COL774 Assignment 3 (Part-B)

Neural Networks

Het Shaileshkumar Patel (2019MCS2562)

April 28, 2020

Contents

List of Figures

List of Tables

1 Introduction

1.1 How to run code

I have submitted a file called Q1.py. You can pass following command-line arguments to this file.

- **-f|--data_folder <path>** : Path to parent folder where data is contained(i.e. the folder which contains given files train.csv and test.csv). If no value passed then it is assumed to be in `./data` directory.
- **-- part [any combination of b,c,d,e and f]** : which part of the assignment you want to run. e.x. you can run part b and c by passing --part bc. If not passed then all five parts are executed.

1.2 Data

In this assignment I have implemented MLP classifier for English alphabets dataset which contains 28x28 images flattened into vector of size 784 and output label ranging from 0 to 25 which represents alphabet a to z. Both train datset containing 13000 datapoints and test dataset containing 6500 datapoints are perfectly balanced. Now my model generates 26 probability outputs from output layers containing 26 sigmoid units so I have converted output label into one-hot representation of 26 dimension vectors.

2 Part A : Implementing MLP class

I have implemented MLP Classifier for the given problem in the `MLP` class in `MLPClassifier.py` file. Documentation of this class is as below (also given in the docstring)

```
Parameters:

        input_size(Default:784) : [int]
            Dimenstions of input dataset, also neurons in input layer

        layers([100,]) : [list]
            List containing number of hidden neurons in hidden layers
```

output_size(Default:26) : [int]

    Number of classes in output label

batch_size(Default:100) : [int]

    Batch size for mini batch SGD

lr(Default:0.1) : [float]

    Learning rate (Seed learning rate for adaptive learning rate)

adaptive_lr(Default:False) : [boolean]

    Flag suggesting use of adaptive learning rate.

    This is like inverse scaling learning rate and

    in each epoch learning rate will be lr/sqrt(n_epoch)

activation_fn(Default:'sigmoid') : [string]

    Activation function for neurons. Can pass either 'sigmoid'

    or 'relu'. Note that actiavtion function for neurons in

    output layer will always be sigmoid.

max_epoch(Default:1000): [int]

    Maximum number of epoch that SGD can run

tol(Default:1e-4) : [float]

    Tolerence that will be used for stopping criteria for SGD.

    If early_stopping=False:

        loss for n_iter_no_change is not reducing by tol then stop

    If early_stopping=True:

        validation accuracy for n_iter_no_change does not increase by tol then stop

n_iter_no_change(Default:10) : [int]

    Used for stopping criteria for SGD

```
        initialization(Default:'glorot-uniform') : [string]
            Method to use for weight initialization

            Available methods:
            1. 'glorot-normal':
                Normal distribution with mean=0 and scale=sqrt(2/(fan_in+fan_out))
            2. 'glorot-uniform':
                Uniform distribution with low=sqrt(6/(fan_in+fan_out))
                and high=sqrt(6/(fan_in+fan_out))
            3. 'he-normal':
                Normal distribution with mean=0 and scale=sqrt(2/fan_in)
            4. 'he-uniform':
                Uniform distribution with low=sqrt(6/fan_in) and high=sqrt(6/fan_in)
            5. 'normal':
                Normal distribution with mean=0 and scale=0.05
            6. 'uniform':
                Uniform distribution with low=-0.05 and high=0.05

        early_stopping(Default:False) : [boolean]
            if Ture then set aside 10% of Train dataset as validation dataset
            and then use accuracy on that validation dataset in each epoch for
            stopping criteria for SGD

        loss_fn(Default:'squared-loss') : [string]
            Loss_function used for optimization problem
            Available options are 'squared-loss' and 'cross-entropy'

Attributes:

        layers : [list]
            List containg number of neurons in each layer including input and output layers
        n_layers : [int]
            number of layers in network including input and output layers
        weights : [list]
```

```
            list of length (n_layers-1) containing weight matrices
        intercepts : [list]
            list of length n_layers containing intercept lists
        loss_lst : [list]
            list of accumulated loss calculated in each epoch

Methods:

        fit(self, X, Y) : [None]
            This method trains MLP using batch SGD. If you feelmlike stopping training
            anytime and get the weights learned till that time just Interrupt and the
            training will be stopped(Here Y should be one-hot encoded matrix)

        predict(self, X) : [array]
            Returns predicted one-hot encoded output matrix generated by our model
            shape : (n_samples,n_classes)

        predict_proba(self, X) : [array]
            Returns probabilities generated for each class label by our model
            shape : (n_samples,n_classes)

        predict_log_proba(self, X) : [array]
            Returns log probabilities generated for each class label by our model
            shape : (n_samples,n_classes)

        score(self, X, Y) : [float]
            Returns accuracy over dataset X and ground truth Y
```

3 Part B : Training MLP with single hidden layer and fixed learning rate

In this part I have used the class implemented in the previous part to generate MLP model to train 5 different models with one hidden layer with 1, 5, 10, 50 and 100 neurons.

3.1 Stopping Criteria

I the class I have implemented two different methods to check for convergence. One is using average loss over every iteration in one epoch and another one is using different validation set and accuracy on this validation set after every epoch. In both methods loss and score will be computed for every epoch and if the reduction in loss or increase in accuracy is less than tol for n_iter_no_change number of consecutive epochs then I will stop the training. Here I have experimented with both types of stopping criteria and documented respective metrics in the next section.

