Col 774: Assignment3 Part B

Vivek Singh

2019MCS2574

Neural Networks

A: Implement Generic Neural Network

- English alphabets data-set containing 28x28 pixel images of 26 alphabets.
- Implemented a neural network which takes following input as parameters.

  Batch_Size : This is the batch size of data taken during each iteration of mini-batch SGD.

  Input feature : 784 in our case.

  Architecture : Depicts total hidden layers and number of neurons in each layer.

  Target_class : Total class in output layer. 26 in our case.

  ETA : Initial learning rate.

  Max_iter : Maximum iterations for convergence.

  ActivationMode : Activation function used either Sigmoid or Relu.

  LearningRate : Either normal i.e constant or adaptive.
- Since we are using SGD for cost minimization we use two randomness in the algorithm.

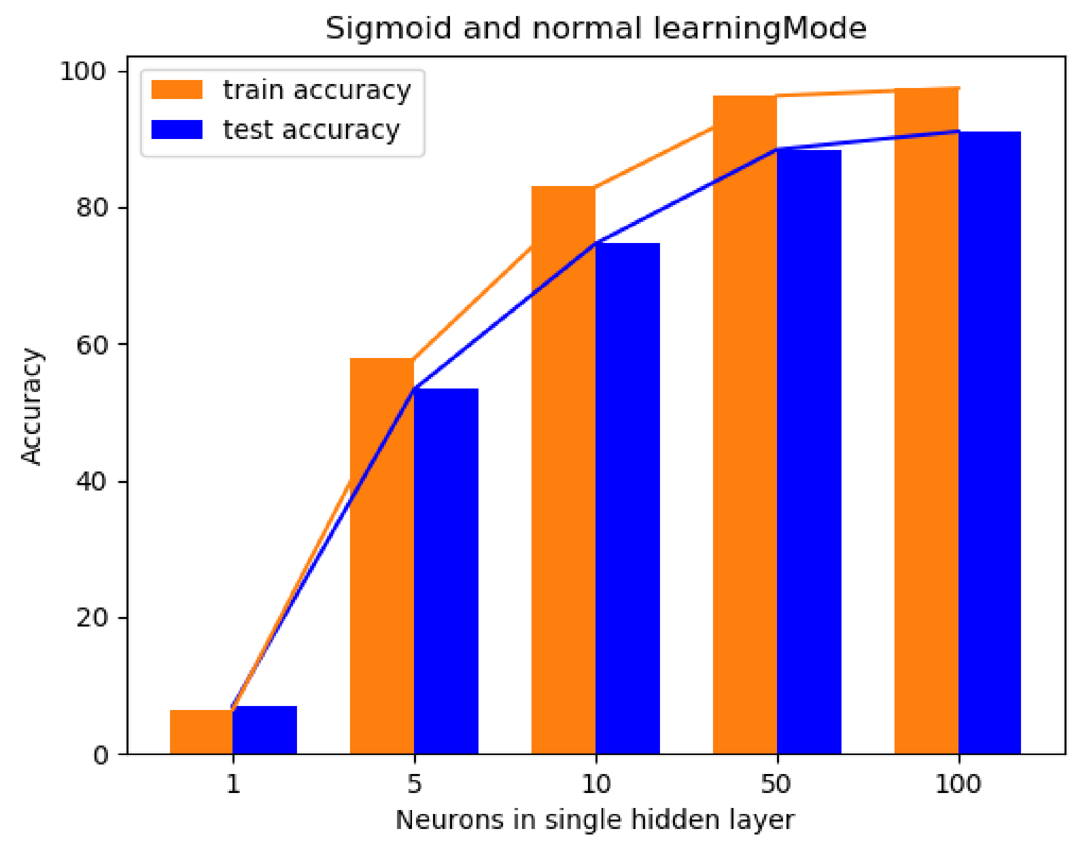
  After each epoch data is shuffled randomly for next iteration.

