Handwritten Digit Recognition using Tensorflow and MNIST dataset

Objective:

The aim of this project is to classify images of numbers as the digit that it represents without the aid of human eyes. We are going to teach a machine how to predict that number by showing it many samples that are already labelled then we are going to ask the machine to predict the values on unlabelled images to test its accuracy. The goal is to reach highest accuracy possible in the least amount of time so that we have a sentient being which can look at a sudoku puzzle and extract the numbers from it or read the number plate of cars etc.
The algorithms that we use to train the machine with a set of examples or experience without hardcoding the rules are called machine learning algorithms. The following report consists of implementations of several such machine learning algorithms. I used my own Convolutional Neural Network architecture which gave a accuracy of **97.94** in just 10 epochs.The models are written in Tensorflow from scratch without any higher level API like keras. MNIST handwritten digits dataset from Yann LeCun website is used for training and testing.

Using logistic regression to classify image data

Model:

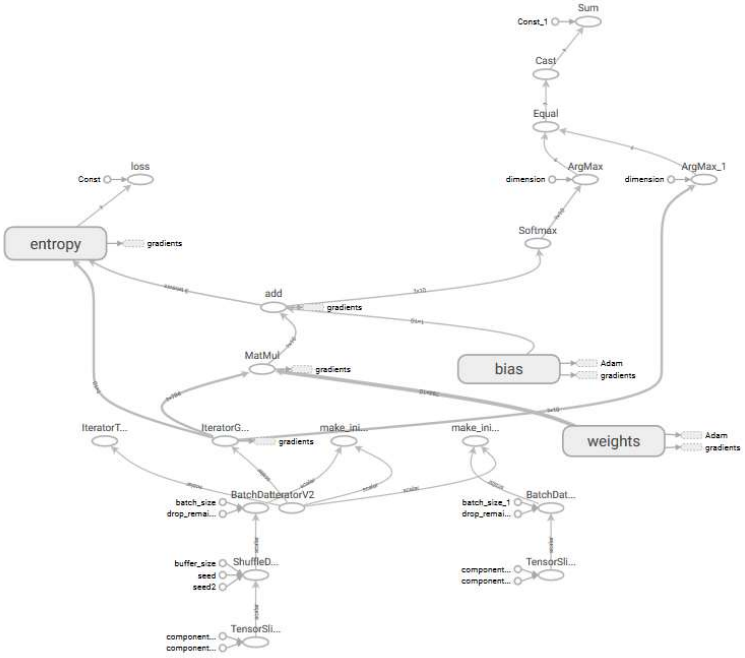
Simple Logistic regression with 784 inputs and 10 outputs. The code resides in a1a.py. The ouput logits from wX+b are connected to softmax activation function. Used mini batch stochastic gradient descent with batch size of 128 and adam optimizer with learning rate of 0.01 to minimize the cross entropy loss.

Parameters

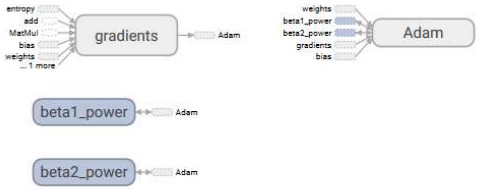
learning_rate = 0.01
batch_size = 128
n_epochs = 30
n_train = 60000
n_test = 10000

TensorBoard Graph:

Main Graph



Auxiliary Nodes



Accuracy Results:

Average loss epoch 0: 0.36615645521601964
Average loss epoch 1: 0.2961393812021544
Average loss epoch 2: 0.28516315083171045
Average loss epoch 3: 0.2784664580641791
Average loss epoch 4: 0.27820061214788017
Average loss epoch 5: 0.27183693950605947
Average loss epoch 6: 0.2702045906595019
Average loss epoch 7: 0.2686337898463704
Average loss epoch 8: 0.2693879988477674
Average loss epoch 9: 0.26581296919043673
Average loss epoch 10: 0.26332885520749316
Average loss epoch 11: 0.26326023283046346
Average loss epoch 12: 0.26261485452915345
Average loss epoch 13: 0.26556264740783114
Average loss epoch 14: 0.2616427352435367
Average loss epoch 15: 0.2609298462784568
Average loss epoch 16: 0.2573050965923209
Average loss epoch 17: 0.260258146808591
Average loss epoch 18: 0.2578184757641582
Average loss epoch 19: 0.2584787118400252
Average loss epoch 20: 0.25492706498087836
Average loss epoch 21: 0.25654620417328766
Average loss epoch 22: 0.25535715123595193

