ASSIGNMENT-3(b)

Abhyuday Bhartiya
2017CS10321

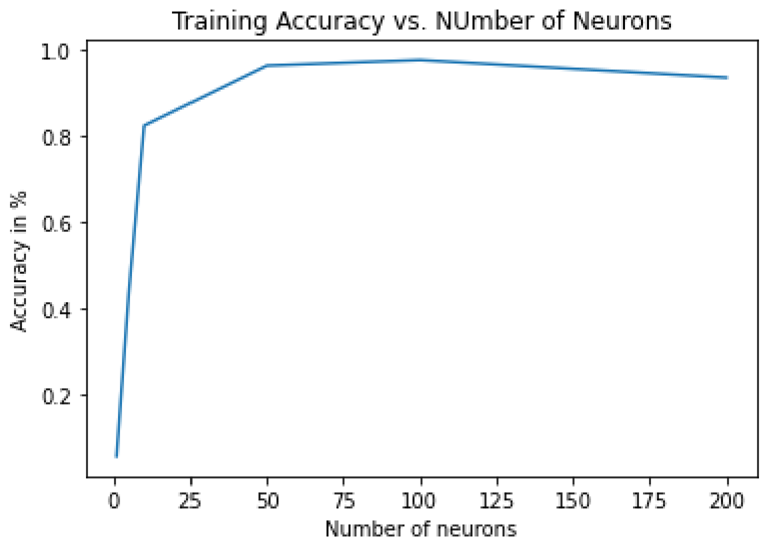
1. The architecture was implemented in the form of a class called Neural_Network . The initializer had 4 parameters :-- mini_batch_size , num_features, hidden_layer_architecture and target classes . Vector variables called weights and bias were initialized according to these parameters . Besides the class had the following function :--

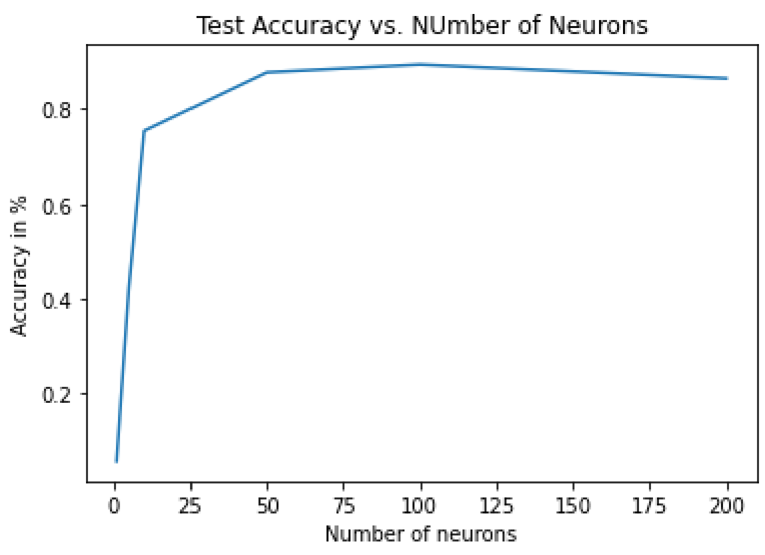
    A. Forward -- This had the input data (X) as a parameter to the function and it generated the values at each layer for the input data and stored in the form of a list with values for each layer appended as they are calculated. **This list is stored as a class variable called values** . Returns the predicted probabilities of target class as output .
    B. Backprop - Backpropagation to implement weight and bias gradients are calculated in this function using the class variables -- weights , bias and values . The function returns these weight and bias gradients .
    C. GetLoss - Function takes the true labels of data as input and calculates the value of loss function using the output values stored in list called values. Hence , getLoss uses values generated when function forward is called.
    D. Train - This function combines the functions forward , backpropagation and getLoss to train a neural network with a constant learning rate . This learning rate is an optional parameter to the function with default as 0.1 . The function requires input features(X) and true labels(Y) as input parameters .
    E. Train_adaptive - Essentially the same as function train but with an adaptive learning rate where initial rate is specified as an optional parameter to the function with default being 0.5 . This rate is inversely proportional to square root of epoch number as specified in part (c) of the assignment .

