Machine Learning Engineer Nanodegree

Capstone Project

Tushar Bansal
August 20th, 2017

I. Definition

Project Overview

A handwritten digit recognition system was used in a demonstration project to visualize artificial neural networks,

MNIST is a widely used dataset for the hand-written digit classification task. It consists of 70,000 labelled 28x28 pixel greyscale images of hand-written digits. The dataset is split into 60,000 training images and 10,000 test images. There are 10 classes (one for each of the 10 digits). The task at hand is to train a model using the 60,000 training images and subsequently test its classification accuracy on the 10,000 test images.

The human visual system is one of the wonders of the world. Consider the following sequence of handwritten digits:

Most people effortlessly recognize those digits as 504192. That ease is deceptive. In each hemisphere of our brain, humans have a primary visual cortex.

Neural networks approach the problem in a different way. The idea is to take a large number of handwritten digits, known as training examples,

and then develop a system which can learn from those training examples. In other words, the neural network uses the examples to automatically infer rules for recognizing handwritten digits. Furthermore, by increasing the number of training examples, the network can learn more about handwriting, and so improve its accuracy. So while I've shown just 100 training digits above, perhaps we could build a better handwriting recognizer by using thousands or even millions or billions of training examples.

Problem Statement

The goal in this is to take an image of a handwritten single digit and determine what that digit is. This dataset is containing number of images and their labels i.e. their respective numbers. This is a classification problem. This model will recognise the unseen digit with number (0-9).

First we load the dataset preprocess it and split it into training and testing sets.

Now after that we will use Deep Neural Networks to train our model. We will use Softmax and Gradient Descent Optimiser to train our model and TensorGraph for Visualisation.

Metrics

This is evaluated on the categorization accuracy of your predictions (the percentage of images you get correct).

Accuracy = Number of samples Predicted correctly / Total Number of samples

We use accuracy as it measures the correctness of model we need highest numbers of digits to be predicted correctly in this model.

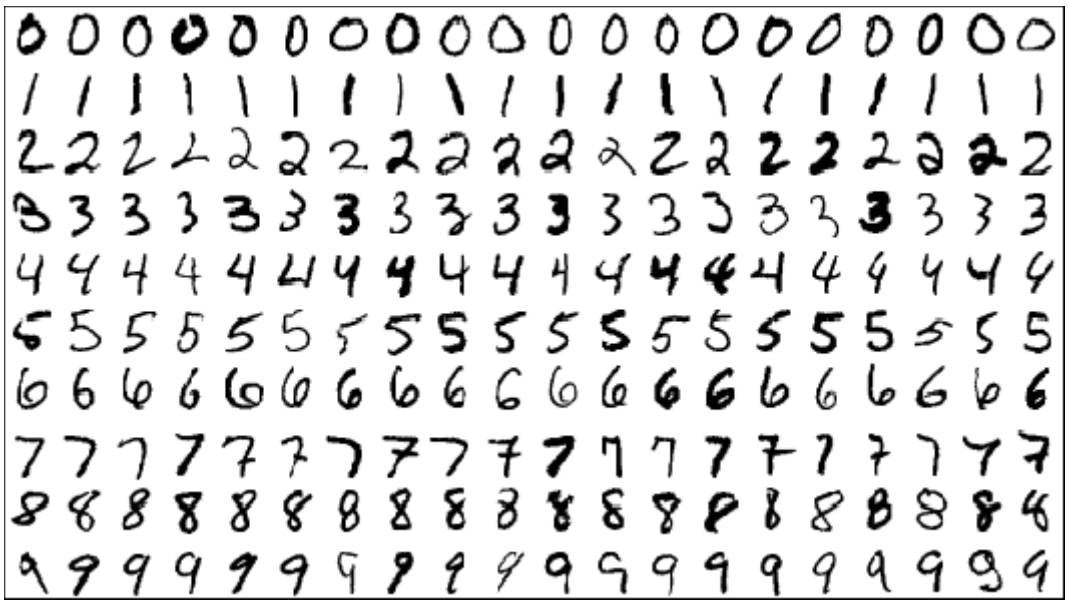
II. Analysis

Data Exploration

The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.