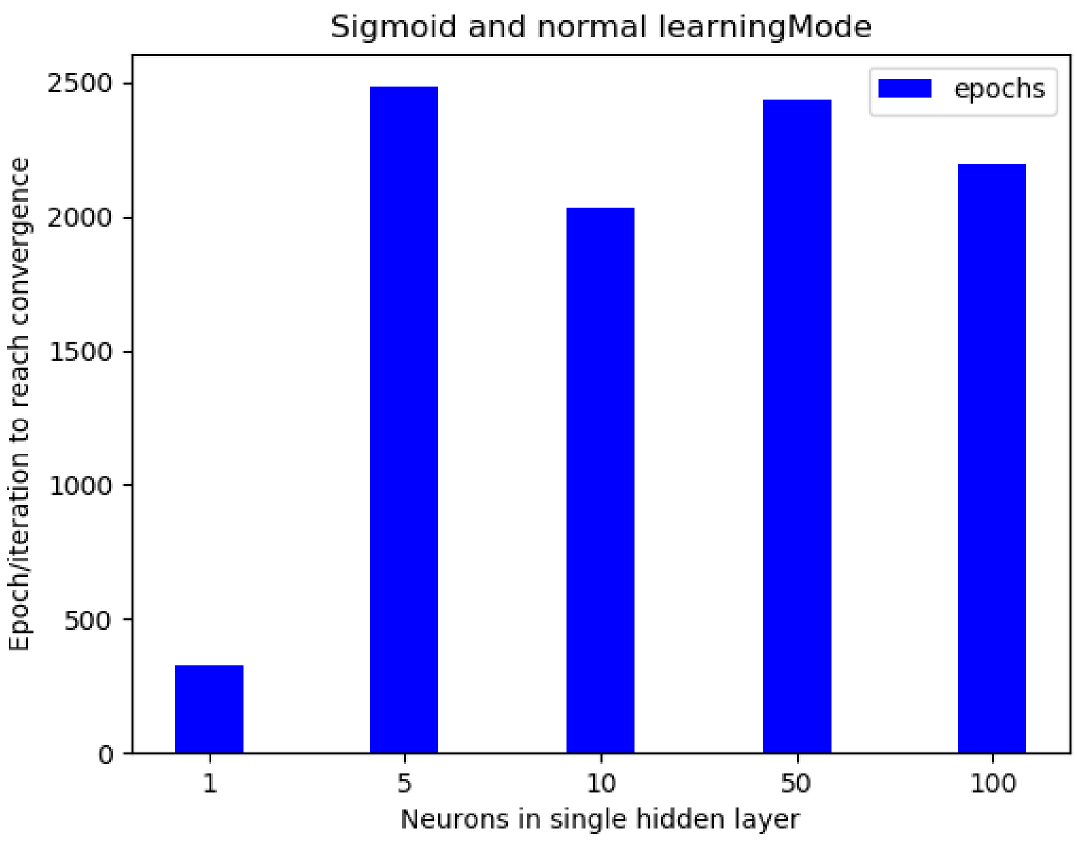
  Parameters are initialized by random numbers close to zero.
- Weights are initialized by random normal distribution of mean 0 variance 1.
- Bias are initialized to zero.
- The network weights are initialized using Xavier initialization.

