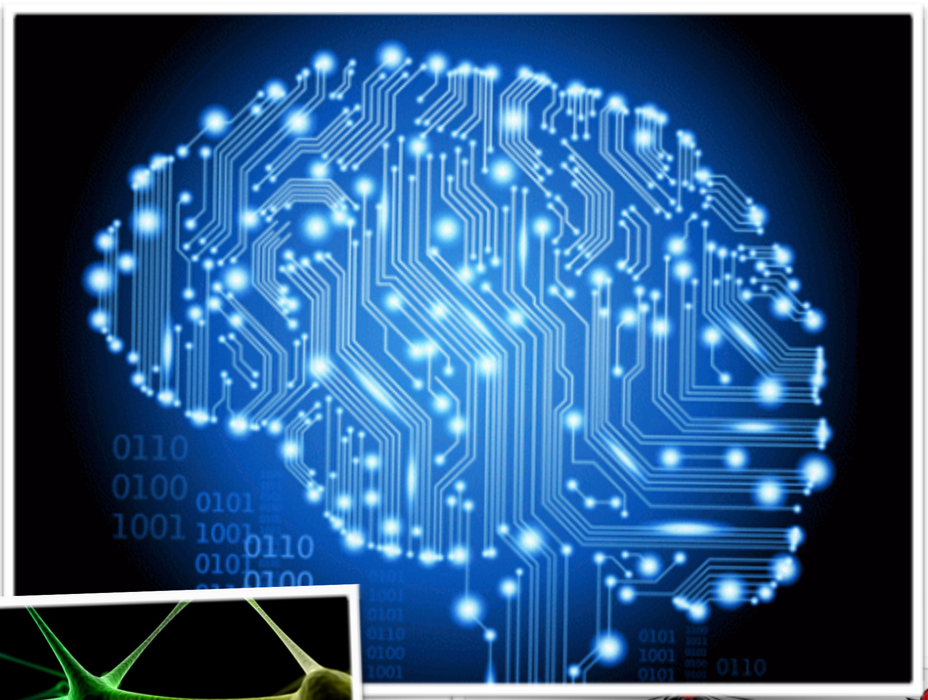
Saurabh Hinduja

Neural Network

Homework # 3



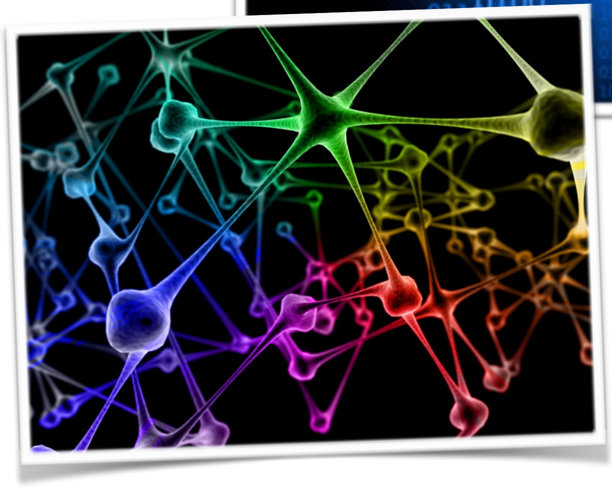
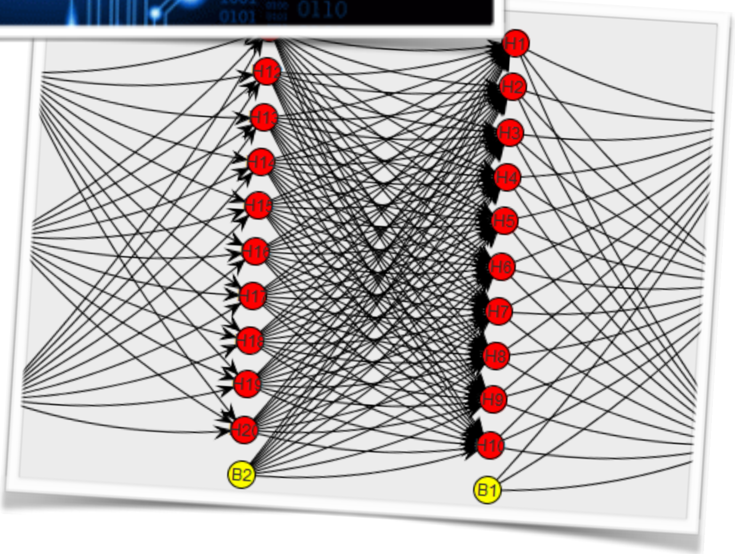