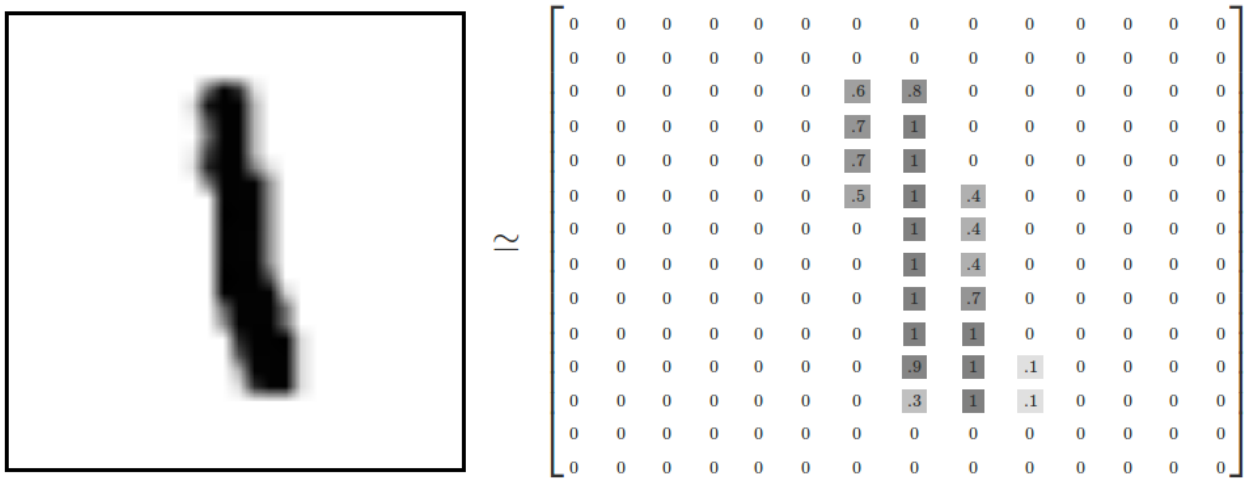
The MNIST data is split into three parts: 55,000 data points of training data (`mnist.train`), 10,000 points of test data (`mnist.test`), and 5,000 points of validation data (`mnist.validation`). This split is very important: it's essential in machine learning that we have separate data which we don't learn from so that we can make sure that what we've learned actually generalizes!

Exploratory Visualization



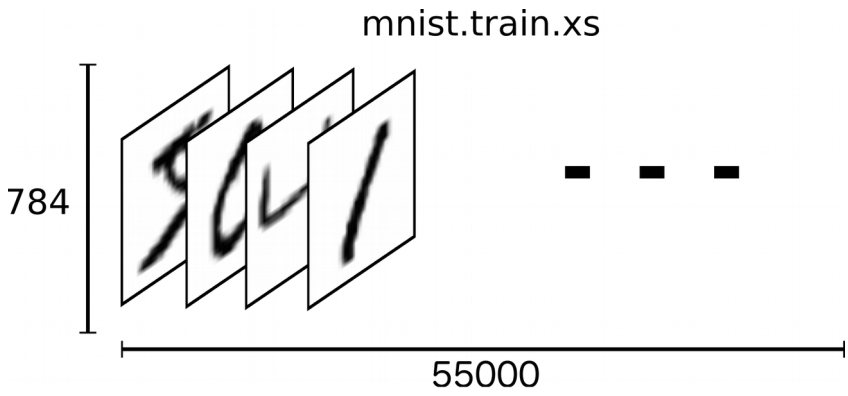
Every MNIST data point has two parts: an image of a handwritten digit and a corresponding label. We'll call the images "x" and the labels "y". Both the training set and test set contain images and their corresponding labels; for example the training images are `mnist.train.images` and the training labels are `mnist.train.labels`.

Each image is 28 pixels by 28 pixels. We can interpret this as a big array of numbers:

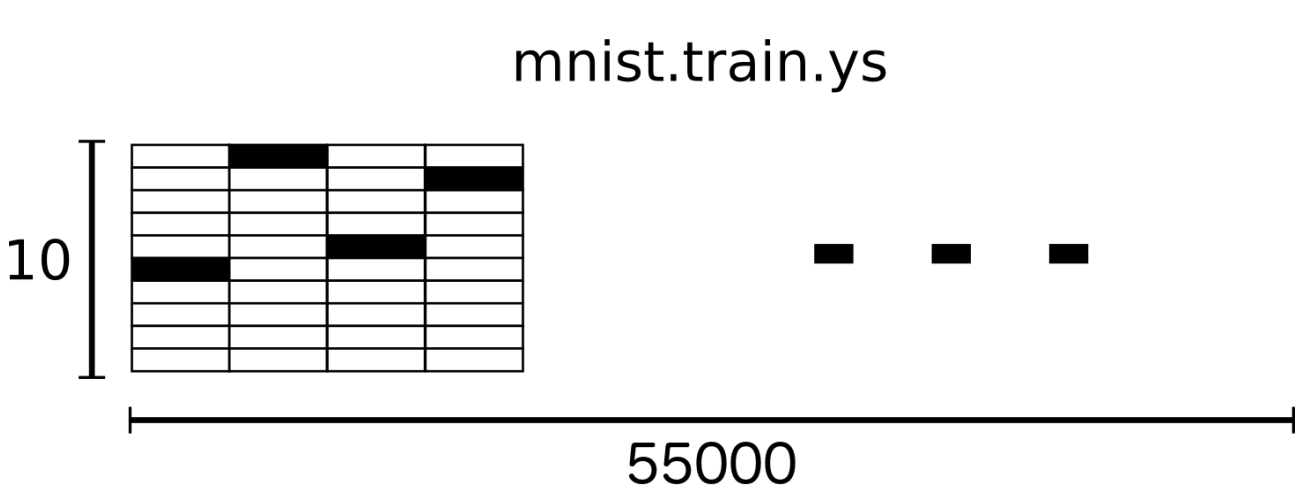


We can flatten this array into a vector of 28x28 = 784 numbers. It doesn't matter how we flatten the array, as long as we're consistent between images. From this perspective, the MNIST images are just a bunch of points in a 784-dimensional vector space, with a very rich structure.