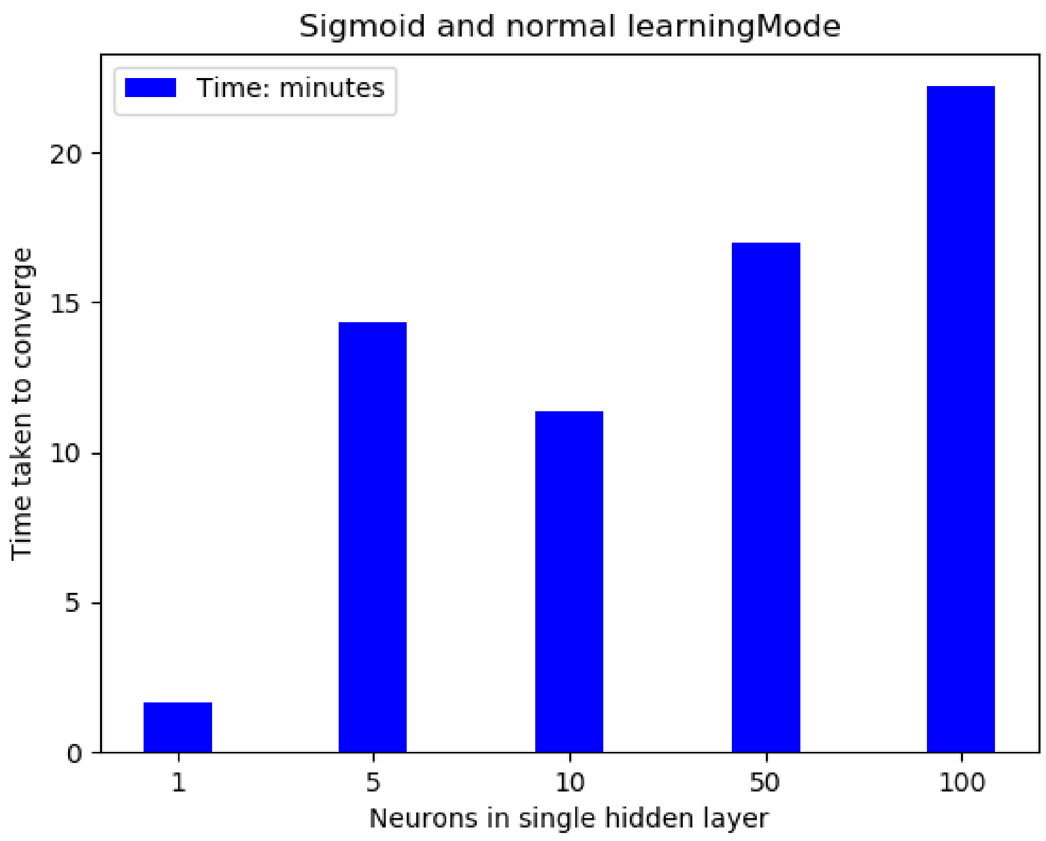
B: Neural Network: Single Hidden Layer

- In this part we will analyse neural network using single hidden layer, while varying neurons in that layer.
- **ETA :** learning rate is fixed to 0.1.
- **Batch_size :** Batch Size is fixed to 100.
- **ActivationMode :** Is set to Sigmoid.
- **LearningRate :** Is normal i.e constant throughout the network.
- Max_iter is set to 3000.
- **Hidden Layers :** [1, 5, 10, 50, 100] is used for number of neurons in single layer.
- **Stopping criteria :** Avg cost is calculated after each iteration i.e epoch. If the change in cost (improvement) is less than 10^{-5} for 3 consecutive iterations then the convergence is achieved.

PLOTS :


Sigmoid and normal learningMode
train accuracy
test accuracy
Accuracy
100
80
60
40
20
0
1
5
10
50
100
Neurons in single hidden layer


Sigmoid and normal learningMode
epochs
Epoch/iteration to reach convergence
2500
2000
1500
1000
500
0
1
5
10
50
100
Neurons in single hidden layer



- **Accuracy :**

Neurons	Train	Test
1	6.45 %	6.93 %
5	57.8 %	53.29 %
10	82.89 %	74.6 %
50	96.21 %	88.35 %
100	97.33 %	91.01 %

- **Convergence analysis:**

Neurons	Epoch	Time(Minutes)
1	329	1.6
5	2483	14.3
10	2031	11.39
50	2433	16.98
100	2193	22.2

- **Observations:**

  * Accuracy can be seen increasing with increase in number of neurons for both test and train throughout the architecture in the network.