Average loss epoch 23: 0.2567003796266955
Average loss epoch 24: 0.255083247118218
Average loss epoch 25: 0.25493412421539774
Average loss epoch 26: 0.25597158063982806
Average loss epoch 27: 0.2531915844526402
Average loss epoch 28: 0.2551793723258861
Average loss epoch 29: 0.2560468336870504
Total time: 18.58875298500061 seconds
Final Testing Accuracy after 30 epochs is 91.64%

Using Deep Neural Networks to classify image data

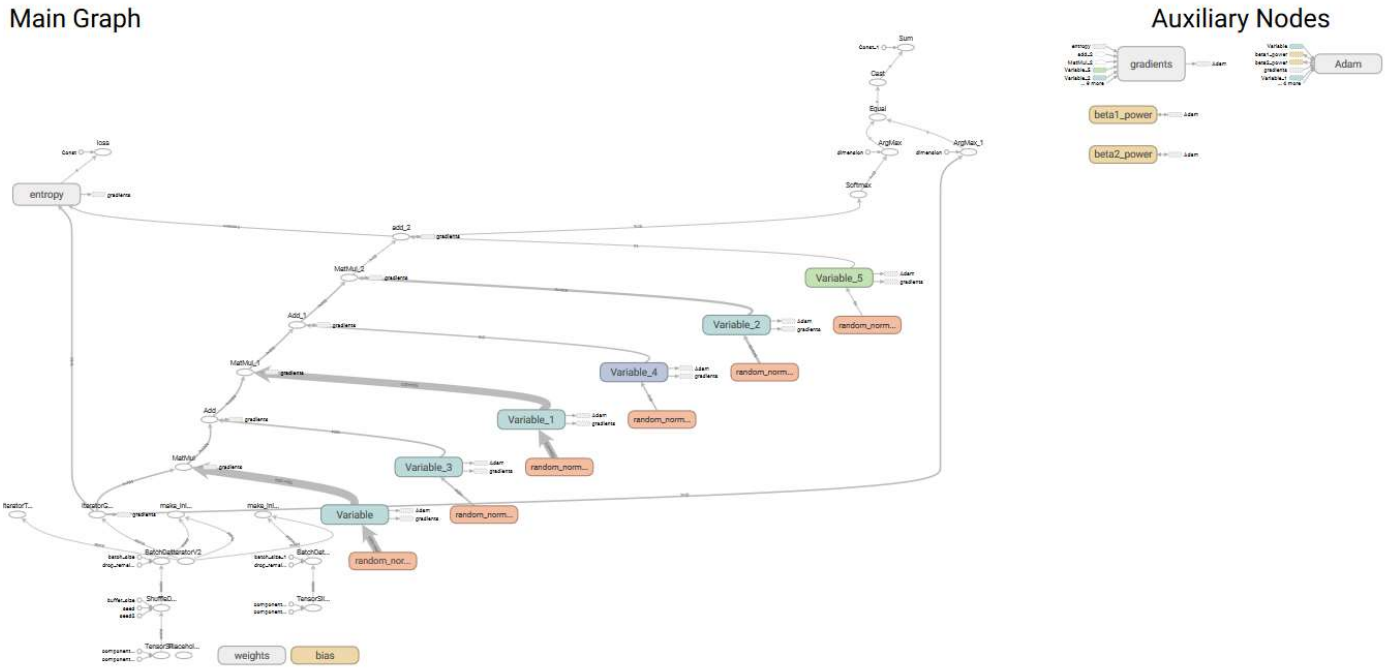
Model:

Used a deep neural network to classify the MNIST hand written dataset. The code resides in a1c.py. The deep neural network consists of 2 hidden layers and one output layer. The first hidden layer has 1024 neurons the second hidden layer consists of 256 neurons and the final output layer consists of 10 neurons. The output logits from output layer are connected to softmax activation function. Used stochastic gradient descent with batch size of 128 and adam optimizer with learning rate of 0.005 to minimize the cross entropy loss.