The result is that `mnist.train.images` is a tensor (an n-dimensional array) with a shape of `[55000, 784]`. The first dimension is an index into the list of images and the second dimension is the index for each pixel in each image. Each entry in the tensor is a pixel intensity between 0 and 1, for a particular pixel in a particular image.



Each image in MNIST has a corresponding label, a number between 0 and 9 representing the digit drawn in the image.



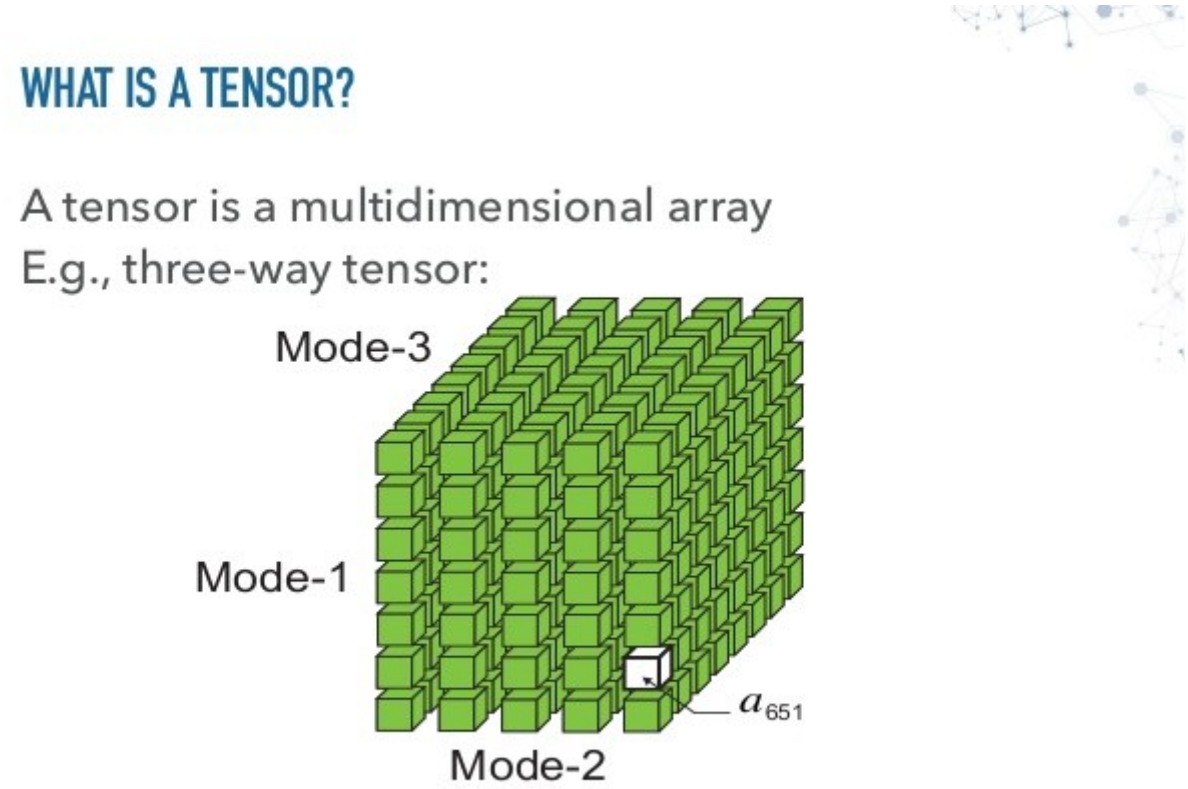
Algorithms and Techniques

Every digit will lie between zero and nine so we know we have 10 possibilities for each case. For example the digit is 0 so it will also have some probability that it is 8 or 9. but probability of besing 0 will be greatest.

We use Deep Neural Networks in this project.

Tensorflow Represents data in a Tensor. It takes input as a tensor and output as a tensor as well.

Tensor is a multidimensional array of numbers and and they flow between operations



Benchmark

I used the SVM with RBF kernel as my benchmark and several kernels on Kaggle competition SVM with RBF takes much more time to train than neural nets we can use several parameters with SVM_RBF but it takes much more time to train as compared to neural networks.

