MIDDLE EAST TECHNICAL UNIVERSITY

MMI 727 HOMEWORK I

MURAT SEZGİN BALOĞLU

2469518

Derivatives of Activation Functions

→ Sigmoid : $\frac{1}{1+e^{-x}}$

$\xrightarrow{d/dx}$ $\frac{(1+e^{-x}).0-(+e^{-x})}{(1+e^{-x})^2} = \left[\frac{e^{-x}}{(1+e^{-x})^2}\right]$

→ Tanh : $\frac{e^{2x}-1}{e^{2x}+1}$

$\xrightarrow{d/dx}$ $\frac{(2e^{2x})(e^{2x}+1)-(2e^{2x}(e^{2x}-1))}{(e^{2x}+1)^2}$

$= \left[\frac{4e^{2x}}{e^{4x}+2e^{2x}+1}\right]$

→ ReLU : $\max(0,x)$

$\xrightarrow{d/dx}$ $\begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$

→ Softmax :

$$(X_{out})_i = \frac{e^{-\beta(X_{in})_i}}{\sum_k e^{-\beta(X_{in})_k}}$$

$$\frac{d(X_{out})_i}{d(X_{in})_j} = \text{for } i,j \text{ input pairs there are two general case: } i=j \text{ and } i \neq j$$

$i=j$ case :

$$\frac{e^{-\beta(X_{in})_i}\sum_k e^{-\beta(X_{in})_k} - e^{-\beta(X_{in})_j} e^{-\beta(X_{in})_i}}{\left(\sum_k e^{-\beta(X_{in})_k}\right)^2} =$$

$$= \frac{e^{-\beta(X_{in})_i}}{\sum_k e^{-\beta(X_{in})_k}} \cdot \frac{\sum_k e^{-\beta(X_{in})_k} - e^{-\beta(X_{in})_j}}{\sum_k e^{-\beta(X_{in})_k}}$$

$$= \frac{e^{-\beta(X_{in})_i}}{\sum_k e^{-\beta(X_{in})_k}} \left(1 - \frac{e^{-\beta(X_{in})_j}}{\sum_k e^{-\beta(X_{in})_k}}\right) \quad \text{for } i=j$$

$i \neq j$ case 2

$$\frac{\partial}{\partial} = \frac{e^{-\beta(x_{in})_j} \; e^{-\beta(x_{in})_i}}{\left(\sum_k e^{-\beta(x_{in})_k}\right)^2}$$

$$= \frac{e^{-\beta(x_{in})_j} \; e^{-\beta(x_{in})_i}}{\left(\sum_k e^{-\beta(x_{in})_k}\right)^2}$$

$\Rightarrow$ The term $\beta(x_{in})$ is not understood clearly. Is β a constant to $(x_{in})_i$ or is $\beta(x_{in})$ is a output vector of function β and $\beta(x_{in})_i$ is i^{th} element of this vector. Hence β ignored.

Q 1) Part B :