  * We know that cost function is highly non-convex for neural networks with many local minima.

  * For architecture with 1 neuron it converges fastest and with fewest epoch. It signifies that the gradient descent get stuck to local optima very fast. Which signifies of small slopes in cost function thus successive iteration when unable to jump to better local optima. Hence gets converged.

  * For 5 and 10 neurons it takes more time and epoch due to sharper slopes in their cost function than 1 thus able to jump-off a local-optima to better one. But may have taken too much effort for finding next better optima before convergence thus resulting in more time and epoch.

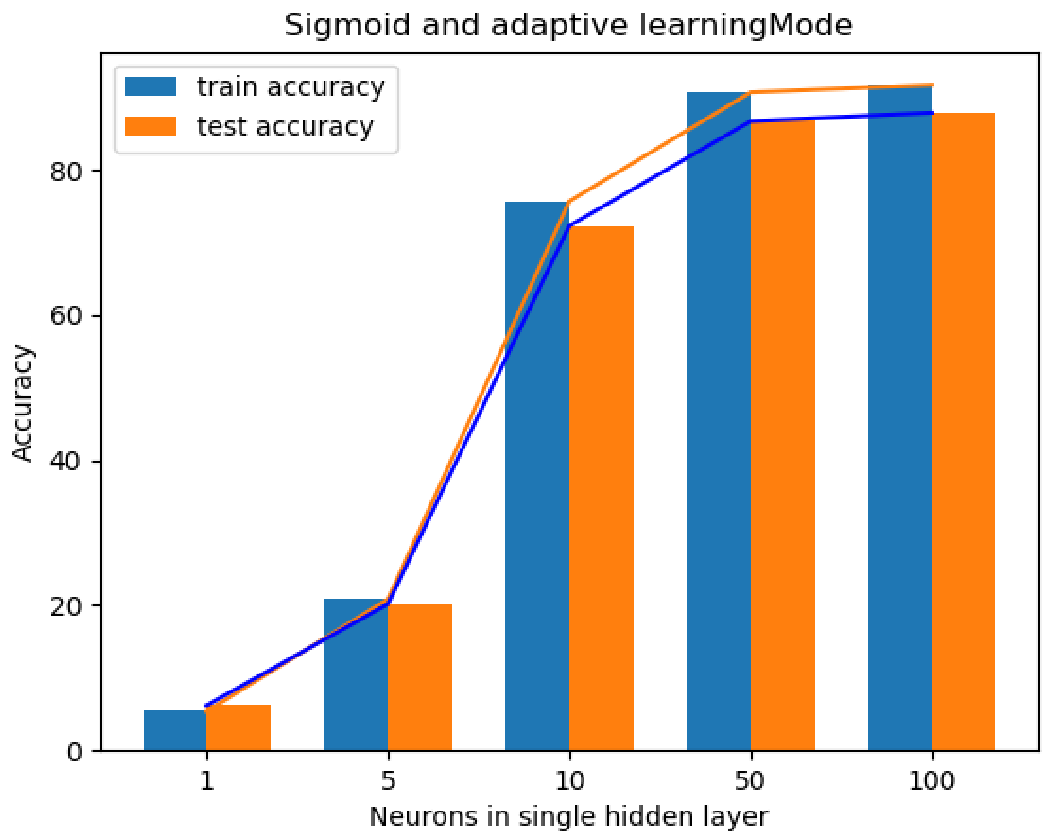
  * For 50 and 100 again results show that it's gradient is sharper than all architecture before and hence able to escape much more local optima to a much better one than previous models.