You can check the results here:

http://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html

I saw many of them on Kaggle competition

https://www.kaggle.com/c/digit-recognizer/kernels

III. Methodology

Data Preprocessing

We split the data into testing and training set

TRAIN_IMAGES = 'train-images-idx3-ubyte.gz'

TRAIN_LABELS = 'train-labels-idx1-ubyte.gz'

TEST_IMAGES = 't10k-images-idx3-ubyte.gz'

TEST_LABELS = 't10k-labels-idx1-ubyte.gz'

Implementation

We first Downloaded the input dataset, and split it into testing and training data then preprocess it using the data_dl.py script.

We need to set the Hyper parameters first i.e Learning Rate, training_iteration, batch_size and display_step.

We will create two placeholder operations first. And assign 784 dimension to x and 10 to y as x is an image with 28x28 (which are converted into 1D array) and y are labels.

Now we will set our Weights and biases and set value of them to zero.

Now we will implement our model logistic regression by matrix multiplication. We will also visualize weights and biases by creating summary operation. We wil also create cost funtion which will minimize errors during training by sung cross entropy option and also create scalar summary to visualize it.

We will use **Gradient Descent Optimiser to** train the data

In most Supervised Machine Learning problems we need to define a model and estimate its parameters based on a training dataset. A popular and easy-to-use technique to calculate those parameters is to minimize model's error with Gradient Descent. The Gradient Descent estimates the weights of the model in many iterations by minimizing a cost function at every step.

```
Repeat until convergence {

     Wj = Wj - λ θF(Wj)/θWj

}
```

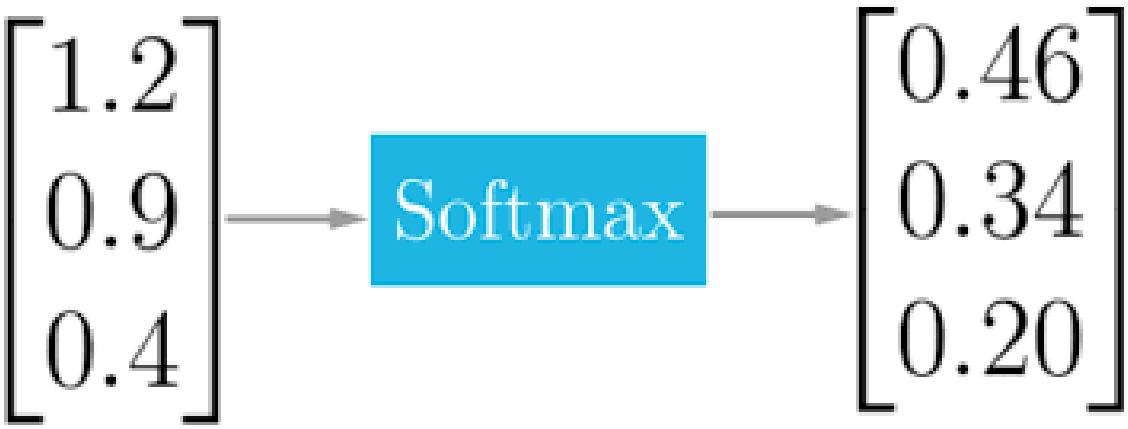
Where Wj is one of our parameters (or a vector with our parameters), F is our cost function (estimates the errors of our model), θF(Wj)/θWj is its first derivative with respect to Wj and λ is the learning rate.

If our F is monotonic, this method will give us after many iterations an estimation of the Wj weights which minimize the cost function. Note that if the derivative is not monotonic we might be trapped to local minimum. In that case an easy way to detect this is by repeating the process for different initial Wj values and comparing the value of the cost function for the new estimated parameters.

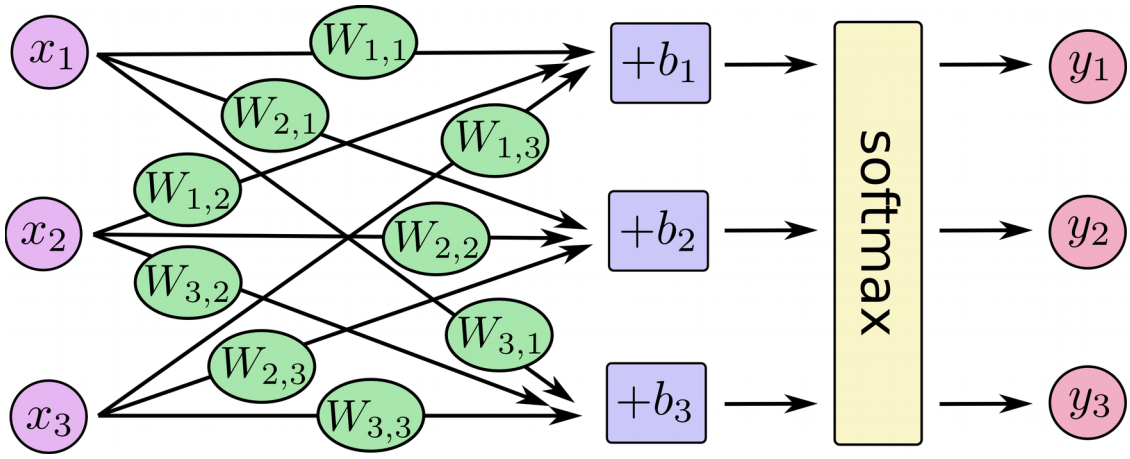
In order for Gradient Descent to work we must set the λ (learning rate) to an appropriate value. This parameter determines how fast or slow we will move towards the optimal weights. If the λ is very large we will skip the optimal solution. If it is too small we will need too many iterations to converge to the best values. So using a good λ is crucial.

