Assignment 3B (COL 774)

By Ritvik Vij (2017CS50417)

1. Building a Neural Network

I have built a neural network with Stochastic Gradient Descent and MiniBatch implementation. The mini batch size, neural net architecture,type of activation function can be inputed as custom inputs by the user. I have uniformly taken batch size as 100.

1(b)

Hidden Layer Size 100:

Time for convergence - 563 sec
Train Accuracy - 97.7%
Test Accuracy - 89.2%

Stopping Condition:

Difference of Loss = 10^-4 or number of epochs<10000

Hidden Layer Size 50:

Time for convergence - 563 sec
Train Accuracy - 96.2%
Test Accuracy - 84.6%

Stopping Condition:

Difference of Loss = 10^-4 or number of epochs<10000

Hidden Layer Size 10:

Time for convergence - 474 sec
Train Accuracy - 86.7%
Test Accuracy - 73.1%

Stopping Condition:

Difference of Loss = 10^-4 or number of epochs<10000

Hidden Layer Size 5:

Time for convergence - 414 sec
Train Accuracy - 54.99%
Test Accuracy - 49.97%

Stopping Condition:

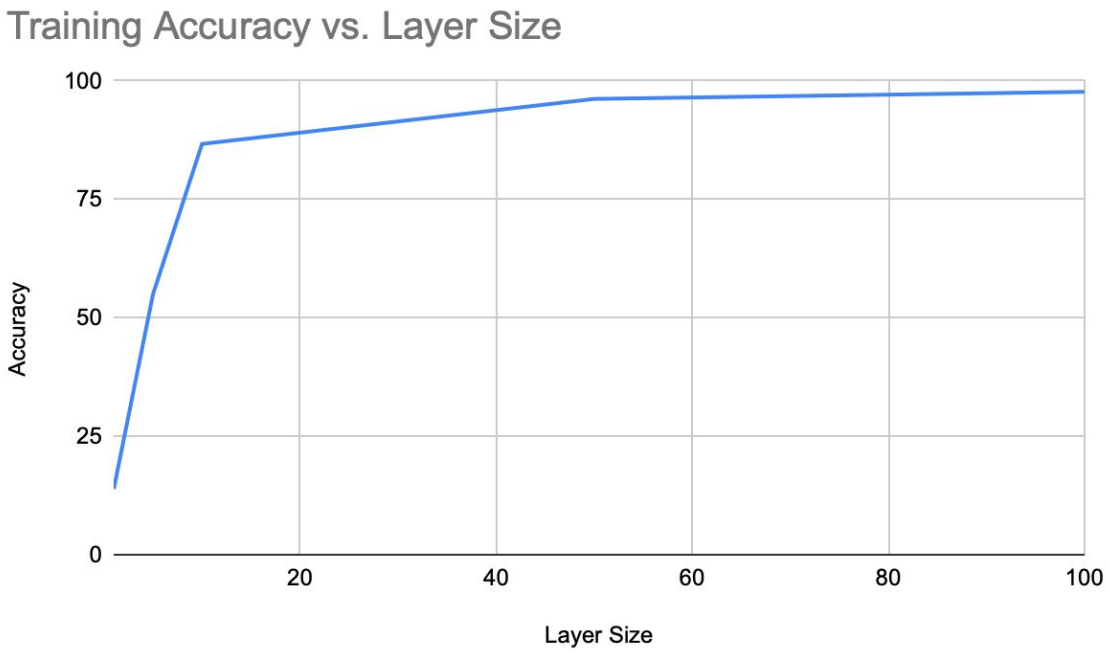
Difference of Loss = 10^-5 or number of epochs<10000

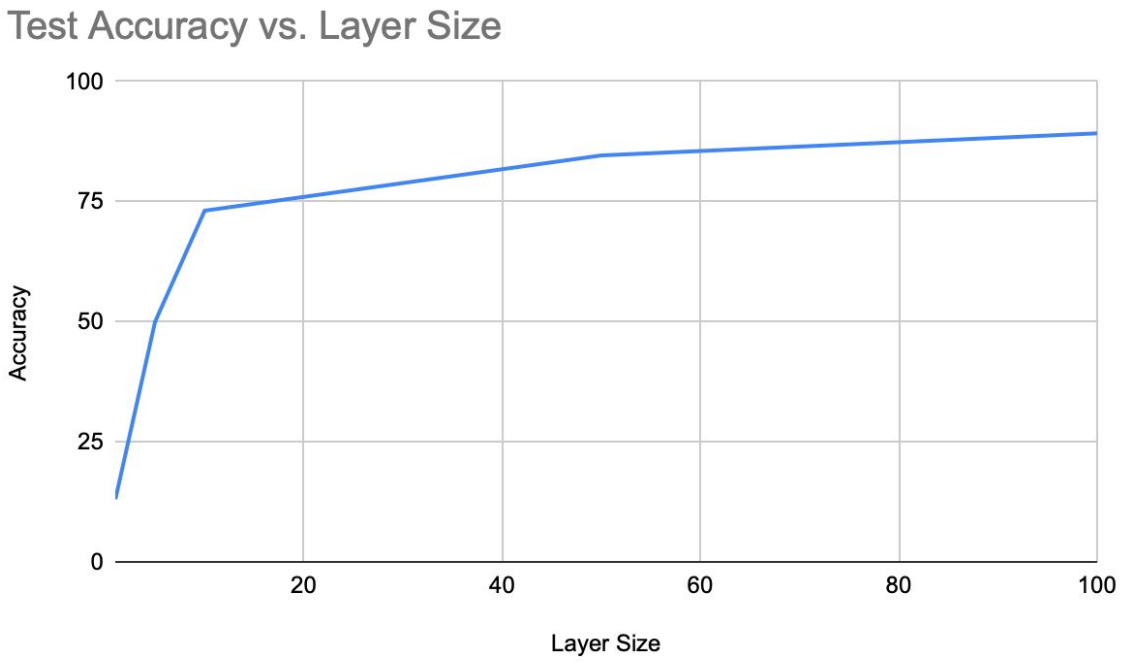
Hidden Layer Size 1:

Time for convergence - 302 sec
Train Accuracy - 13.87%
Test Accuracy - 13.03%

Stopping Condition:

Difference of Loss = 10^-6 or number of epochs<10000





Using Adaptive Learning Rate

I have taken N(not) in this case as 0.5 for the cases 100 and 50 size of the hidden layer and 1 in the case of 10,5,1 because the training would collapse very fast in the latter cases if N(not) was taken as 0.5.

Hidden Layer Size 100:

Time for convergence - 1023 sec
Train Accuracy - 88.9%
Test Accuracy - 82.75%

Stopping Condition:

Difference of Loss = 10^-4 or number of epochs<10000

Hidden Layer Size 50:

Time for convergence - 876 sec
Train Accuracy - 86.78%
Test Accuracy - 81.3%

Stopping Condition:

Difference of Loss = 10^-4 or number of epochs<10000

Hidden Layer Size 10:

Time for convergence - 770 sec
Train Accuracy - 36.44%
Test Accuracy - 35.68%

Stopping Condition:

Difference of Loss = 10^-5 or number of epochs<10000

Hidden Layer Size 5:

Time for convergence - 304 sec
Train Accuracy - 18.06%
Test Accuracy - 17.38%

Stopping Condition:

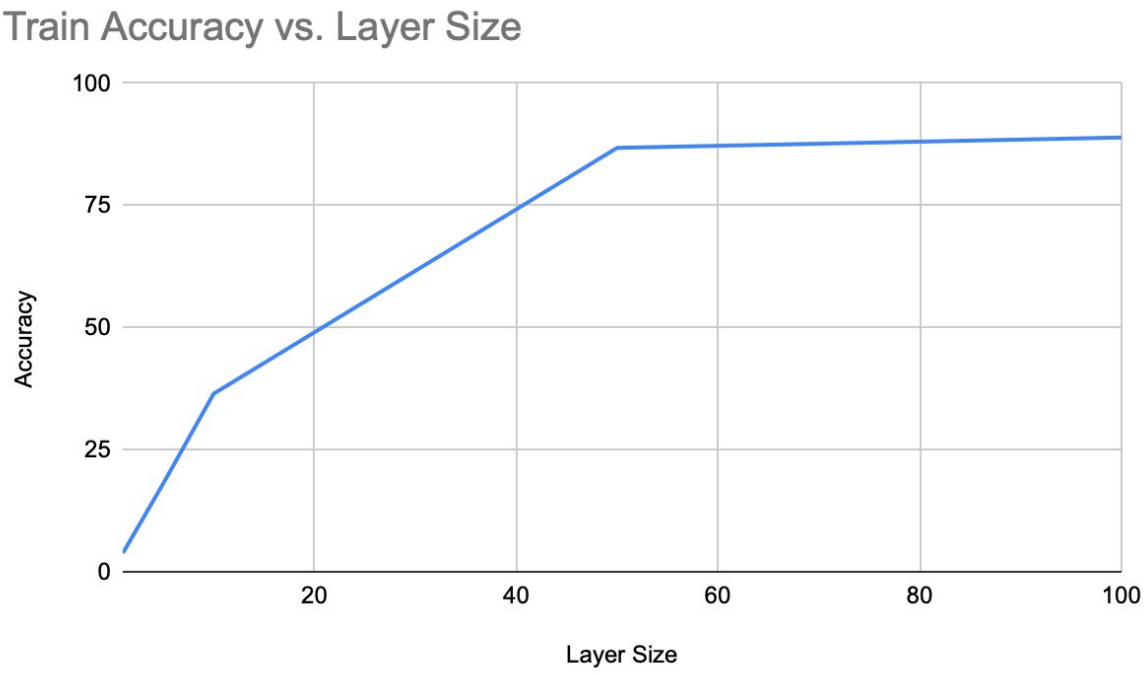
Difference of Loss = 10^-5 or number of epochs<10000