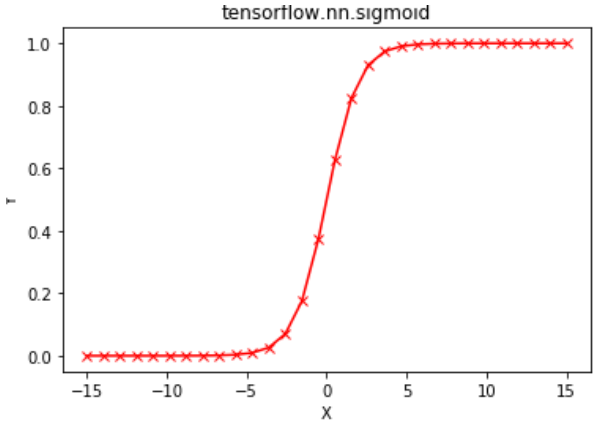

Figure 1 Sigmoid Activation Function

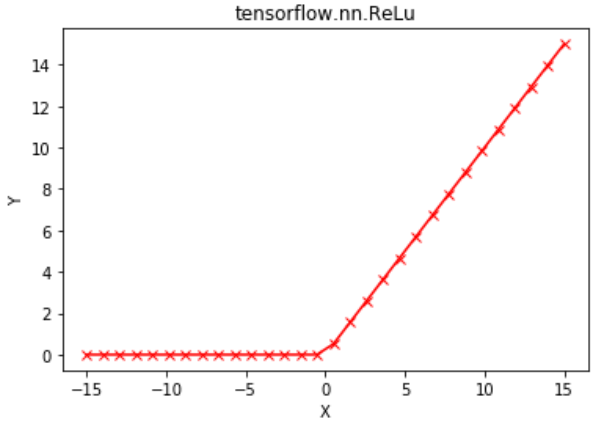

Figure 2 ReLu Activation Function

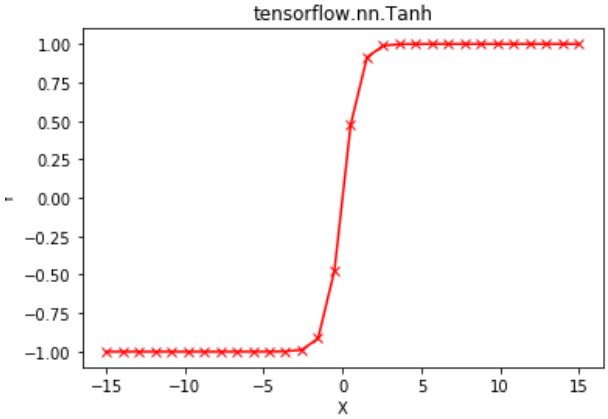

Figure 3Tanh Activation Function

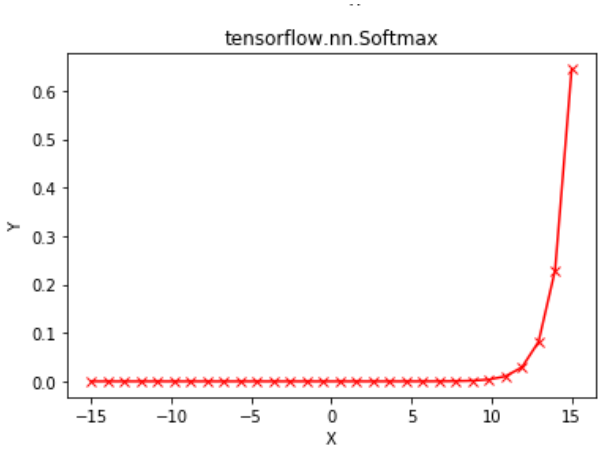

Figure 4 Softmax Activation Function

Q 1) Part C :

Sigmoid :

Advantages :

+ Easy to understand.

+ Output is bounded.

+Differentiable everywhere

Disadvantages :

-Vanishing gradient problem

-Slow convergence

-Output is not zero centered

Where to use?

->Popular with classification problems

->Usually used in the output layer of a binary classification problem where result is either 0 or 1.

Tanh:

Advantages :

+Output is zero centered

+Output is bounded

+Gradients are steeper with respect to sigmoid

Disadvantages :

-Vanishing gradient problem

Where to use?

Usually used in hidden layers of a NN as its values lies between -1 to +1 hence mean of hidden layyer approximates to 0. This makes learning for the next layer much easier.

ReLu:

Advantages:

+Avoids vanishing gradients problem.

+Computationally less expensive compared to sigmoid and tanh.

Disadvantages :

-Gradient may die durig training.

-Output is unbounded.

-It could result in dead neurons, substantial part of network may go passive.

Where to use?

It should be used in hidden layers.

Widely used in many cases especially in CNNs.

Softmax:

Advantages:

+Able to handle multiple classes.

Where to use?

Usefull for output neurons.

Widely used in multiple classificiation logistic regression model.

2. PARAMETER TUNING

A. LEARNING RATE EFFECT

Bigger learning rate increases the speed of convergence but may result in unsteady system. As it can be seen in figures 7-8. In figrue 7, with learning rate 50, accuracy oscillated below 0.6 moreover, as learning rate increase further to 500, accuracy oscillated below 0.5 meaning larger oscillations.

On the other hand, small learning rate such as 0.005 resulted in slow convergence and it is clear in figure 6 that NN is not converged yet as of 1000th iterations.