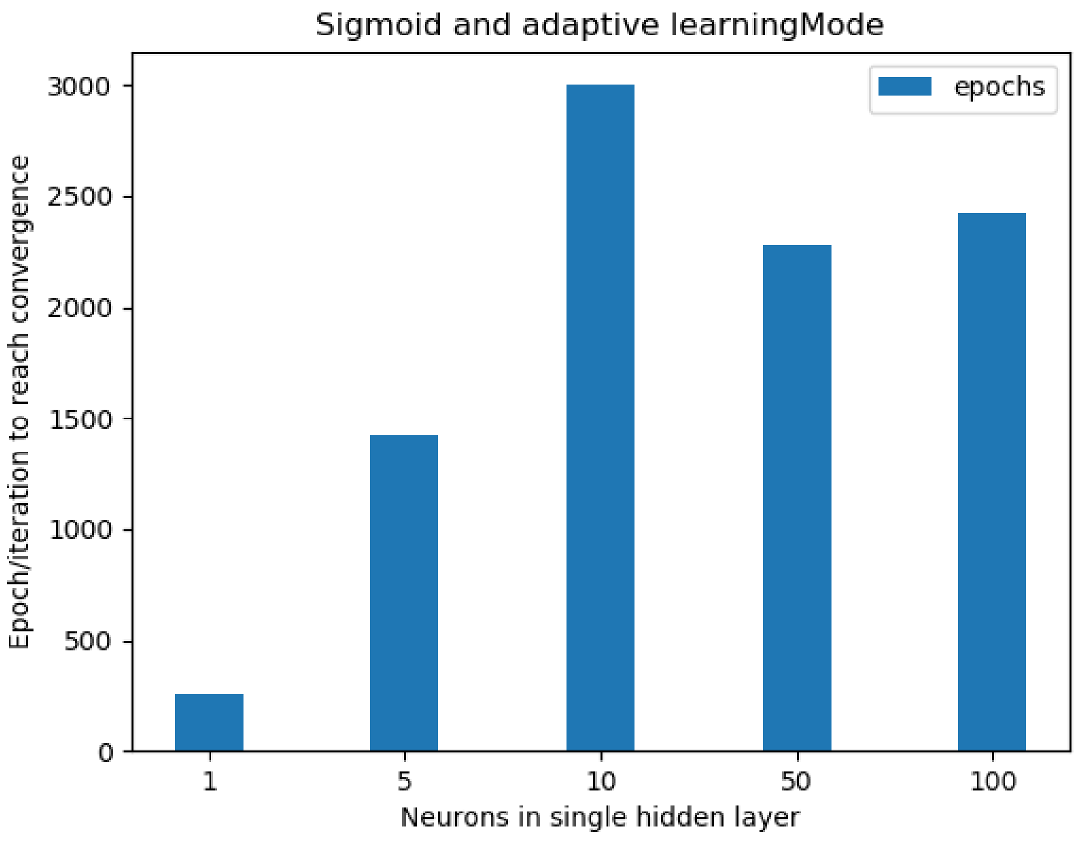
  * Thus we can infer that increasing neurons in the architecture gives the network much more ability to find a better optima. Hence able to learn very complex classification/learning problems.