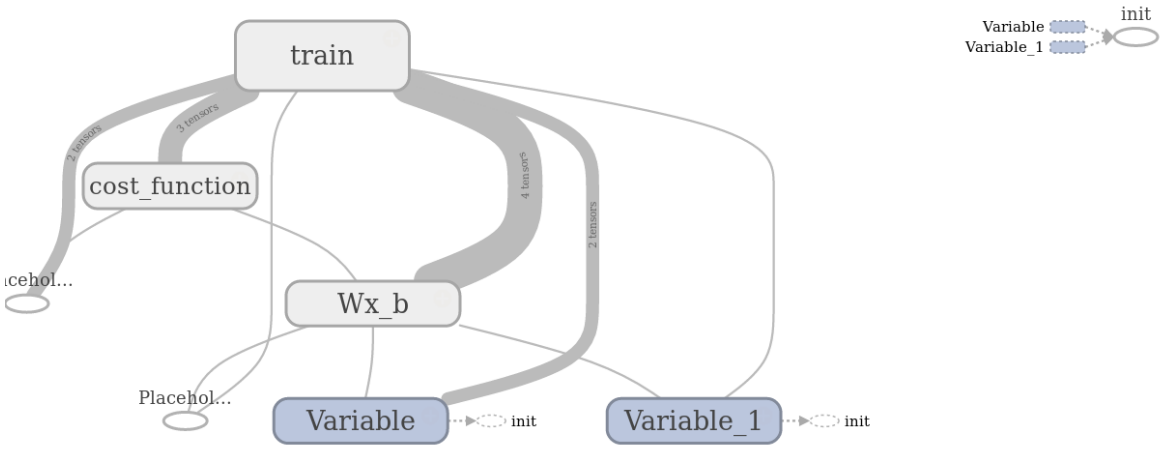
Softmax

Softmax regression (or multinomial logistic regression) is a generalization of logistic regression to the case where we want to handle multiple classes. In logistic regression we assumed that the labels were binary: $y(i) \in \{0,1\}$. We used such a classifier to distinguish between two kinds of hand-written digits. Softmax regression allows us to handle $y(i) \in \{1, \ldots, K\}$ where K is the number of classes.



$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$





We will initialize all of our variables we will launch our graph in an Tensorflow session. And set the location to the summary writer which we will later visualize using tensorboard.

Now we will set for loop for number of iterations and we will also print our average cost on every iteration so that we can check that the model is actually learning or not.

We will compute our batch size and start training our model over each data. After training we will fit the batch data in our Gradient Descent Algorithm for the back propogarion and computing loss and writing logsfor each iterations by using summary writer.

We will display the error logs in terminal for each display step.

Now at the end we will test the model by comparing the model values to the output values. And Calculate the accuracy.

Refinement

We will tune our hyper parameters to tune out model and gain more accuracy. In final i set them to

learning_rate=0.01

training_iteration = 50

batch_size = 64

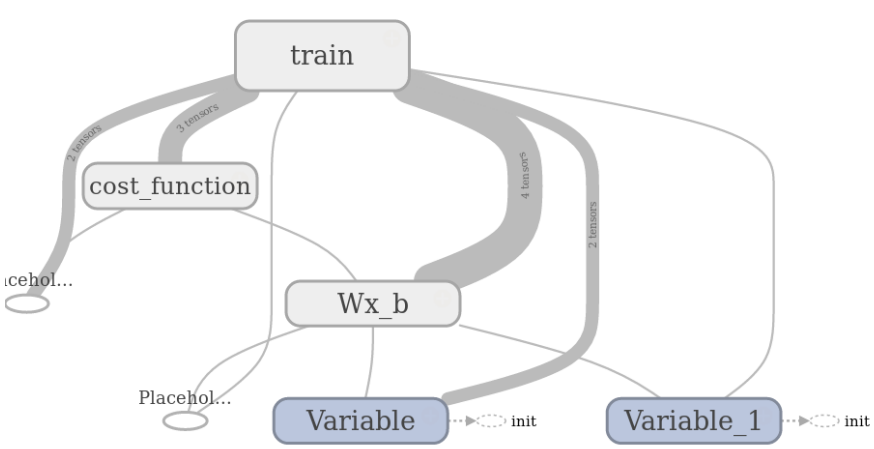
I did set them manually by training them on high end GPU and tuned them when i received highest accuracy i could.