Parameters

learning_rate = 0.005
batch_size = 64
n_epochs = 32
n_train = 60000
n_test = 10000

TensorBoard Graph:



Accuracy Results:

Training Average loss epoch 0: 513.7043691701667
Testing Accuracy epoch 0: 89.53999999999999%
Training Average loss epoch 1: 288.1595486119736
Testing Accuracy epoch 1: 88.11%
Training Average loss epoch 2: 199.32467073728873
Testing Accuracy epoch 2: 86.39%
Training Average loss epoch 3: 140.19646526381027
Testing Accuracy epoch 3: 88.07000000000001%
Training Average loss epoch 4: 104.04333844961123
Testing Accuracy epoch 4: 88.07000000000001%
Training Average loss epoch 5: 75.77764784790749
Testing Accuracy epoch 5: 88.12%
Training Average loss epoch 6: 57.74088625852452
Testing Accuracy epoch 6: 87.19%
Training Average loss epoch 7: 43.858268483295
Testing Accuracy epoch 7: 89.4%
Training Average loss epoch 8: 31.256696361719175
Testing Accuracy epoch 8: 88.11%
Training Average loss epoch 9: 23.632596603798312
Testing Accuracy epoch 9: 86.74%
Training Average loss epoch 10: 18.06407882848451
Testing Accuracy epoch 10: 88.03999999999999%
Training Average loss epoch 11: 14.582898021021554
Testing Accuracy epoch 11: 87.03999999999999%

Training Average loss epoch 12: 13.48470704208973
Testing Accuracy epoch 12: 87.39%
Training Average loss epoch 13: 13.063212687331577
Testing Accuracy epoch 13: 84.09%
Training Average loss epoch 14: 12.775556096018747
Testing Accuracy epoch 14: 87.75%
Training Average loss epoch 15: 12.225312802888626
Testing Accuracy epoch 15: 84.07%
Training Average loss epoch 16: 11.796422531646352
Testing Accuracy epoch 16: 88.72%
Training Average loss epoch 17: 11.17209043953308
Testing Accuracy epoch 17: 84.89999999999999%
Training Average loss epoch 18: 10.846656801430834
Testing Accuracy epoch 18: 84.24000000000001%
Training Average loss epoch 19: 10.42778755309017
Testing Accuracy epoch 19: 85.45%
Training Average loss epoch 20: 10.410528026140014
Testing Accuracy epoch 20: 86.99%
Training Average loss epoch 21: 9.936287257079849
Testing Accuracy epoch 21: 86.65%
Training Average loss epoch 22: 9.969835560456957
Testing Accuracy epoch 22: 87.35000000000001%
Training Average loss epoch 23: 9.743539109063702
Testing Accuracy epoch 23: 84.77%
Training Average loss epoch 24: 8.924752203635242
Testing Accuracy epoch 24: 88.56%
Training Average loss epoch 25: 9.310567932350692
Testing Accuracy epoch 25: 87.74%
Training Average loss epoch 26: 8.750125947248105
Testing Accuracy epoch 26: 87.59%
Training Average loss epoch 27: 8.312265286568639
Testing Accuracy epoch 27: 86.97%
Training Average loss epoch 28: 8.380556426769079
Testing Accuracy epoch 28: 86.49%
Training Average loss epoch 29: 7.98817785522619
Testing Accuracy epoch 29: 85.05%
Training Average loss epoch 30: 7.64589896153572
Testing Accuracy epoch 30: 87.3%
Training Average loss epoch 31: 7.507957710464334
Testing Accuracy epoch 31: 86.53999999999999%
Total time: 172.84515070915222 seconds
Final Testing Accuracy after 32 epochs is 86.53999999999999%

Using Convolutional Neural Networks to classify image data

Model:

Used a Convolutional neural network to classify the MNIST hand written dataset. The code resides in a1b.py. The convolutional neural network is a combination of 3 convolutional layers, 3 max pool layers, 1 hidden neural and finally one output layer.