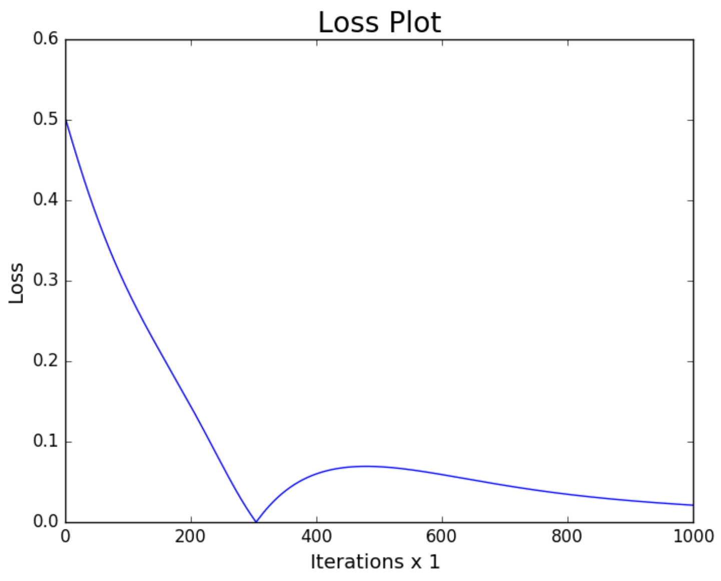
Table of Contents

Introduction

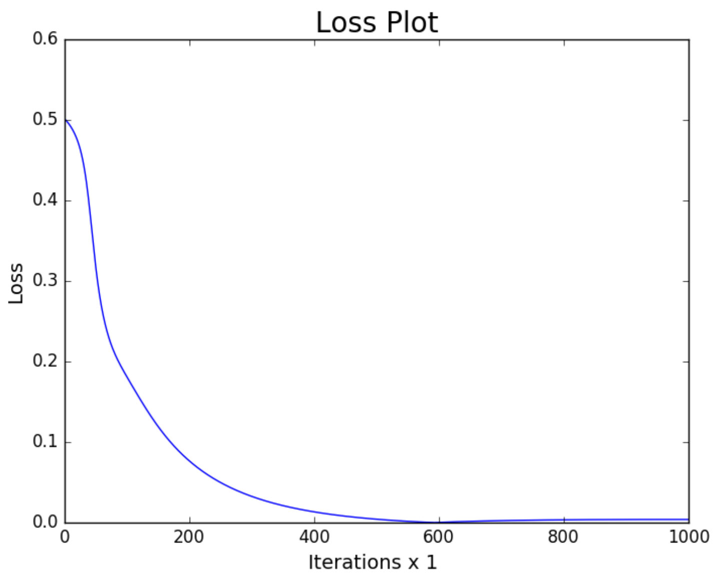
Neural networks is a way of training the computer to recognize patterns and separate samples based on classes after training. Their are various training models which are available, this is an attempt to make a neural network and change various parameters of the neural network and check the change in performance of the neural network. For checking the performance of different neural networks hw2_dataset2.txt was used, in every execution the dataset was shuffled and 90% of it was used for training while remaining 10% was used for testing. Tanh activation function with a neural network of 5 layers with 5 neurons in each layer was used for most cases. Loss plot and percentage error on testing dataset were used to check performance.