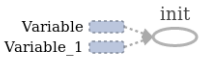
IV. Results

Model Evaluation and Validation

Cost Function

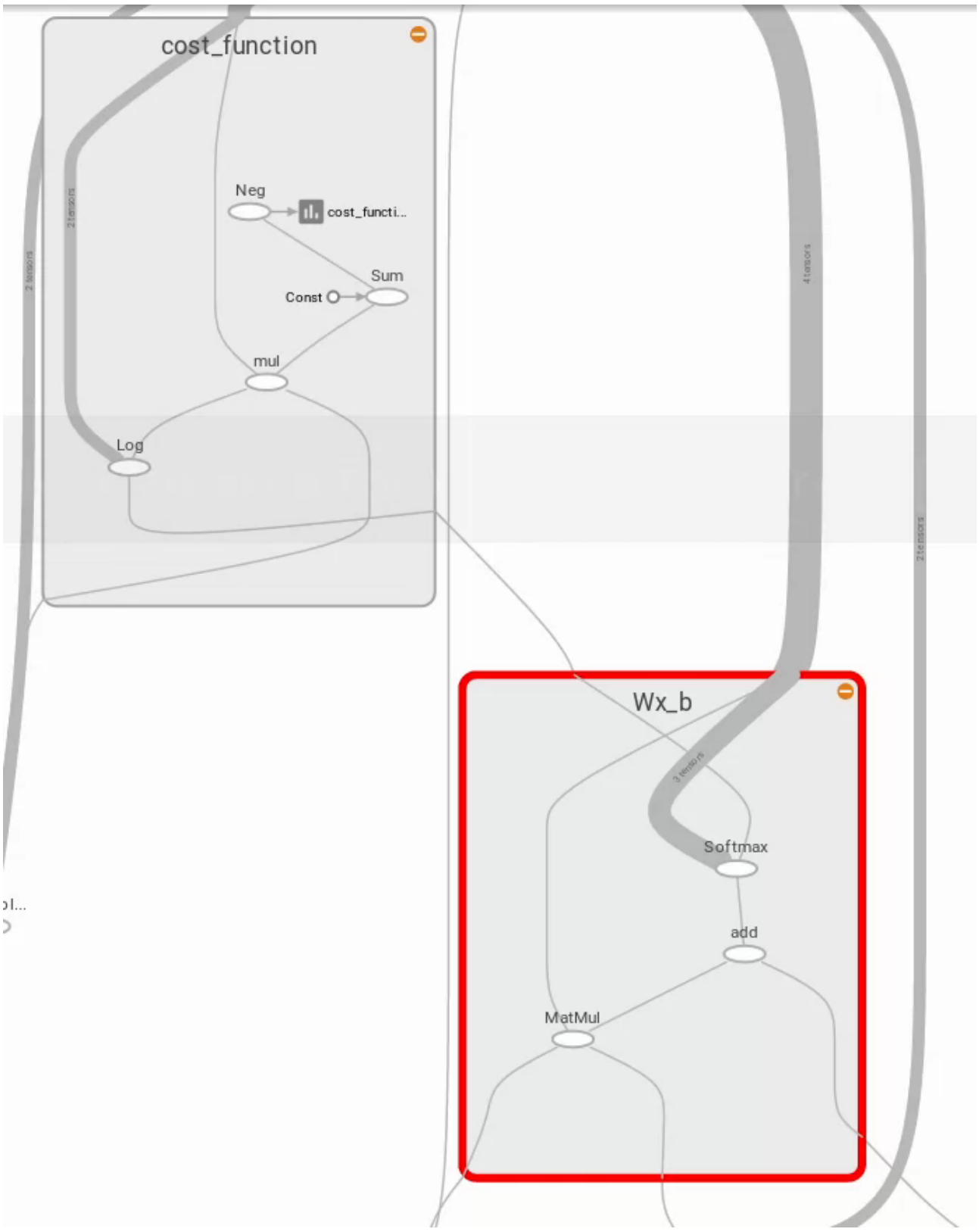
We Can see our graph of cost function over time how it changes as our model trains.





Here we can see how our model works we can flow of tensors over our edges connecting our nodes or we can say operations.

We can see each detail that how tensors are flowing using tensorgraph.



There is much more in graphs we can see all of them using tensorgraph.

