CSE 598 - Data Intensive Machine Learning

Spring 2020

Assignment 1 :

Handwritten Digit Recognition using TensorFlow and MNIST dataset

Prof. Jia Zou

Arunima Mookherjee

(ASU ID: 1217178976)

Arizona State University Computer Science and Engineering

1. Introduction

Our task in the following assignment is to implement logistic regression and implement a deep neural network to classify handwritten digits of the MNIST dataset. Our data set consists of 55000 training samples, 5000 validation sample and 10000 test samples. Each greyscale image of 28*28 size. Each image is labelled from 0 to 9. Each image is flattened to make our feature vector of length 784.

2. Task: Building the model

2.1 Implement handwriting recognition using Logistic Regression

Logistic Regression is also known as the Sigmoid Function. It is a classification algorithm. Here, we initialize the model with batch size, number of epochs and the size of training and test set.

batch_size = 10000

n_epochs = 30

n_train = 60000

n_test = 10000

We need to create a weight matrix and initialized by choosing random values from truncated gaussian distribution and bias as 0. Then build the model using

```
logits = tf.matmul(img, w) + b
```

This will do a matrix multiplication of the training sample and the weights, thereafter, adding bias to the resulting matrix.

Loss is calculated using the softmax function using the predicted labels and the actual labels, bias adjusted and the model is trained again. Along with loss, we also need to initialize the optimizer, to minimize the loss function. Here, in our case we use AdamOptimizer.

Figure 1 shows us how our model looks on the tensorboard.