Layers and Neurons:

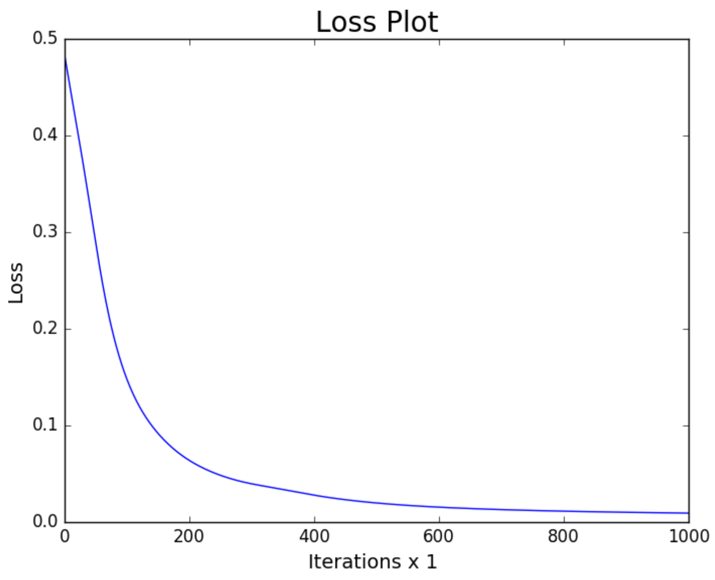
For testing the change in performance based on change in number of layers and neurons, the activation function was tanh, number of iterations was 1000, lamda (learning rate) of 0.01 were used and the combination of layers and neurons in each layer was changed.



Plot 1: 2 Layers 2 Neurons. Loss: 0.12



Plot 2 5 Layers 2 Neurons. Loss 0.118



Plot 3 5 layers 5 Neurons. Loss 0.082

From plot 1, it can be seen that when only 2 layers were used the training was not stable as the loss reduced and then increased before stabilizing at a higher value than the minimum, giving a loss of 0.12 on the testing dataset. For plot 2, the number of layers were increased to 5 keeping the number of neurons as 2 per layer, it was observed that the loss plot was more stable with it dropping sharply, but there was not much difference in the loss in testing dataset, it reduced to 0.118. For plot 3, the number of neurons per layer was increased to 5, it can be observed that the loss graphs does not drop as sharply as in plot 2, but the loss on testing dataset reduces to 0.082.

Conclusion:

Increasing number of layers makes training more stable and increasing number of neurons per layer helps improve the loss, that is, it classifies test dataset more efficiently.

Activation Functions:

Three activation functions were used to check the change in performance. The functions used were: 1. Sigmoid, 2. Tanh, 3. ReLu. For testing each activation function, the number of iterations was fixed at 1000, lamda at 0.01, number of layer at 5 and number of neurons in each layer at 5.