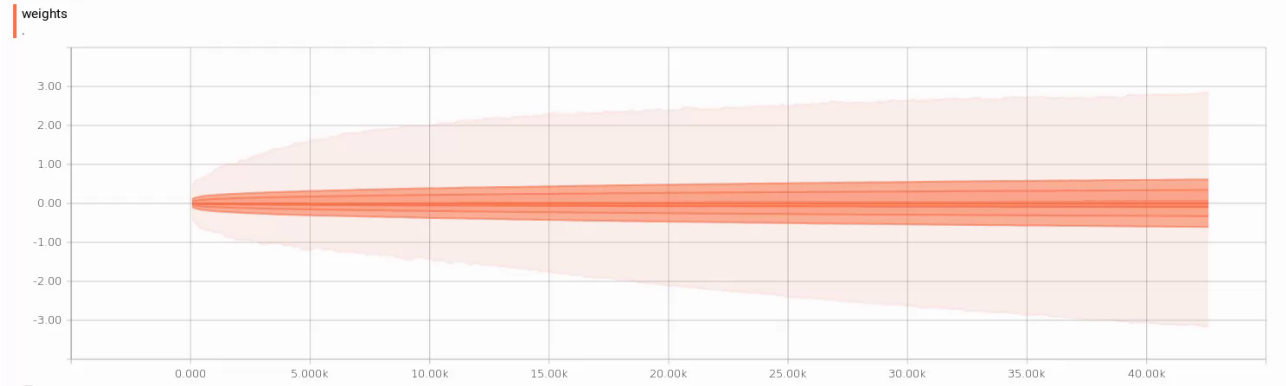
Justification

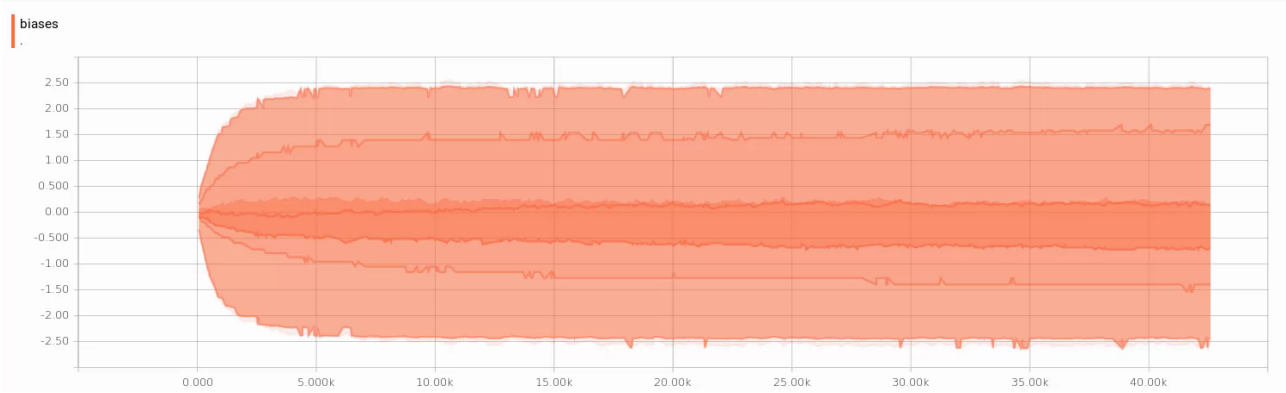
We can see that my benchmark model had very less accuracy less than 90% and that took much more time to train.

My current model has an accuracy of ~92% and it takes very less time to train i.e less then 5 minutes.