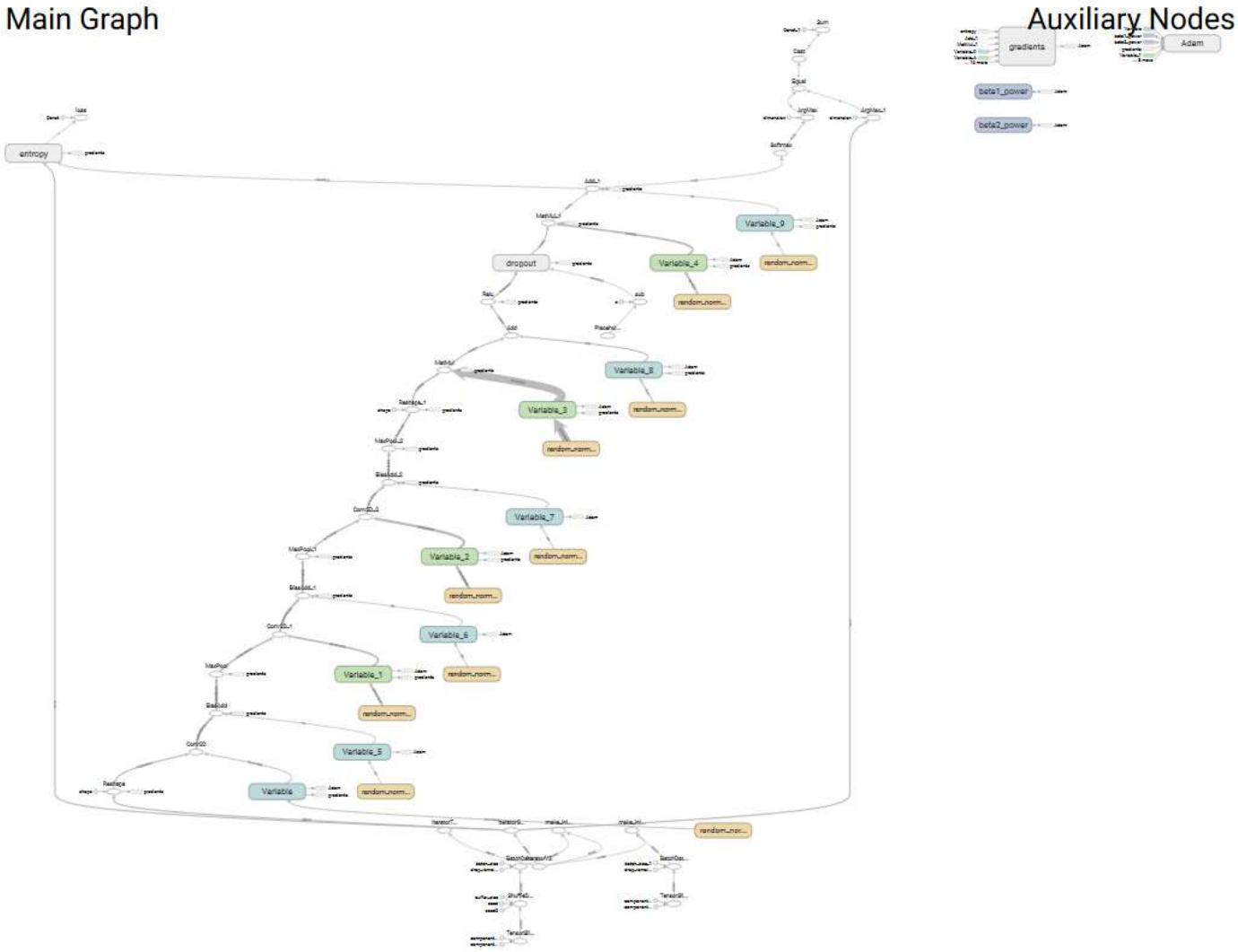
Layer Name	Type	Kernel Size	Stride	Padding	Input layers/Neurons	Output layers/Neurons
conv1	Convolutional Layer	3 X 3	1 X 1	SAME	1	16
max_pool1	Max Pool Layer	2 X 2	2 X 2	SAME	16	16
conv2	Convolutional Layer	3 X 3	1 X 1	SAME	16	32
max_pool2	Max pool Layer	2 X 2	2 X 2	SAME	32	32
conv3	Convolutional Layer	3 X 3	1 X 1	SAME	32	64
max_pool3	Max pool Layer	2 X 2	2 X 2	VALID	64	64
hiddenlayer	Neural Layer	-	-	-	2304	512
outputlayer	Neural Layer	-	-	-	512	10

ReLU activation function and dropout of 0.9(keep probability) is applied to the output of hidden layer to prevent overfitting.
The output logits from output layer are connected to softmax activation function. Used stochastic gradient descent with batch size of 128 and adam optimizer with learning rate of 0.005 to minimize the cross entropy loss.