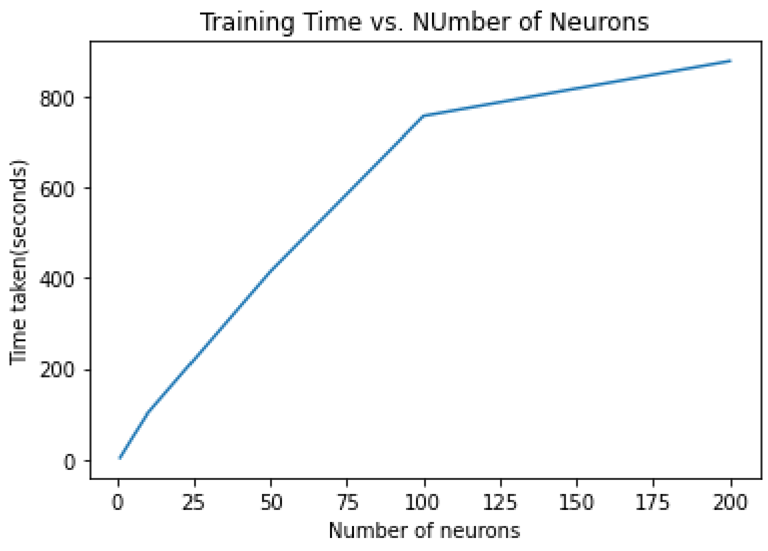
2. Using the class neural network as described in part (a) , we trained a neural network with activation function for all layers as sigmoid . For the stopping criteria , it wasn't feasible to compare the loss of a particular batch with another batch because those could vary significantly since the data is different . So , in **each epoch we sum the loss functions to get an aggregate epoch loss . If this loss does not change by more than 0.05% for more than 10 consecutive epochs , we believe that the function has converged .** Hence , we stop the training .

We can observe from the following data that the training and test data increases as the number of units in the hidden layer increases until the number of hidden units reaches 100 after which it sightly falls . Yet , **we can say that training and test accuracy increases as the number of units in hidden layer increases.** Yet , the improvement seems to **plateau off as the number of units increase** and we can observe a very small growth of about 1.5% in both training and test accuracy as we move from neurons being 50 to 100 .

Number of Neurons	Number of Epochs	Time taken (in seconds)	Training accuracy (in %)	Test accuracy (in %)
1	32	4.27	3.96	3.95
5	286	48.07	43.71	42.27
10	562	101.55	82.42	75.37
50	1361	414.53	96.38	87.71
100	1598	756.84	97.67	89.35