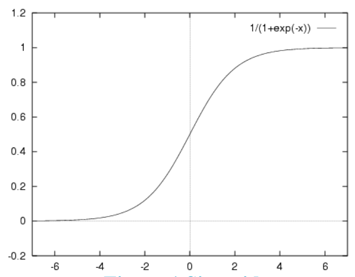


Figure 1 Sigmoid

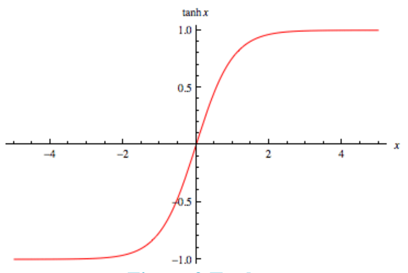


Figure 2 Tanh

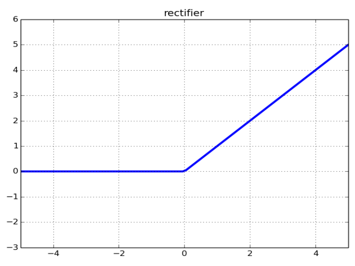
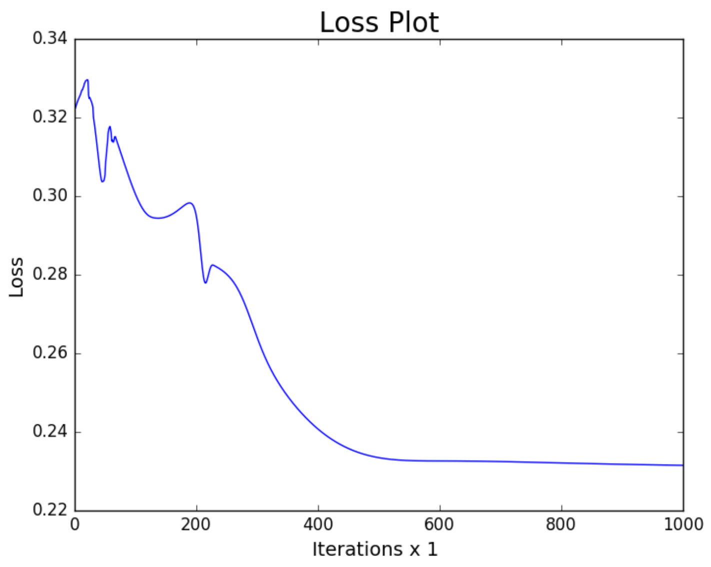
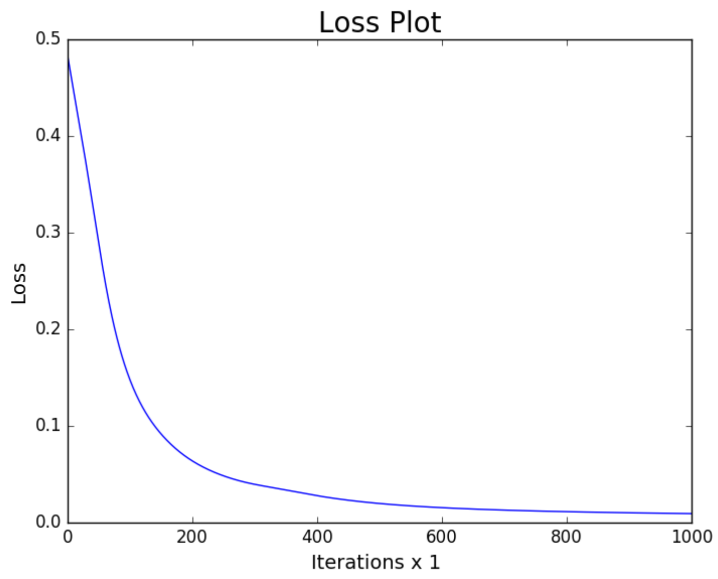


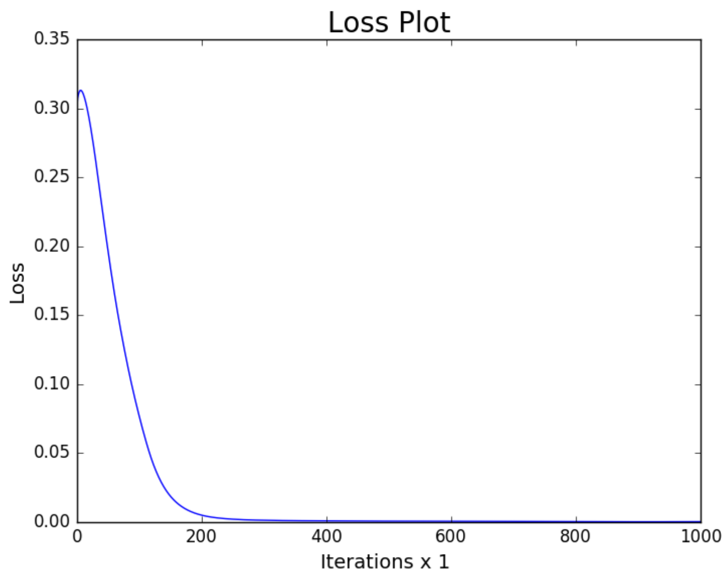
Figure 3 ReLu



Plot 4 Sigmoid activation function



Plot 5 Tanh activation function



Plot 6 ReLu activation function

Conclusion:

Sigmoid activation function starts giving NaN (not a number error) when the input to sigmoid becomes very small. With this error the training cannot be completed successfully and when we try to test it on test dataset it either gives the error as 0 or NaN. To get around this error, numpy has an inbuilt function called NaN_to_num, this function converts all NaN values to 0, all positive infinity to a very large positive number and very large negative number to very small numbers. The

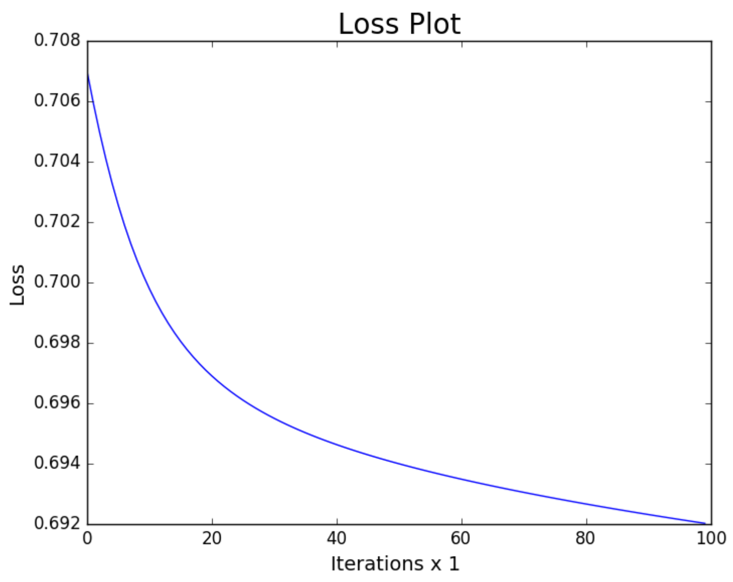
NaN error can be Conclusion Tanh gave the lowest loss of 0.103 compared to ReLu loss of 0.122 and sigmoid loss of 0.151.

Softmax function:

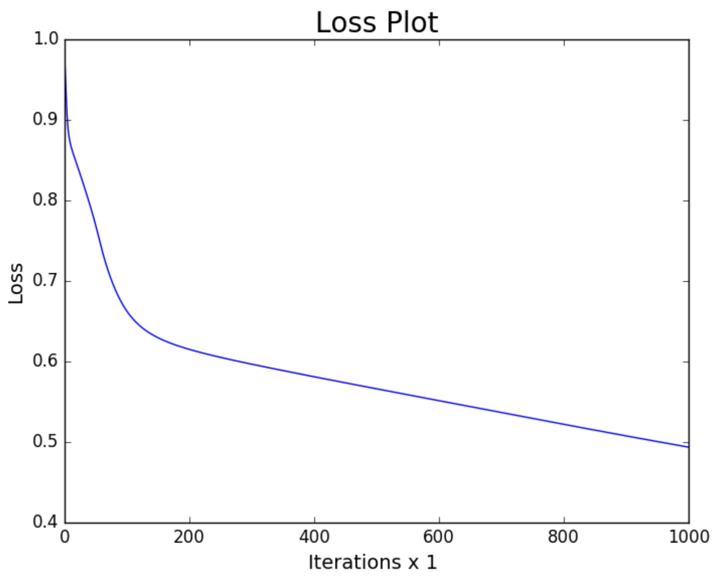
Softwmax was tested with Cross entropy as the loss functions, Tanh as activation function with 100, 1000 and 10000 iterations.