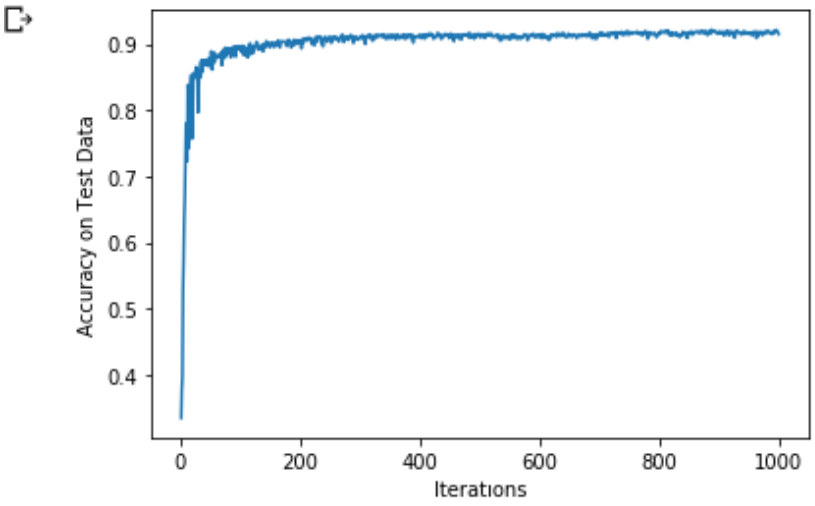

Figure 5 Learning rate 0.5

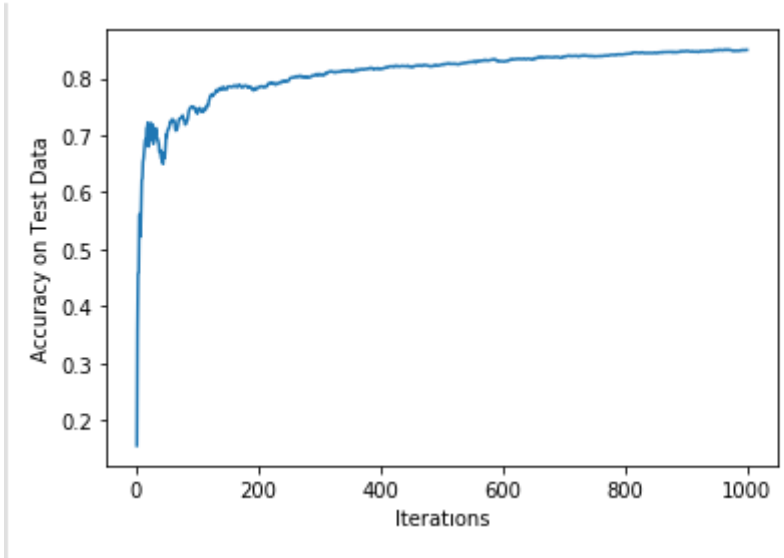

Figure 6 Learning rate 0.005

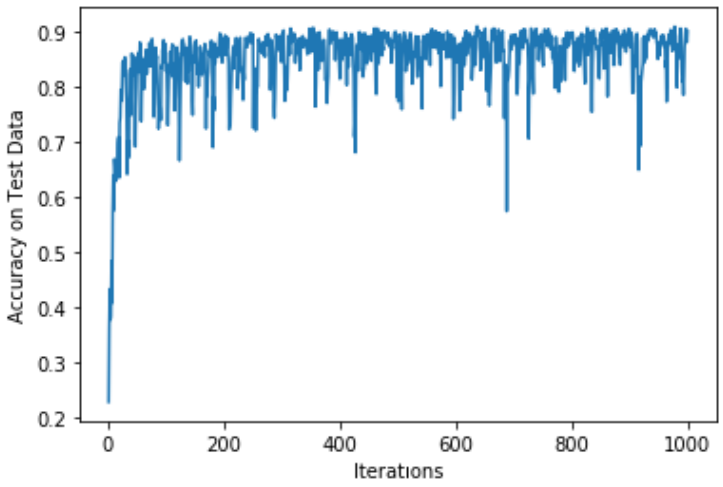

Figure 7 Learning Rate 50

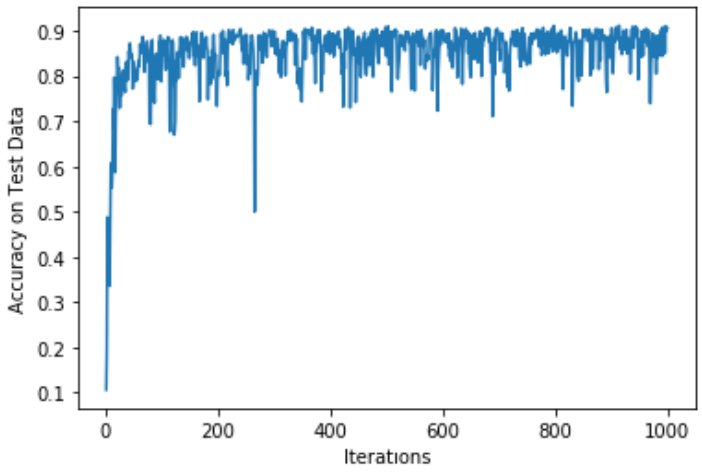

Figure 8 Learning Rate 500

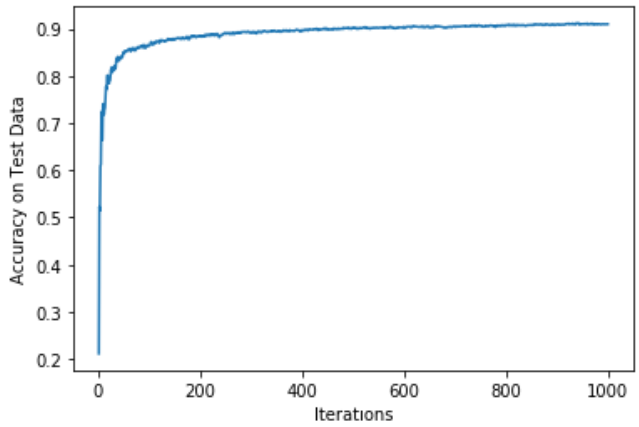

Figure 9 Learning rate 0.1

B) DIFFERENT OPTIMIZERS

Optimizer change did not result in change in computation time, convergence rate is also not drastically changed. However, Gradient Descent is more stable than compared to Adagrad and Adam. Adam optimizer's performance is weaker compared to Gradient Descent and Adagrad Optimizers.