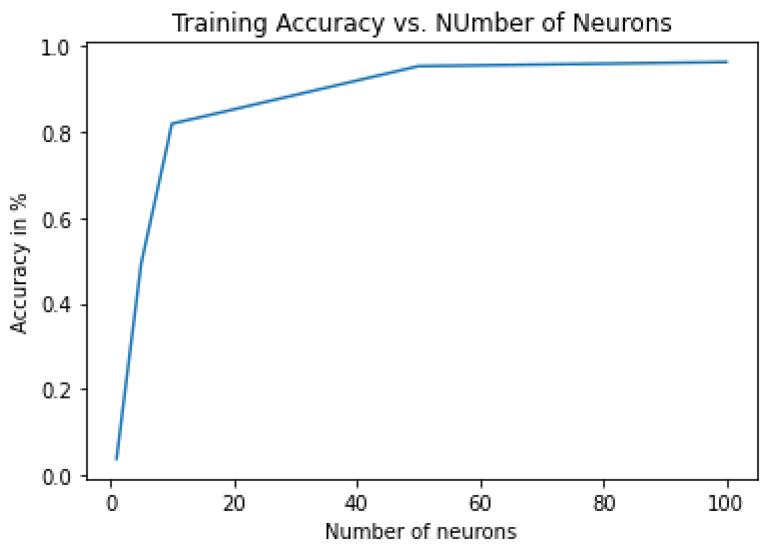


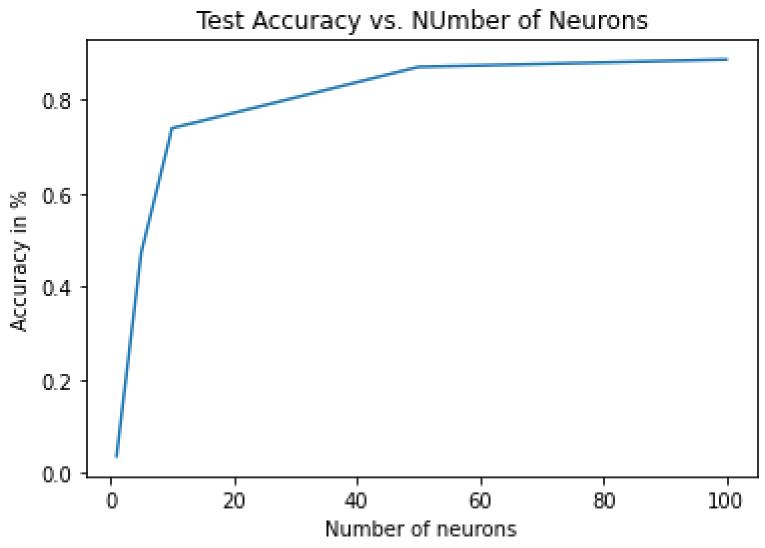
3. For this part , the implementation was similar to part (b) , except that we used an adaptive learning rate. In terms of implementation, that means we called the train_adaptive method of the Neural Network class rather than the train method . For the stopping criteria, using the same criteria as part (b) led to subpar performance on the test set so we changed the stopping criteria limit to be **If this aggregate epoch loss does not change by more than 0.005% for more than 10 consecutive epochs , we believe that the function has converged**

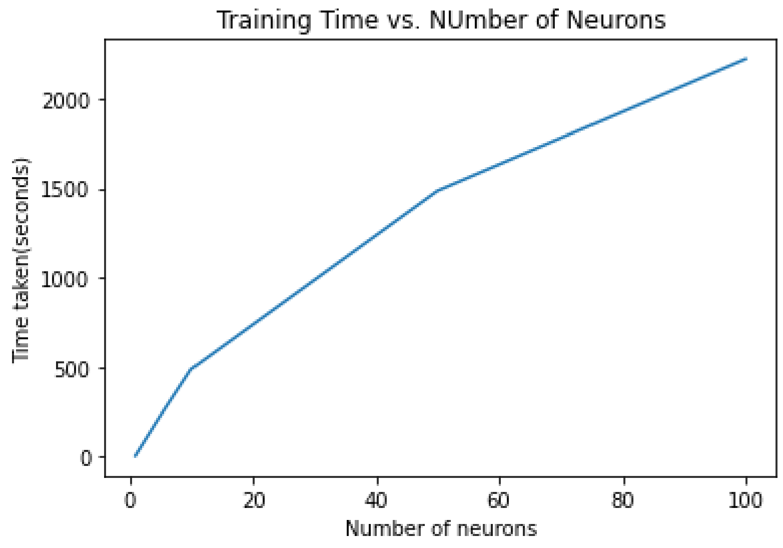
From the data , we can observe that the training accuracy and test accuracy is almost same as that observed in part (b) but seems to be **slightly lesser than those observed in part(b) for the same number of neurons in hidden layer** . A possible explanation for this can be that since the learning rate decays rapidly for such an adaptive rate , the change in network weights and biases also becomes very smaller as epochs increases . **Hence the change in output is also smaller and therefore the loss function does not change much at higher epochs too which leads to the stopping criteria being met** . To improve performance , lesser benchmark for stopping than 0.005% was also tried but led to even higher training time

The **training time does not seem to improve either in part(c) compared to the learning rate being constant** . This could be because the initial learning rate of 0.5 becomes 0.1 after the 25th epoch and becomes very small by say the 400th epoch . Due to this the loss function reduces very very slowly and takes a larger time to converge .

Number of Neurons	Number of Epochs	Time taken (in seconds)	Training accuracy (in %)	Test accuracy (in %)
1	33	4.45	3.93	3.58
5	1326	224.09	49.33	47.22
10	2712	486.21	81.82	73.82
50	4839	1485.01	95.24	86.98
100	8532	2220.61	96.2	88.59







4. For using RELU as activation function , I implemented a different class with the same functions as those described in part (a) but with activate units as RELU . Using 2 hidden layers with 100 neurons in both layers gives us the following results :---

Activation Unit	Epochs taken	Training accuracy (in %)	Test accuracy (in %)
Sigmoid	9361	96.85	88.92
RELU	1178	97.71	91.34

We can observe that the **performance of network with RELU activation unit is better in terms of training accuracy , test accuracy as well as number of epochs .**

Compared to results in part (b) , the Network with activation function as sigmoid performs slightly poorly (88.92% vs 89.35%) but using RELU **we have improved performance by 2% from 89.35% to 91.34%** .

5. Using MLPClassifier of the sklearn library , we build a classifier with the activation unit as RELU and we obtain the following results :---

> Time taken is 345.86 seconds
> Training accuracy is 99.98%
> Test accuracy is 91.58%

Comparing the performance with part (d) , we see that the performance on the **training data is slightly lesser but comparable and for the test data it is very close and only differs by merely 0.24%** .