$$p(i) = \frac{e^{\frac{f(i)}{T}}}{\sum_j e^{\frac{f(j)}{T}}}$$

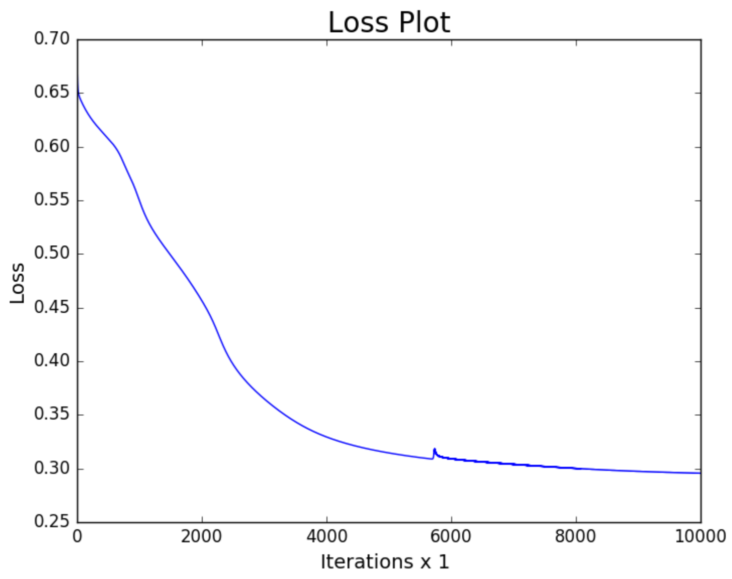
Equation 1 Softmax



Plot 7 100 iterations



Plot 8 1000 iterations



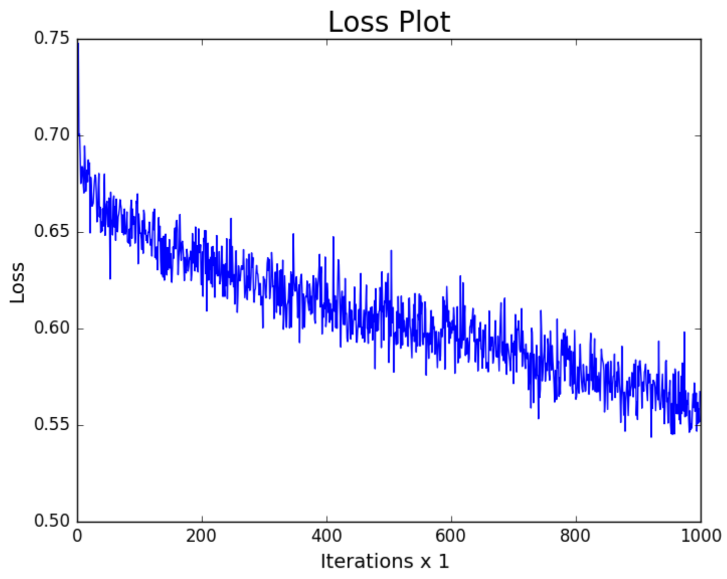
Plot 9 10000 iterations

Conclusion:

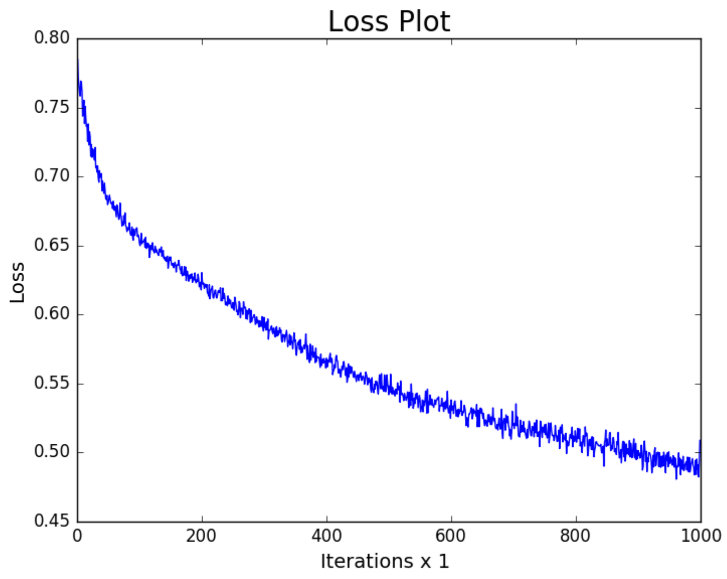
Softmax takes more iterations to train, but the error using softmax is less compared to the regression function. The error reduced from 44% in 100 iterations to 17% in 1000 iterations to 8.13% in 10000 iterations