V. Conclusion

Free-Form Visualization

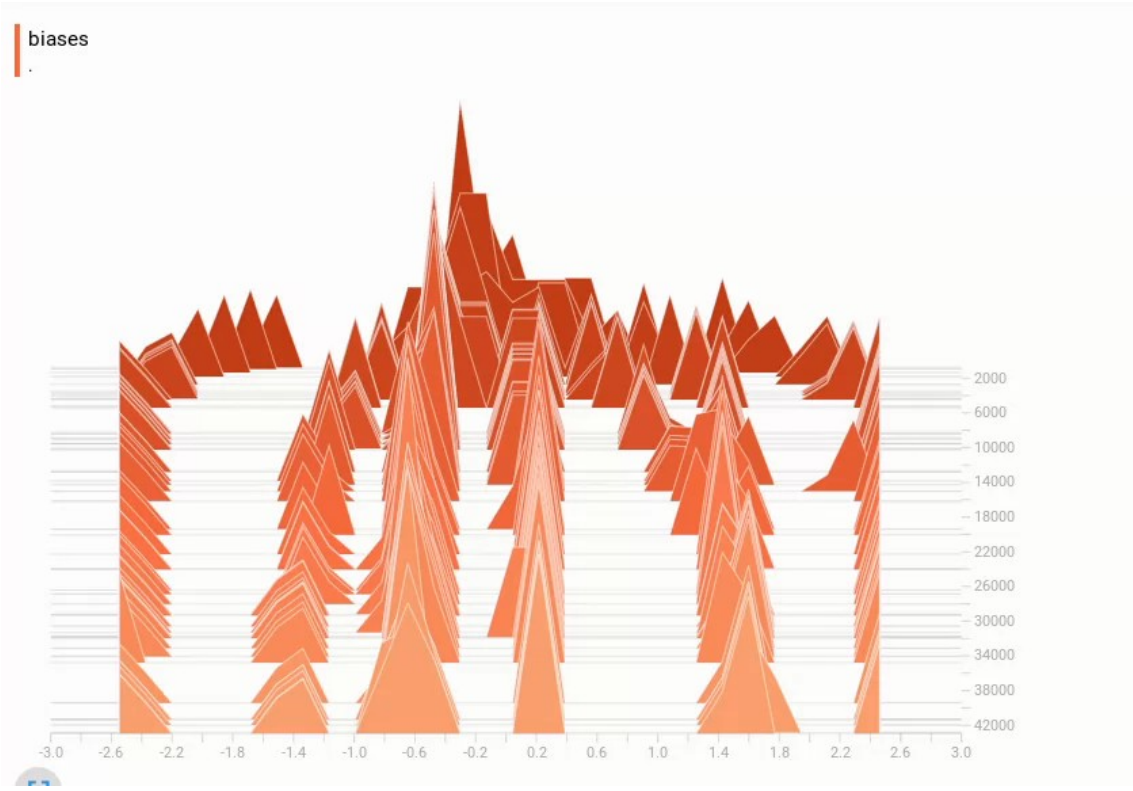




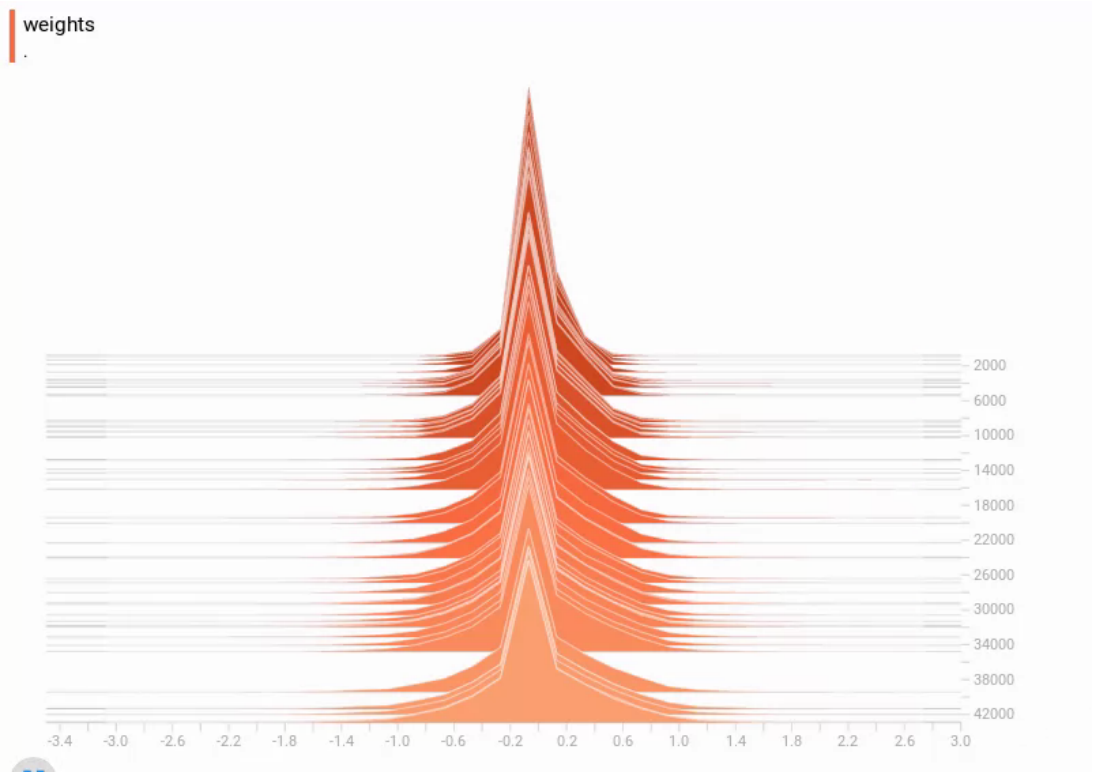
Here we can see the distribution of weights and biases over the data how they change with the change while training and optimise the model.

Histogram

Biases



Weights



Here We can see the histogram of biases and weights while we train the model

Reflection

While preprocessing the data choosing the training and testing set and making different files was quite challenging. Using the tensor graph was quite a fun that we can easily visualise the biases and weighs and also see how our model works in deep. Tuning the hyperparameter and using the softmax function was quite interesting that you can get different results with each and every hyperparamter and tuning them, making the batch size smaller and bigger, changing the learning rate and number of iterations. Choosing the best hyper parameters to gain best accuracy within sufficient time.

Improvement

We can make further improvements to this project we could use SVM with RBF kernels and different parameter values of C and gamma and optimising it using Grid Search and gain accuracy around 92% but we need much more computing power to do it in less time and it takes about a day on a quad-core system.

We could also use CNN with Keras with preprocessing data and gain ~99% accuracy.