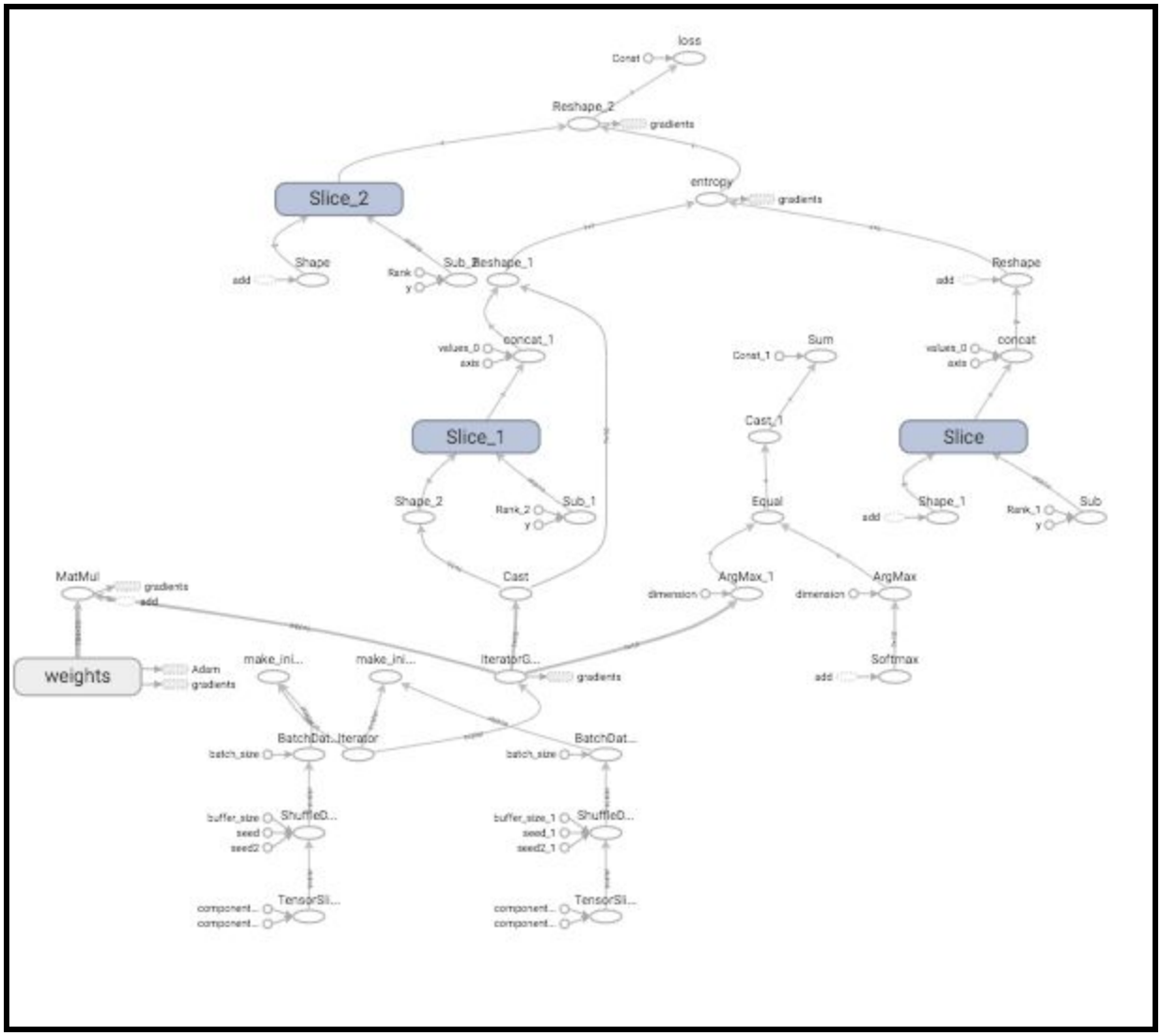

Figure 1

2.1 Implement handwriting recognition using Deep Neural Network

The deep neural network model will contain 2 hidden layers, and one output layer. The input layer will contain 784 neurons, as each image is 28x28, such 784 neuron corresponds to each pixel of an image.

no_hidden_layer1 = 500

no_hidden_layer2 = 120

no_output_layer = 10

10 nodes of the output layer are labels are from 0 to 9. (Ex for 1, [1,0,0,0,0,0,0,0,0,0], for 2, [0,2,0,0,0,0,0,0,0,0], etc)
Similarly like we did for logistic regression, we need to create a weight matrix. It is essentially a 2D tensor which contains the mappings between each input and neuron. It needs to be initialized by choosing random values from truncated gaussian distribution with standard deviation 2/ sqrt(number_of_inputs). This helps the model to converge relatively faster.
After this, I initialized the bias as 0. Then, calculate the result matrix, by calculating the matrix multiplication on the training data and wights, then finally adding the bias to it. Finally, I chose the activation function, ‘swish’.

In the next step, we need to formulate a cross entropy function. In this case, I chose, sparse_soft max_cross_entropy_with_logits. After this, I used Proximal Gradient Descent Optimizer to minimize the cost function. Learning rate will be the only input to this cost minimizer function.
Figure 2 shows us how our model looks on tensorboard.