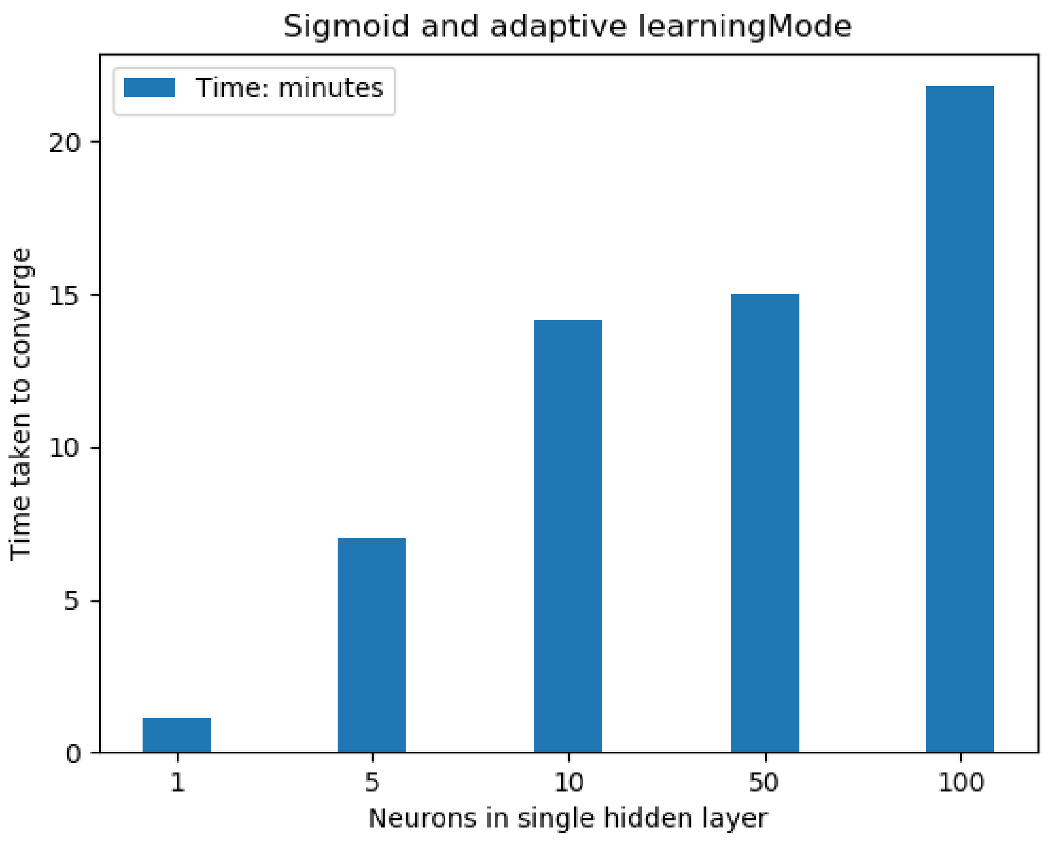
C: Adaptive learning i.e inversely proportional to epoch

- In this part we will repeat the part-b but with a new adaptive learning mode.
- Here in the new model we keep changing our ETA with every iterations/epoch. The new ETA will be inversely proportional to square-root of current epoch.
- **ETA :** Initial learning rate is fixed to 0.5 which is variable/iterations.
- **Batch_size :** Batch Size is fixed to 100.
- **ActivationMode :** Is set to Sigmoid.
- **LearningRate :** Adaptive.
- Max_iter is set to 3000.
- **Hidden Layers :** [1, 5, 10, 50, 100] is used for number of neurons in single layer.
- **Stopping criteria :** Avg cost is calculated after each iteration i.e epoch. If the change in cost (improvement) is less than 10^{-5} for 3 consecutive iterations then the convergence is achieved. Similar to part-B no change in convergence method.

PLOTS :


Sigmoid and adaptive learningMode
train accuracy
test accuracy
Accuracy
0
20
40
60
80
1
5
10
50
100
Neurons in single hidden layer


Sigmoid and adaptive learningMode
epochs
Epoch/iteration to reach convergence
0
500
1000
1500
2000
2500
3000
1
5
10
50
100
Neurons in single hidden layer



- **Accuracy :**

Neurons	Train	Test
1	5.39 %	6.2 %
5	20.83 %	20.16 %
10	75.66 %	72.27 %
50	90.67 %	86.69 %
100	91.69 %	87.84 %

- **Convergence analysis:**

Neurons	Epoch	Time(Minutes)
1	249	1.1
5	1421	6.9
10	2900	14.11
50	2276	15
100	2421	21

- **Observations:**

  * After replacing our constant ETA (learning Rate) in part B with a adaptive one. We find that all convergence time for all the architecture has dropped, except 10 where it is talking more time than before as well as more epochs.