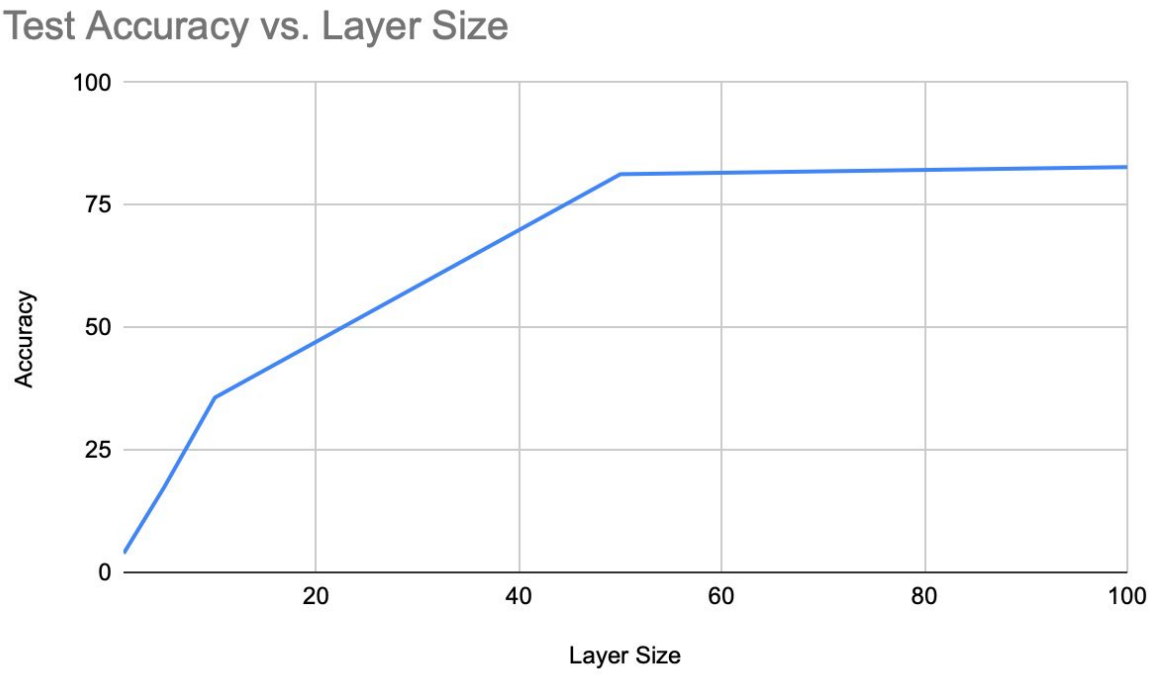
Hidden Layer Size 1:

Time for convergence - 91 sec
Train Accuracy - 3.82%
Test Accuracy - 3.86%

Stopping Condition:

Difference of Loss = 10^-6 or number of epochs<10000





With hidden layer size as 1/5 , the accuracy is much closer to random output. The cumulative error for an epoch for hidden layer size 1 also doesnt go below 58.

The training in adaptive learning case is much slower for a few reasons, the computation of square root might increase the implementation time. Also as the learning rate keeps on decreasing at a steady rate, the learning hence the convergence takes more time than a fixed learning rate. Stopping conditions has been modified in certain cases, mostly decreased because adaptive learns much slower than constant learning and hence there is a higher chance of stagnancy at a low derivative point.

RELU Implementation

ReLU is applied on all layers except for the last layer(sigmoid). Now each Delta(J) is updates as (current Delta(J)*Derivative of activation of layer output). The derivative will consist of a matrix consisting of 1's for values > 0 and 0's for values <=0. This is the only difference as compared to the sigmoid implementation.

The accuracy in this case increases as compared to sigmoid. 91.5% for test data and 98.66% for training data for the hidden layer architecture (100,100). One major benefit is the reduced likelihood of the gradient to vanish. In contrast, the gradient of

sigmoids becomes increasingly small as the absolute value of x increases. The constant gradient of ReLUs results in faster learning. Also it is computationally much easier to compute. It went to the full 10000 epochs, being very close to converging but didn't exactly converge(which shows scope of more better training!). It took 1312 sec for it to run. That time is probably because this time the neural network was deeper so it took more time to train for each epoch. Also the learning rate is very slow(0.005) at the end of 10000 epochs so learning is pretty slow. This accuracy is definitely better than both (b) and (c) part hence ReLU with last layer sigmoid was definitely a better choice.

MLP Classifier

classifier = MLPClassifier(hidden_layer_sizes=(100,100), max_iter=2000,learning_rate_init=0.1,learning_rate='adaptive', activation = 'relu',solver='sgd',random_state=1,batch_size=100)

Here are the arguments provided to the MLP Classifier. This classifier converges much faster than my original implementation and provides much better accuracy, so much so that it completely memorises the training data with 100% accuracy and 92.02% test accuracy. Prediction was done using *classifier.predict_proba(X_train/X_test).* The loss

function is Binary cross entropy as the Output values are fed as one-hot encoding and hence it sums up the losses for all the 26 alphabets will fitting.

The training using MLP Classifier is slightly better comparing in test data accuracy than the ReLU implementation and takes lesser epochs which is surprising. It might be better initialisation and/or stopping condition.The modificaition of loss function also improves this predicting independently the probability of each alphabet occuring.