As Momentum optimizer's momentum increase, NN behaves more unstable, this can be compansated by decreasing the learning rate but this time convergence rate would also decreased.

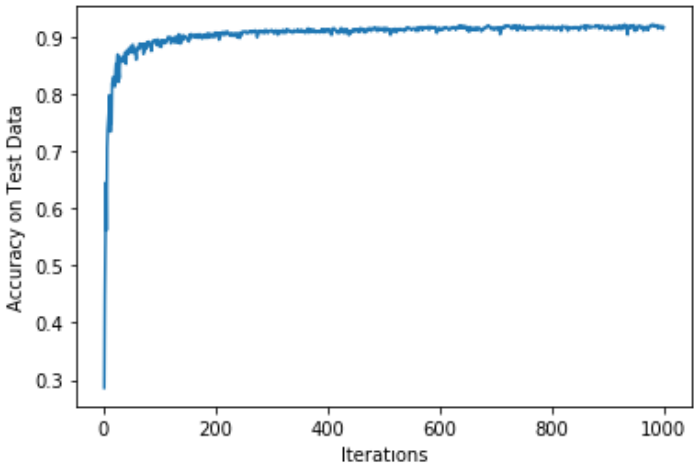

Figure 10 Default Gradient Descent Optimizer Duration 16.036s

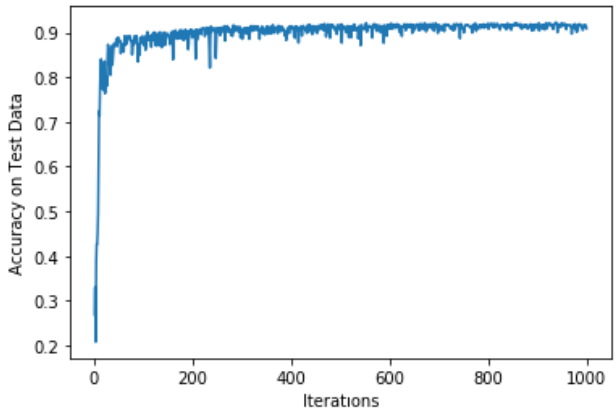

Figure 11 Adagrad Opt. Duration 16.236 s

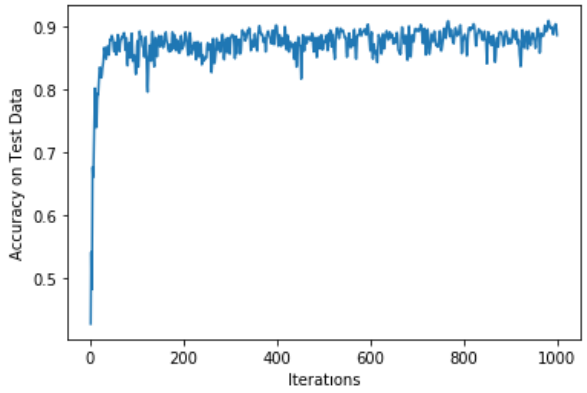

Figure 12 Adam opt. Duration 16.468s

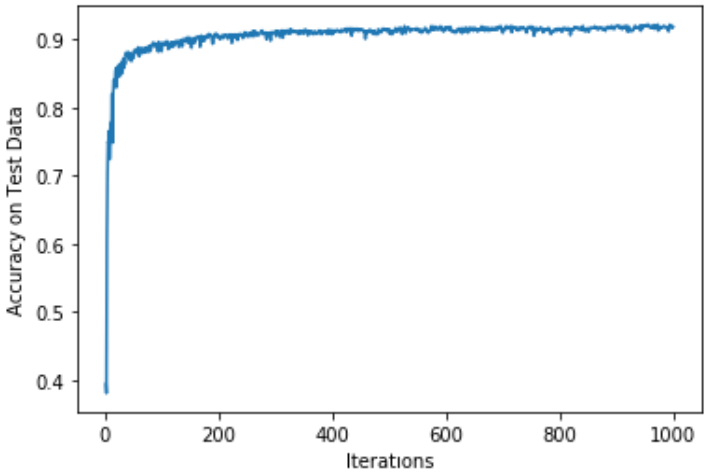

Figure 13 Momentum opt. lr 0.5 momentum 0.1 Duration 16.345 s

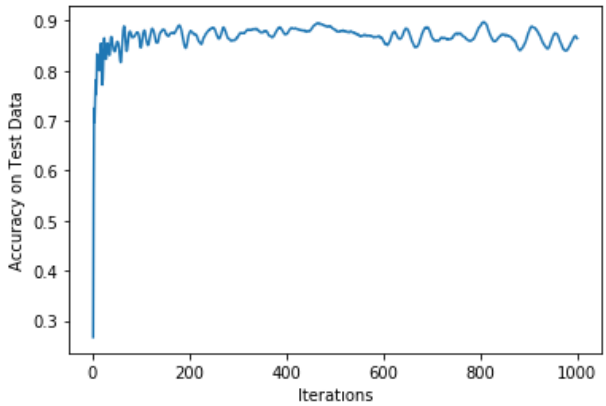

Figure 14 Momentum opt. lr 0.5 momentum 1Duration 16.257 s

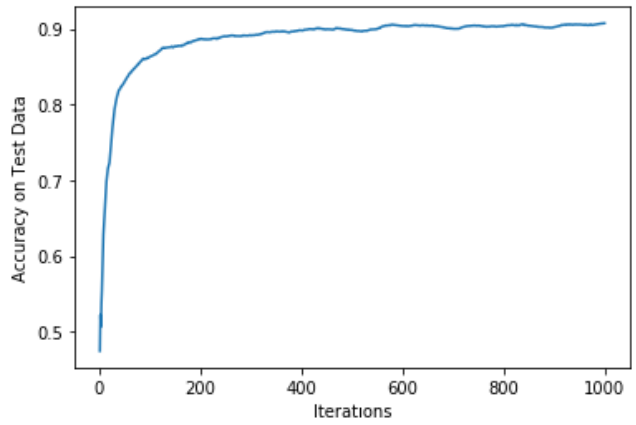

Figure 15 momentum opt. lr 0.005 momentum 1 Duration16.778s

C) BATCH SIZE

- The plots show that small batch results generally in rapid learning but a unstable learning process with higher variance in the classification accuracy. Larger batch sizes slow down the learning process but the final stages result in a convergence to a more stable model exemplified by lower variance in classification accuracy.

Moreover, as batch size increased, accuracy is slightly increased.

Smaller batch sizes resulted in unstable networks, learning rates are decreased in order to compansate this but this led to larger convergence rates as can be seen in figures 17 and 18.