  * We also find that there is drop in accuracy in both train and test data across all the architecture, which is justifiable as we are using this adaptive rate which decreases as the training proceeds for multiple epochs and is not later fixed at some minimum, thus it may suffer from the decay factor too.

  * Inconsistency and lack of similar patterns across the architecture points to the fact that we can tune learning rate to improve our model. Hence learning rate is a hyper-parameter of the network model.

D: Using Relu Activation in Neural Network

- In this part we replace our previously used sigmoid activation unit to introduce non-linearity in the network with Relu activation.
- Relu is widely accepted activation function as it overcomes many challenges faced in using sigmoid e.g saturation where all large outputs of neurons are squeezed to 1.0 and all small to 0.
- **ETA :** Initial learning rate is 0.5.
- **Batch_size :** Batch Size is fixed to 100.
- **ActivationMode :** Is set to Relu and Sigmoid for comparison.
- **LearningRate :** Adaptive.
- Max_iter is set to 3000.
- Convergence is again similar to all previous parts. I have used a similar initialization and convergence throughout different models trained in the assignment to bring consistency in the network.
- **Accuracy**

Activation	Train	Test
Relu	93.1 %	87.86 %
Sigmoid	91.5 %	87.3 %

- **Epoch to converge**

Activation	Epoch
Relu	729
Sigmoid	2957

- Sigmoid takes 35 minutes to converge whereas Relu takes 8.8 minutes for same.
- Here by comparing sigmoid and Relu we can conclude that relu performs better than sigmoid on same architecture, on both train and test data-set.
- We can also conclude that Relu is computationally faster than sigmoid as both activation and differentiation operation thus faster feed-forward and back-propagation.
- Another advantage of using Relu is its linearity of positive values which gives more learning as it does not suffer from saturation of weights.

- **Comparison with part B and C**

Architecture	Train	Test
B: [100] sigmoid	97.33 %	91.01 %
C: [100] sigmoid	91.69 %	87.84 %
D: [100,100] sigmoid	91.5 %	87.3 %
D: [100,100] Relu	93.1 %	87.86 %

- Part B sigmoid that uses constant fixed learning Rate(ETA = 0.1) performs better where single hidden layer of 100 neurons. Hence it can be said it does not suffer from decay factor.
- Part C and D both uses adaptive learning rate and same ETA = 0.5 initialization.
- Part C and D's sigmoid behaves very similarly although time taken by part D is more due huge number of parameters in it comparatively. Also it clears that just increasing the depth of network does not always lead to increase in performance.
- Relu outperforms both sigmoid using same learning rate signifying its importance as activation function.

E: MLPClassifier using Sci-kit learn library

- Using same architecture as in part D.
- Changed alpha = 0.0 in sci-kit learn as for comparison our model uses no L2 regularization.

Architecture	ETA	Train	Test	Epoch/iterations
Relu	0.5	87.38%	84.39 %	1126
Relu	0.1	93.7 %	89.03 %	733
Relu Part D	0.5	93.1 %	87.86 %	729

- MLP-classifer uses a different loss function than our implementation. We have used squared-mean error whereas MLP-classifier uses cross-entropy loss.
- So for binary-cross entropy over 26 class output, instead of passing train_y directly we rather pass one-hot-encoding of each class.
- Relu when using ETA = 0.5 was initially giving accuracy as random prediction. But changing the momentum to 0. It was able to perform much better. Since momentum is a hyper-parameter that MLPclassifier uses to push gradient to direction that it got from previous history to escape local optima, for our model this hyper-parameter default value's default value 0.9 doesn't seems to work.
- Comparing with part D and for ETA = 0.5 mlp-classifier does not give better accuracy and even takes more epochs to converge.
- But ETA = 0.1 and momentum =0.9 for other case seems to give comparable results as achieved in part D Relu.
- Again we can notice change in accuracy and epochs with different set of parameter initialization, pointing to the fact that these are hyper-parameters of the model and need to be tuned for best results.