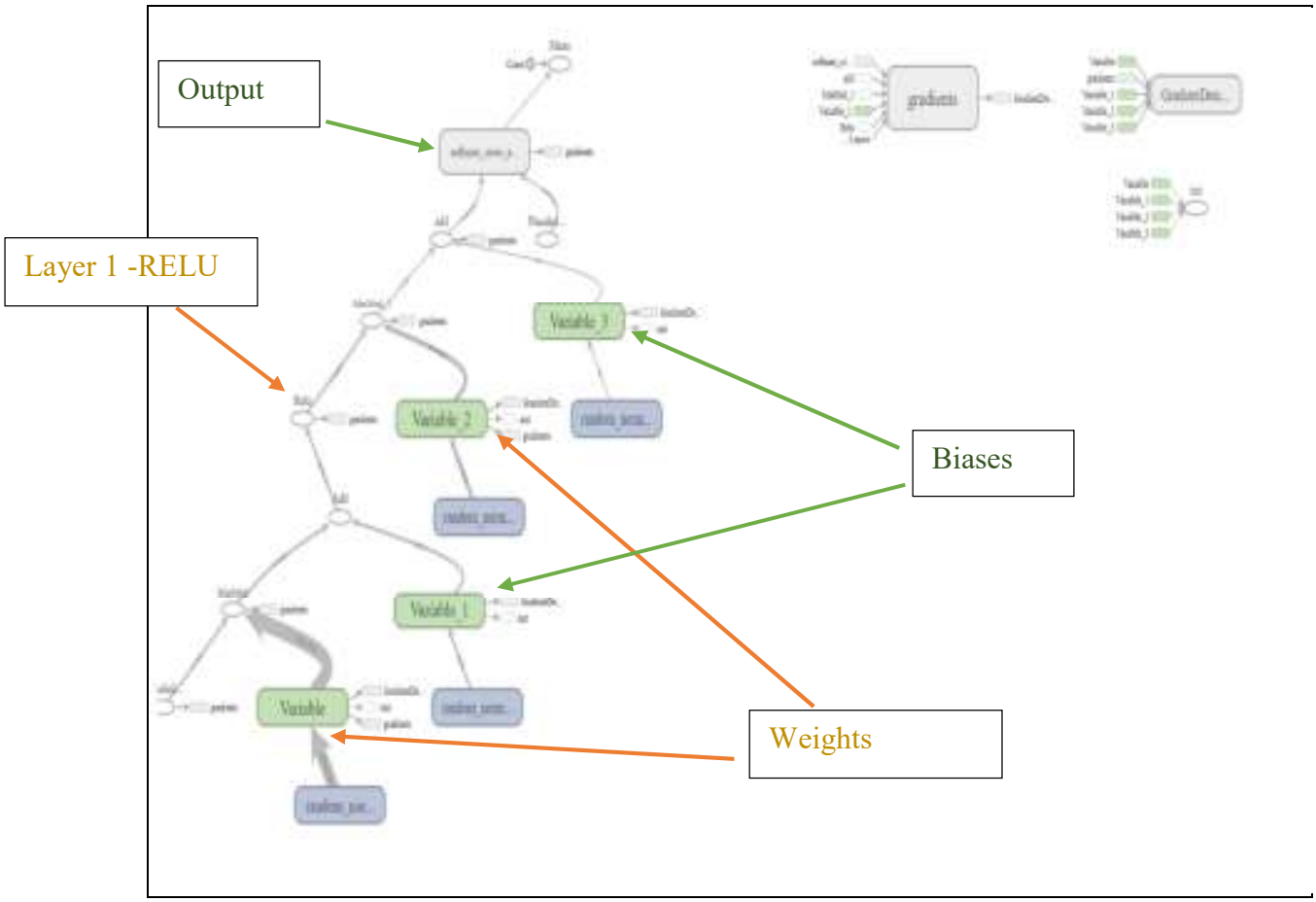


Step 1: Step 1 REDONE

1) Reference work file for this step is “Task1-Prepare_Data_20K.py” or notebook Task1-Prepare_Data_20K.ipynb.”
2) Since the dataset is inspired by MNIST dataset and this dataset already has a lot of support available in the Tensorflow. Therefore, I worked on converting the notMNIST dataset into MNIST format.
3) I took help from David’s work shared at https://raw.githubusercontent.com/davidflanagan/notMNIST-to-MNIST/master/convert_to_mnist_format.py
4) notMNIST dataset is converted to MNIST dataset. Due to limitations of the normal computer system I took 1010 samples from each label of “notMNIST_small” and 20,000 from each label of “notMNIST_large”. This dataset is now bigger than the MNIST dataset.
5) 4 *.ubyte files are created which are than zipped into *.gz format. This format is readable using Tensorflow “input_data” routine.
6) It brought efficiency in the program “input_data” splits the training data into training and validation data. It also converts labels into 1-hot encodings. The data read is in format of flat matrix and it is also normalized.
7) We have to reshaped flat matrix of image into 28x28 matrix to display the image.
8) We can use the MNIST routines on notMNIST data to train our model and do predictions.

Step 2: NEURAL NETWORK

1) Reference work file for this step is “Task2-Multinomial+Logistic+Regression.py” or Task2-Multinomial+Logistic+Regression.ipynb”
2) Tensorflow works differently than normal python functions in terms of how evaluations are done.
3) In Tensorflow we first create a model of computation which shows the flow of the computation.
4) Once our model is ready than we initialize a Tensorflow session in which all the computations are done.
5) Below image shows a graphical view of our model



6) We can change the hyper parameters before the start of the training
7) Parameters (Weights, Biases) are computed and updated during stochastic gradient descent.