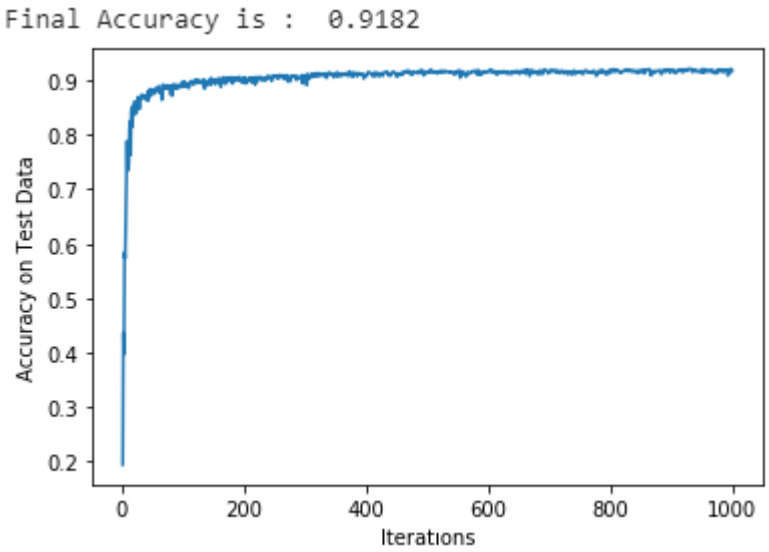

Figure 16 default one batch size = 100 Duration 16s

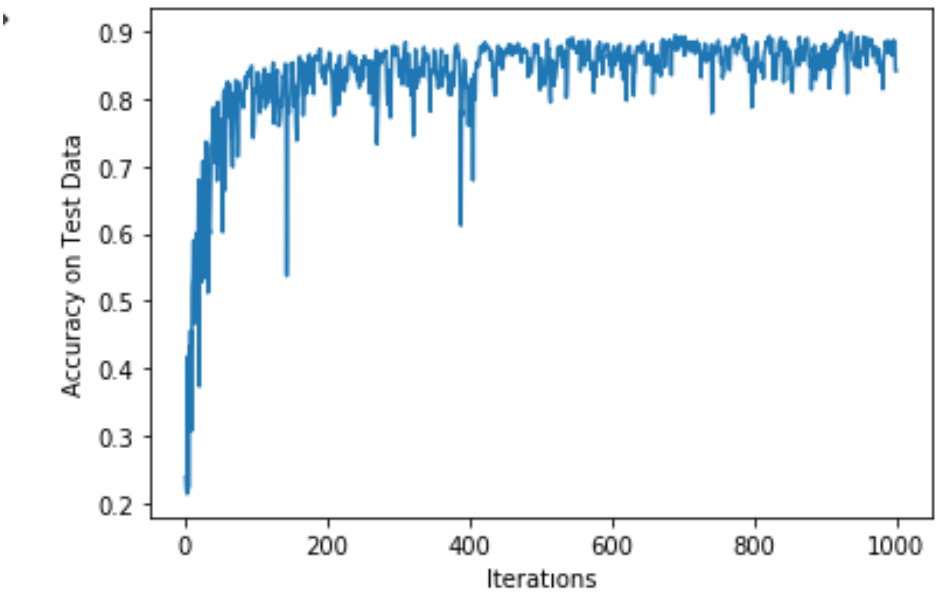

Figure 17 Batch size = 10 lr = 0.5 Duration 16s

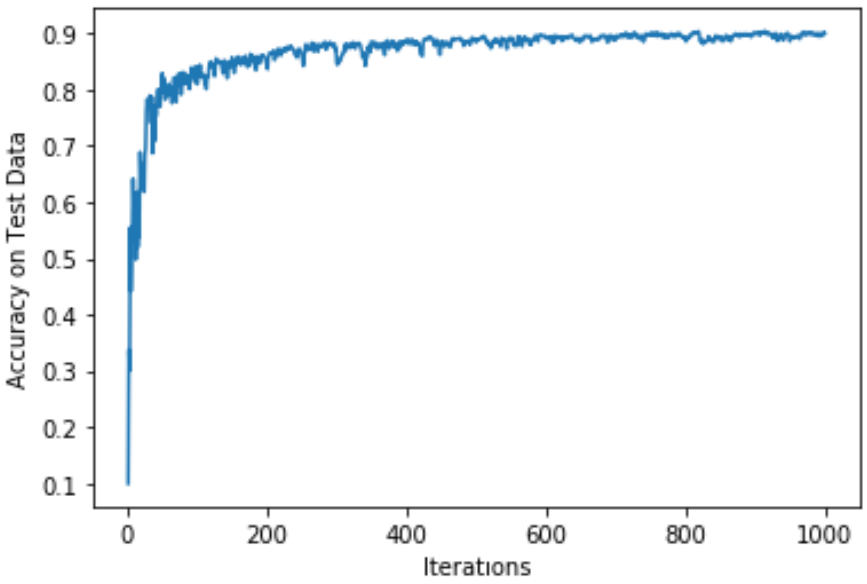

Figure 18 Batch size 10 learning rate decreased to 0.1

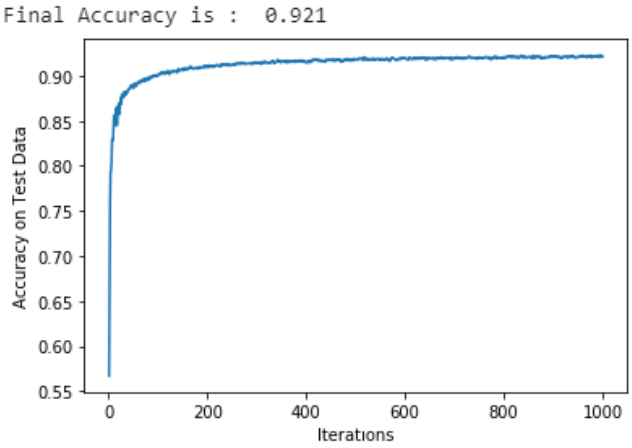

Figure 19 Batch size 1000 learning rate = 0.5 Duration 20,48s

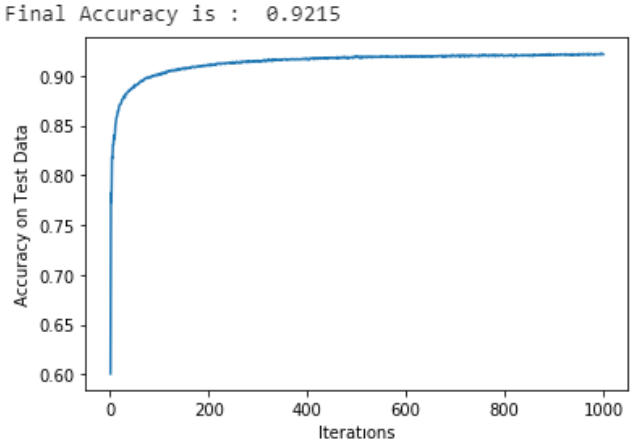