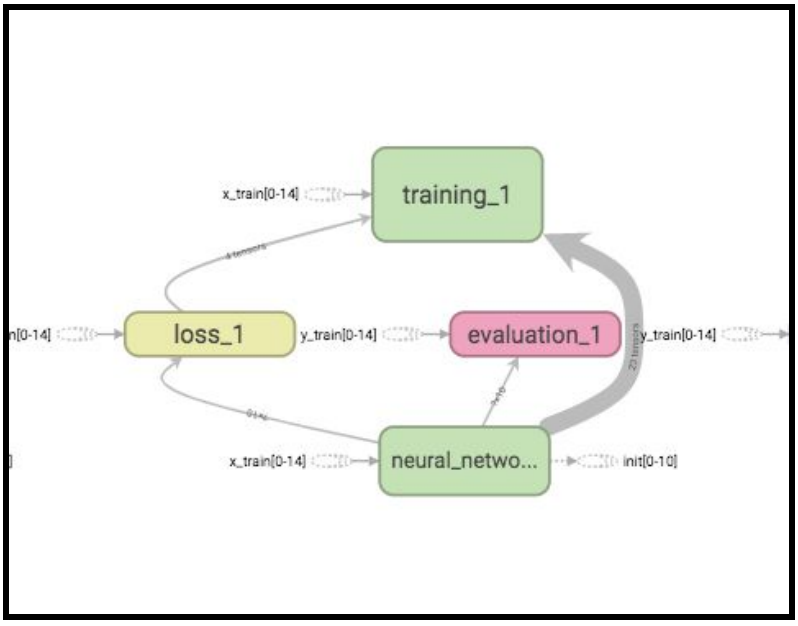

Figure 2: Epoch1 of DNN

3. Accuracy and Analysis

For 30 epocs, the predicted accuracy of Task1, i.e. prediction using Logistic Regression, came to be on test data was **92.72%** . Figure 3 shows us average loss for each epoch of Logistic Regression.

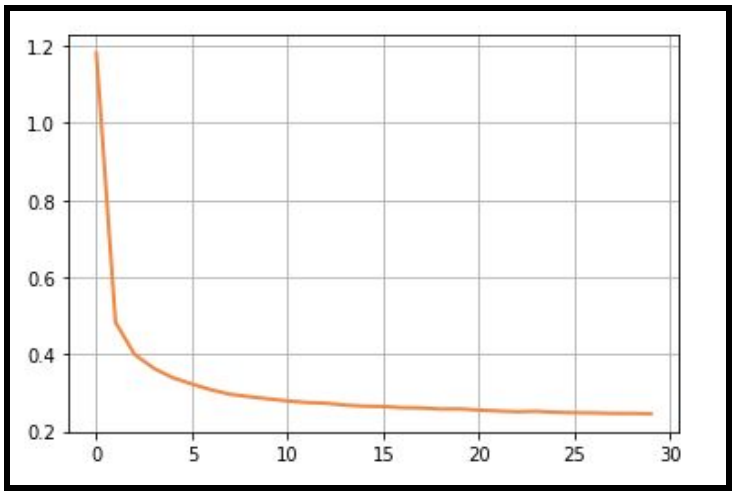

Figure 3: Average loss for each epoch

```
Average loss epoch 0: 1.1748062173525493
Average loss epoch 1: 0.48209670682748157
Average loss epoch 2: 0.3979124327500661
Average loss epoch 3: 0.3625015467405319
Average loss epoch 4: 0.3363356540600459
Average loss epoch 5: 0.3191031515598297
Average loss epoch 6: 0.3083753089110057
Average loss epoch 7: 0.29703561464945477
Average loss epoch 8: 0.2889966865380605
Average loss epoch 9: 0.2830572972695033
Average loss epoch 10: 0.2769399682680766
Average loss epoch 11: 0.2750966896613439
Average loss epoch 12: 0.2704845418532689
Average loss epoch 13: 0.26836539804935455
Average loss epoch 14: 0.2660677085320155
Average loss epoch 15: 0.2619662806391716
Average loss epoch 16: 0.26072441786527634
Average loss epoch 17: 0.25874633342027664
Average loss epoch 18: 0.2569971481959025
Average loss epoch 19: 0.25704872111479443
Average loss epoch 20: 0.25531497846047085
Average loss epoch 21: 0.2524464577436447
Average loss epoch 22: 0.25156428416570026
Average loss epoch 23: 0.2494994377096494
Average loss epoch 24: 0.24977762003739676
Average loss epoch 25: 0.2487749680876732
Average loss epoch 26: 0.24759364873170853
Average loss epoch 27: 0.24635187288125357
Average loss epoch 28: 0.24568271388610205
Average loss epoch 29: 0.24548024187485376
Total time: 87.23128819465637 seconds
Accuracy 0.9272
```

Figure 4: Screenshot of Logistic Regression

For Task2, i.e prediction on test dataset using Deep Neural Network was **98.09%**. As

you can see in the graph in Figure 4, the blue line shows the accuracy of the training sample, the orange line shows the accuracy of the test sample.