Stochastic gradient descent:

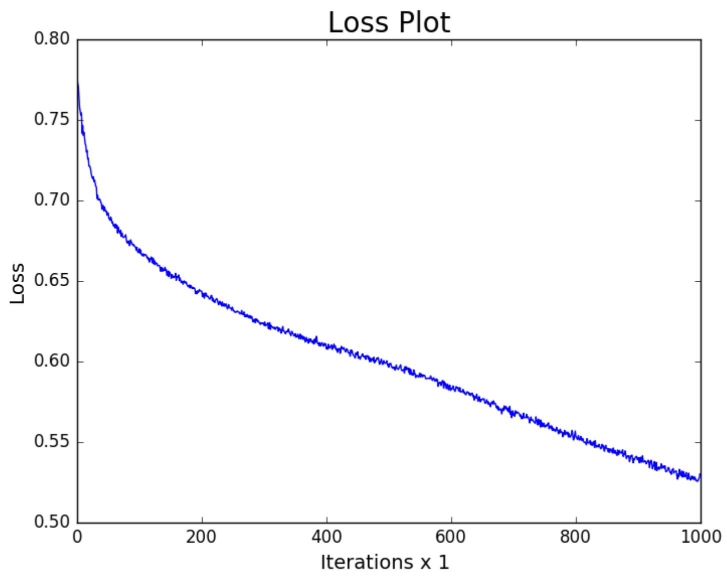
Stochastic gradient descent was tested with taking different percentage of the training dataset was used to check the various in stochastic gradient descent. 10%, 50% and 90% were used.



Plot 10 10% stochastic gradient descent



Plot 11 50% stochastic gradient descent



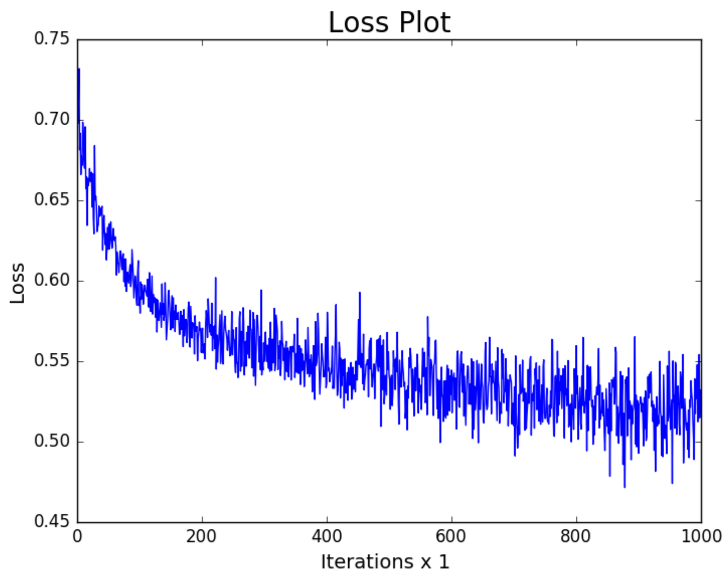
Plot 12 90% stochastic gradient descent

Conclusion:

When smaller chunks of training dataset is used for stochastic gradient descent the variations in each iterations is very high and loss plot converges slower, as the size of training chunks increase the variation decrease and loss plot converges faster, but as we increase the number of iterations the smaller chunks give lower error on testing dataset.