Parameters:
learning_rate = 0.005
batch_size = 128
n_epochs = 10
n_train = 60000

n_test = 10000
num_classes = 10

TensorBoard Graph:

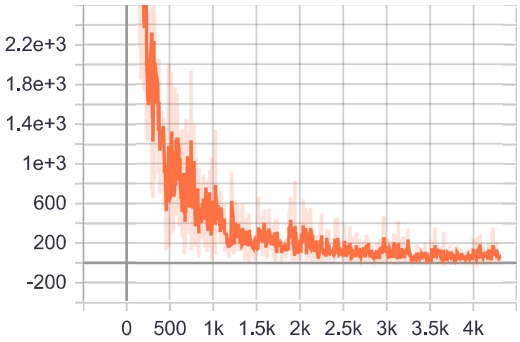


Accuracy:

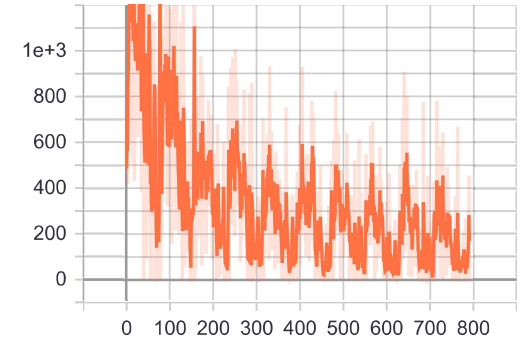
Training Average loss epoch 0: 10092.289313595795
Testing Accuracy epoch 0: 95.34%
Training Average loss epoch 1: 954.8375116037768
Testing Accuracy epoch 1: 95.66%
Training Average loss epoch 2: 493.0808473631393
Testing Accuracy epoch 2: 96.87%
Training Average loss epoch 3: 306.5842036934786
Testing Accuracy epoch 3: 97.04%
Training Average loss epoch 4: 207.59633458159692
Testing Accuracy epoch 4: 97.25%
Training Average loss epoch 5: 166.5334325080694
Testing Accuracy epoch 5: 97.76%
Training Average loss epoch 6: 127.31241749608239

Testing Accuracy epoch 6: 97.61999999999999%
Training Average loss epoch 7: 120.76717296431578
Testing Accuracy epoch 7: 97.67%
Training Average loss epoch 8: 97.13484195450017
Testing Accuracy epoch 8: 97.50999999999999%
Training Average loss epoch 9: 99.9249820900778
Testing Accuracy epoch 9: 97.94%
Total time: 189.2675199508667 seconds
Final Testing Accuracy after 10 epochs is 97.94%

Loss of Training Samples



Loss of Testing Samples



Problems Faced:

Task 1. Using logistic regression to classify image data

Had several issues running the code in windows. Tensorflow v1.4.1 is not available so downloaded the v1.15.0. The dataset I downloaded has different compressions in windows so had to tweak some things to make it work. I made both task1 and task2 run in both windows 10 and ubuntu 20 with some tweaks. a1a.py had an error initially that the *data* variable is used without initialization.

Task 2. Improve the model of Task 1.

First implemented a Deep Neural Network expecting the accuracy to increase from naive logistic regression even though the training loss is decreasing the testing accuracy never crossed 90%. I think that the model was overfitting to training samples. I wanted to achieve the highest accuracy so I moved on to implement a Convolutional Neural Network. Even here overfitting was a problem so I used the dropout technique which

essentialy zeros out the output values of a layer with the specified probability. I had to try different model architectures to arrive at my model which has an accuracy of 97.94%. Since I implemented my own architecture I had to calculate the output size at each stage so that I can initialize the weights of the hidden neural layer. I think my model is overcomplicated by having 3 convolutional and 3 max pool layers. I bet there are models out there which are less complex and achieved a better accuracy.

Time Taken:

Task 1. Using logistic regression to classify image data

It took me around 4 hours for task1.

Task 2. Improve the model of Task 1

It took me around 6 hours for task2.