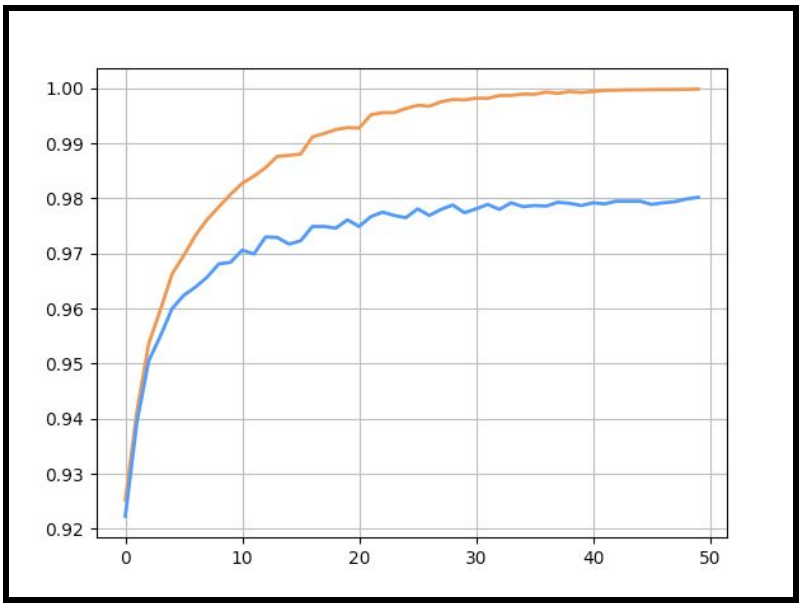

Figure 5: Epoch accuracy for DNN

```
print(accuracy_train_,"\n", accuracy_test_)
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
Epoch 0: Training Accuracy: 0.92489093542099,  Testing accuracy: 0.925000011920929
Epoch 1: Training Accuracy: 0.9395090937614441,  Testing accuracy: 0.9358000159263611
Epoch 2: Training Accuracy: 0.953745424747467,  Testing accuracy: 0.9484000205993652
Epoch 3: Training Accuracy: 0.9589454531669617,  Testing accuracy: 0.9531999826431274
Epoch 4: Training Accuracy: 0.9659090638160706,  Testing accuracy: 0.9593999981880188
Epoch 5: Training Accuracy: 0.9698727130889893,  Testing accuracy: 0.9631999731063843
Epoch 6: Training Accuracy: 0.972672700881958,  Testing accuracy: 0.965499997138977
Epoch 7: Training Accuracy: 0.9766545295715332,  Testing accuracy: 0.9678999781608582
Epoch 8: Training Accuracy: 0.9783999919891357,  Testing accuracy: 0.9700000286102295
Epoch 9: Training Accuracy: 0.9804727435112,  Testing accuracy: 0.9710000157356262
Epoch 10: Training Accuracy: 0.9817273020744324,  Testing accuracy: 0.9715999960899353
Epoch 11: Training Accuracy: 0.9831636548042297,  Testing accuracy: 0.9711999893188477
Epoch 12: Training Accuracy: 0.9858909249305725,  Testing accuracy: 0.9725000262260437
Epoch 13: Training Accuracy: 0.9867454767227173,  Testing accuracy: 0.9745000004768372
Epoch 14: Training Accuracy: 0.9878727197647095,  Testing accuracy: 0.9739000201225281
Epoch 15: Training Accuracy: 0.9887818098068237,  Testing accuracy: 0.9757000207901001
Epoch 16: Training Accuracy: 0.9911817908287048,  Testing accuracy: 0.9764000177383423
Epoch 17: Training Accuracy: 0.990781843662262,  Testing accuracy: 0.9745000004768372
Epoch 18: Training Accuracy: 0.9921636581420898,  Testing accuracy: 0.9768000245094299
Epoch 19: Training Accuracy: 0.9919636249542236,  Testing accuracy: 0.9765999913215637
Epoch 20: Training Accuracy: 0.9935454726219177,  Testing accuracy: 0.9779000282287598
Epoch 21: Training Accuracy: 0.994527280330658,  Testing accuracy: 0.9775999784469604
Epoch 22: Training Accuracy: 0.9939636588096619,  Testing accuracy: 0.9771999716758728
Epoch 23: Training Accuracy: 0.9951817989349365,  Testing accuracy: 0.9786999821662903
Epoch 24: Training Accuracy: 0.9964181780815125,  Testing accuracy: 0.9785000085830688
Epoch 25: Training Accuracy: 0.9956545233726501,  Testing accuracy: 0.9782999753952026
Epoch 26: Training Accuracy: 0.9966363906860352,  Testing accuracy: 0.9782000184059143
Epoch 27: Training Accuracy: 0.9975454807281494,  Testing accuracy: 0.9785000085830688
Epoch 28: Training Accuracy: 0.9975454807281494,  Testing accuracy: 0.9793000221252441
Epoch 29: Training Accuracy: 0.9978545308113098,  Testing accuracy: 0.9789999723434448
Epoch 30: Training Accuracy: 0.998199999332428,  Testing accuracy: 0.9796000123023987
Epoch 31: Training Accuracy: 0.9970181584358215,  Testing accuracy: 0.9793000221252441
Epoch 32: Training Accuracy: 0.9984545707702637,  Testing accuracy: 0.9785000085830688
Epoch 33: Training Accuracy: 0.9986000061035156,  Testing accuracy: 0.9793000221252441
Epoch 34: Training Accuracy: 0.9989272952079773,  Testing accuracy: 0.9793000221252441
Epoch 35: Training Accuracy: 0.9990363717079163,  Testing accuracy: 0.9790999889373779
Epoch 36: Training Accuracy: 0.9991272687911987,  Testing accuracy: 0.9804999828338623
```