Preprocessing:

For preprocessing the whole dataset was normalized and centralized before splitting it into training and testing datasets. 10% of the dataset was used for stochastic gradient decent.



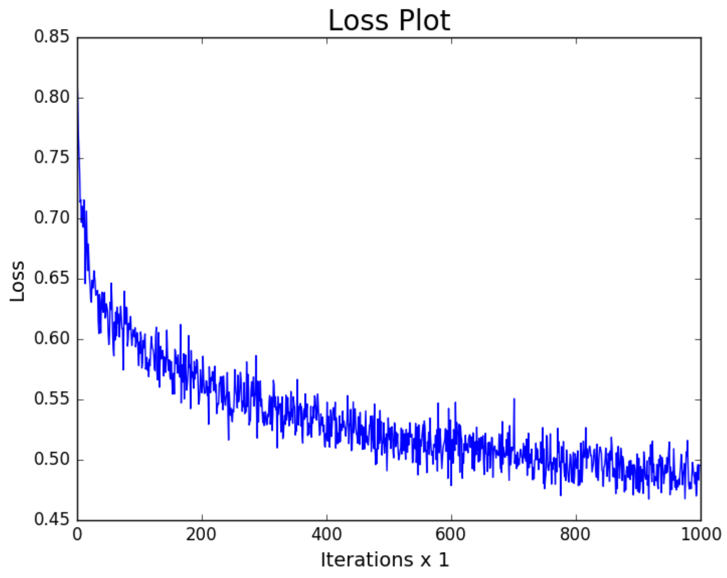
Plot 13 Preprocessing.

Conclusion:

Preprocessing makes more variation in the loss plot but helps to converge faster and the error on test dataset is lower as compared to before preprocessing. The error reduced from 28% to 18% with preprocessing.

Regularization

For regularization of weights, beta (β) of 0.001 was used.



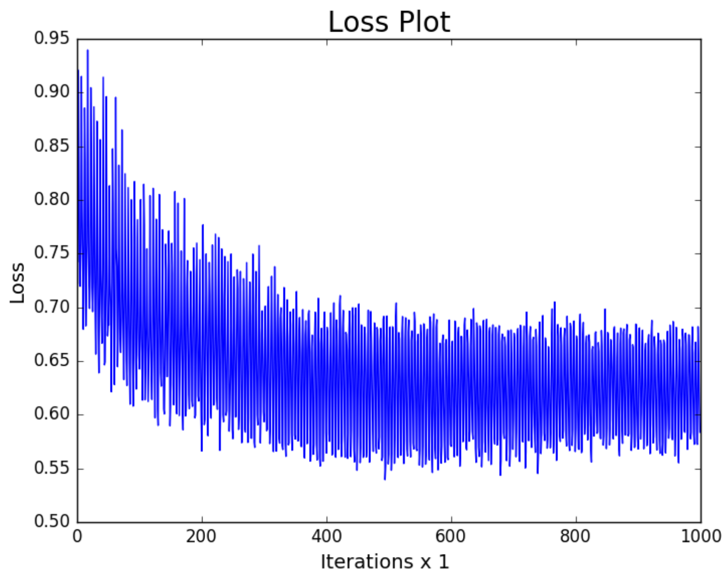
Plot 14 Regularization

Conclusion:

Regularization helps to converge faster and give better results on test dataset. The error reduced to 15% with regularization from 18% with only preprocessing.

Dropout

For testing dropout, a mask was was used to drop neurons in the first layer. For simplicity only one neuron was dropped per iteration and the neuron dropped was changed for every iteration.



Plot 15 Dropout

Conclusion:

With dropout the frequency of variation in the loss function increases sharply and it converges much slower with the error increasing to 25%. This change in error and increase in frequency can be due to the fact that dropout avoids overfitting.