Figure 20 Batch size 10000 learning rate = 0.5 Duration 63s

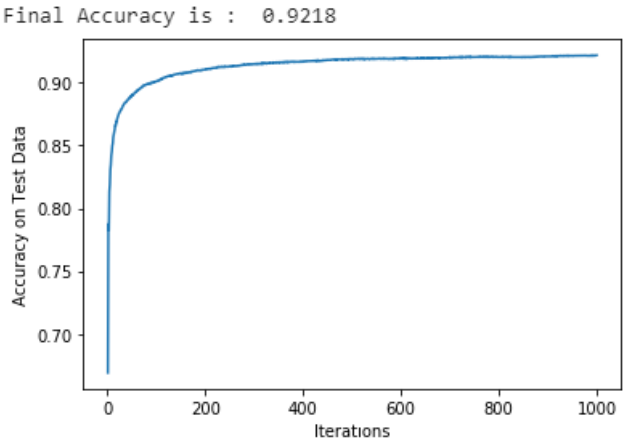

Figure 21 Batch size 50000 Learning Rate 0.5 Duration 270s

3) NETWORK MODEL

A) ORIGINAL SINGLE LAYER NETWORK

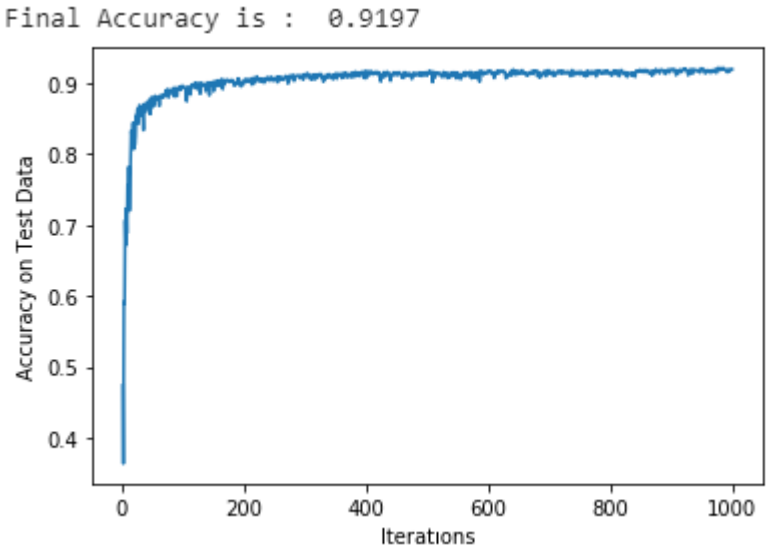

Figure 22 Original Single Layer Network

B) TWO LAYER NETWORK

Two layer network didn't work with its previous hyper parameters because weights in the first