Figure 6: Google Colab screenshot for DNN

4. Estimated Time Spent on the Assignment

Approximation of the time spent on the assignment

Understanding the assignment and getting hands on with building models on tensorflow	2 hours
Setting up the environment	4 hours
Task1 building the model	30 minutes
Task1 - Training the model (for 50 epoch) [multiple iterations]	30 minutes
Task2 build the model	2 hours
Task2 - Train the model (100 epoch) [multiple iterations]	3 hours
Generating graphs on tensorboard	4 hours
Writing the report	2 hours
Total Time	18 hours

5. Description of Problem met

During the course of doing the assignment, I came across various issues.

A. Figuring out the configurations and setting up the environment on which running the code, as well as training it was feasible.
   Approach 1: I tried it on my local environment (macOS 10.12, 8GB RAM, 256 SSD), by reacting a virtual python environment. This caused a problem as I was unable to download the required version tensorflow==1.4.1. I also tried downloading tensorflow using .whl file, but it was rendered incompatible

   Approach 2 : On AWS Educate, using the free-tier resource, (Ubuntu 18.04 1GB RAM) when I tried to train my deep neural net model and I got a Segmentation fault (core dumped). This happens. It is an error indicating memory corruption. This happens when the file size of the input is bigger than 130kb. Therefore, moved on the the next approach

Approach 3 : Running an Ubuntu Docker container on top of the localhost. I was able to train and test the model using this approach, however, the training process was taking longer than expected with low accuracy. Moreover, it was also causing system freezes.

Approach 4: The last approach was to try it running on Google Colab. This method was by far the best approach.

B. Tweaking parameters to increase the accuracy. Specially for the task2, it was difficult to figure out which parameters to tweak to get the best result.

C. Tweaking the learning rate: Changing the learning caused the accuracy to drop down to 68.3% for 20 epochs.

D. Tweaking the batch side: Changing the batch size from 60 to any other value caused the accuracy to drop to plummet.

E. Tweaking the number of neurons in the hidden layer 1 and hidden layer 2: After a number of hit and trials, I decided to go ahead with 500 neurons in hidden_layer1 and 120 neurons in hidden_layer2 which gave me an accuracy of 98.09%.