layer change very little and error propagates. Implemented solution is to initialize weights with random uniform distribution.

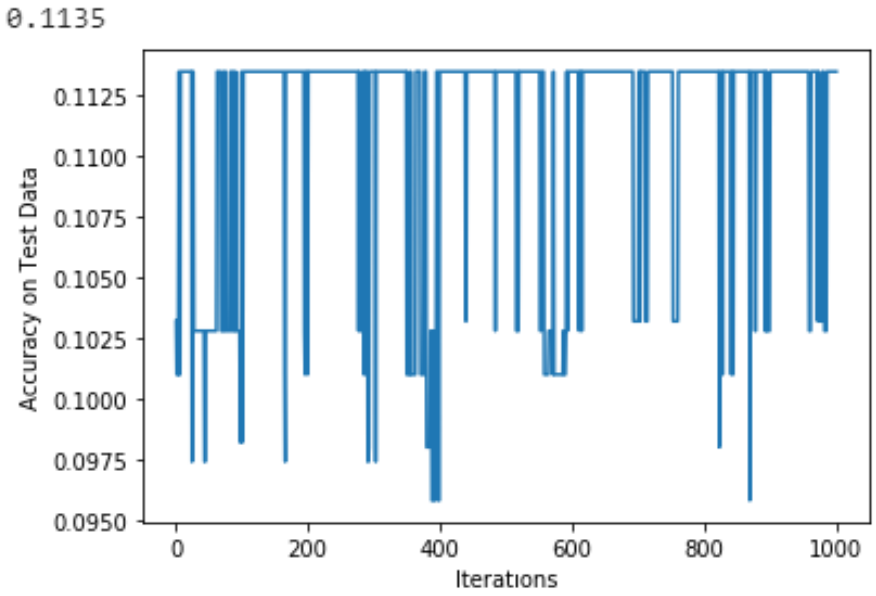

Figure 23 First version of 2-layer network, network did not learn, Duration 140s

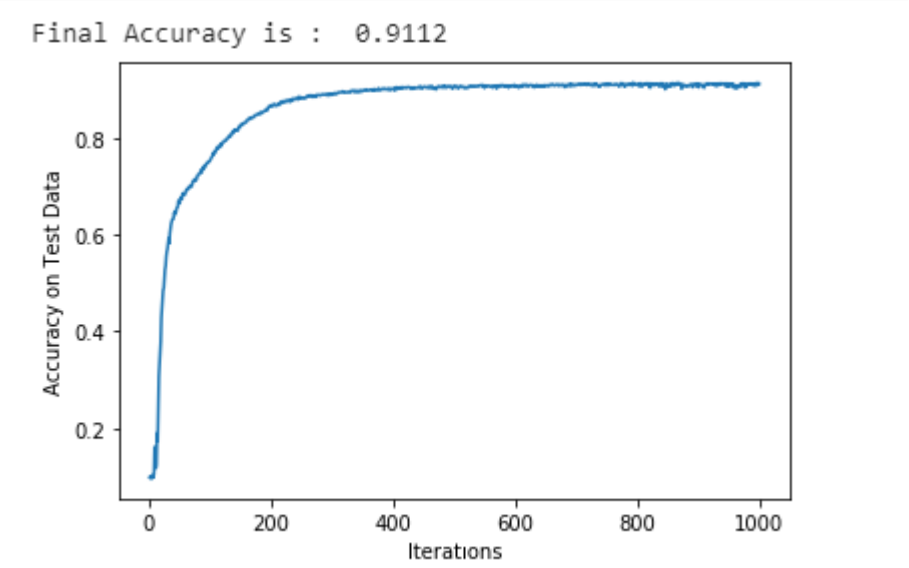

Figure 24 Weights are randomly initialized, batch size increased to 1000 Duration 244s, Learning rate 0.1, Gradient Descent Opt.

C) THREE LAYER NETWORK

3 layers NN did not work at first, then optimizer changed to Adagrad Optimizer then, other hyperparameters tuned.

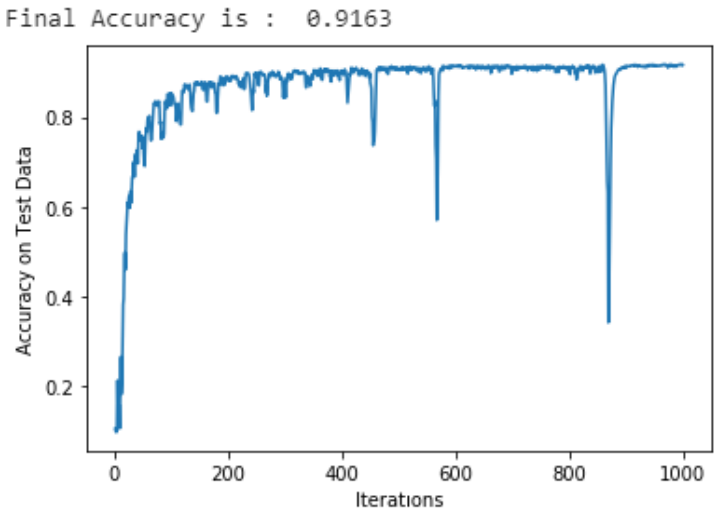

Figure 25 Adagrad opt. Lr 0.5 batch size 1000

D) THREE LAYER IMPLEMENTATION USING TF.LAYER

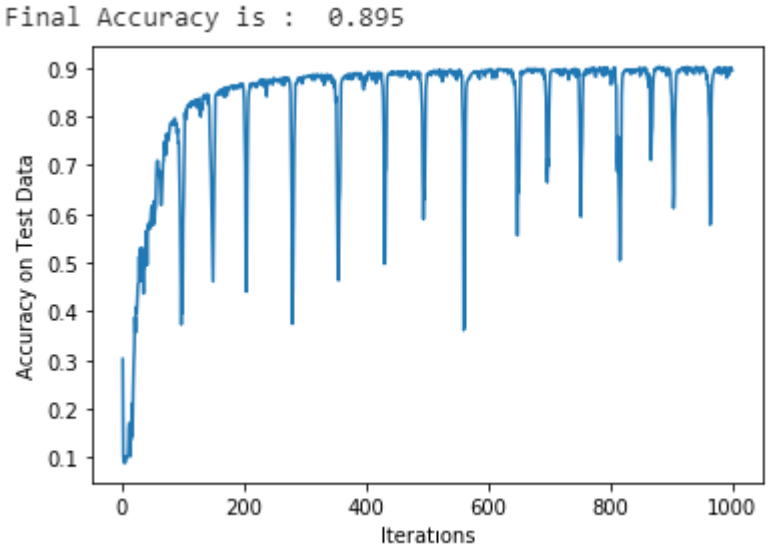

Figure 2 Same network with same hyper parameters in part c implemented with tf.layers module

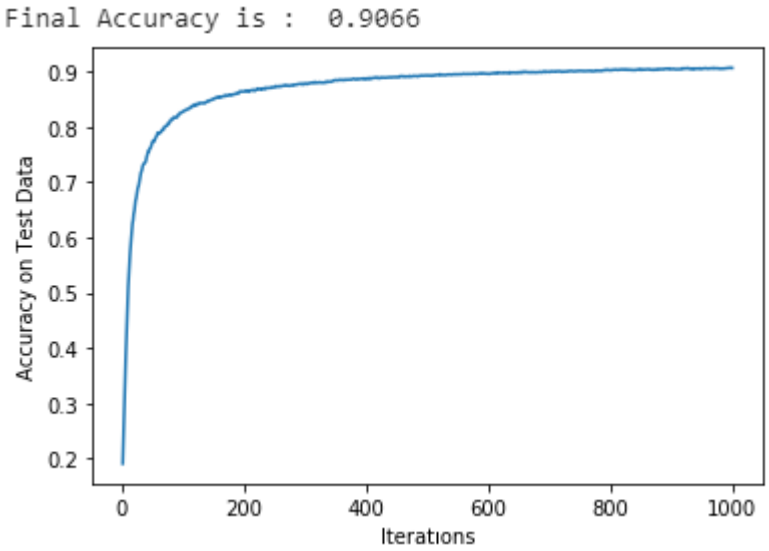

Figure 3 3 layer network implemented with tf.layers module with parameters : GradientDescentOptimizer, lr = 0.01, batch_size = 500