3.2 Output

Below table contains train accuracy, test accuracy, time taken and number of epoch for 5 cases with tol=1e-5, n_iter_no_chnage=50 and early_stopping=False.

Number of neurons	**Test Accuracy**	**Train Accuracy**	**Time(in minutes)**	**number of epochs**
1	8.91	8.83	0.66	576
5	58.25	61.45	2.86	2198
10	76.35	80.78	1.37	968
50	88.62	94.56	2.56	1193
100	90.34	96.33	3.48	1399

Table 1: Output with early_stopping=False & 0.1 constant learning rate

Below table contains train accuracy, test accuracy, time taken and number of epoch for 5 cases with tol=1e-5, n_iter_no_chnage=75 and early_stopping=True.

Number of neurons	**Test Accuracy**	**Train Accuracy**	**Time(in minutes)**	**number of epochs**
1	6.68	6.73	0.10	97
5	46.00	48.33	0.98	849
10	74.65	78.99	0.96	764
50	87.85	93.28	1.78	924
100	88.17	92.51	1.35	592

Table 2: Output with early_stopping=True & 0.1 constant learning rate

Here you can see that with early stopping our model converged in much lesser epochs in every case. But it gave lesser accuracy! So I will proceed with early stopping False as in former case we are training only on 90% data and as our model is not too deep with early stopping False we are able to get good results.

3.3 plots

These are the plots of number of neurons v/s time and number of neurons v/s train, test accuracy

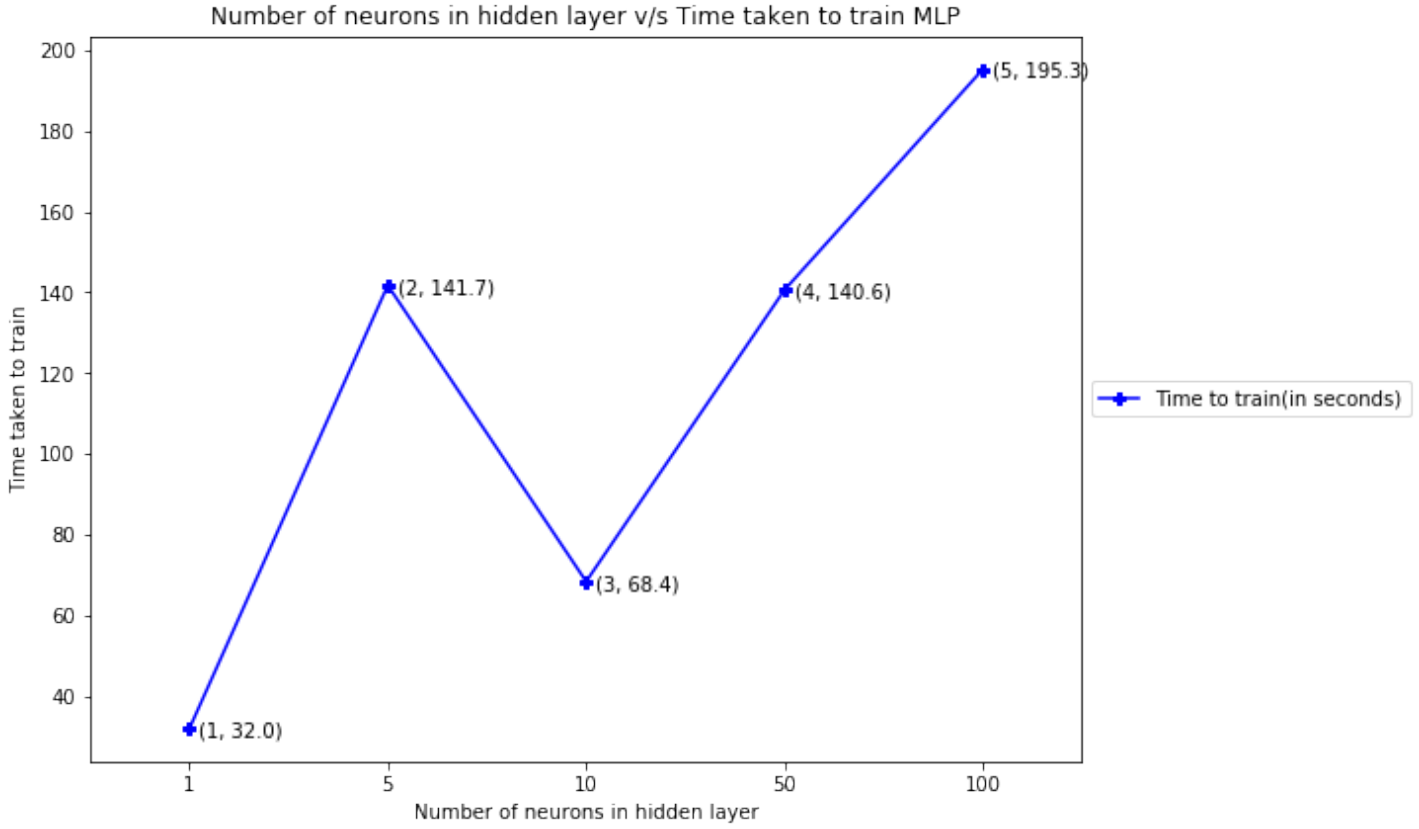


Figure 1: Number of neurons v/s time(in Seconds) to train model (with 0.1 constant learning rate)

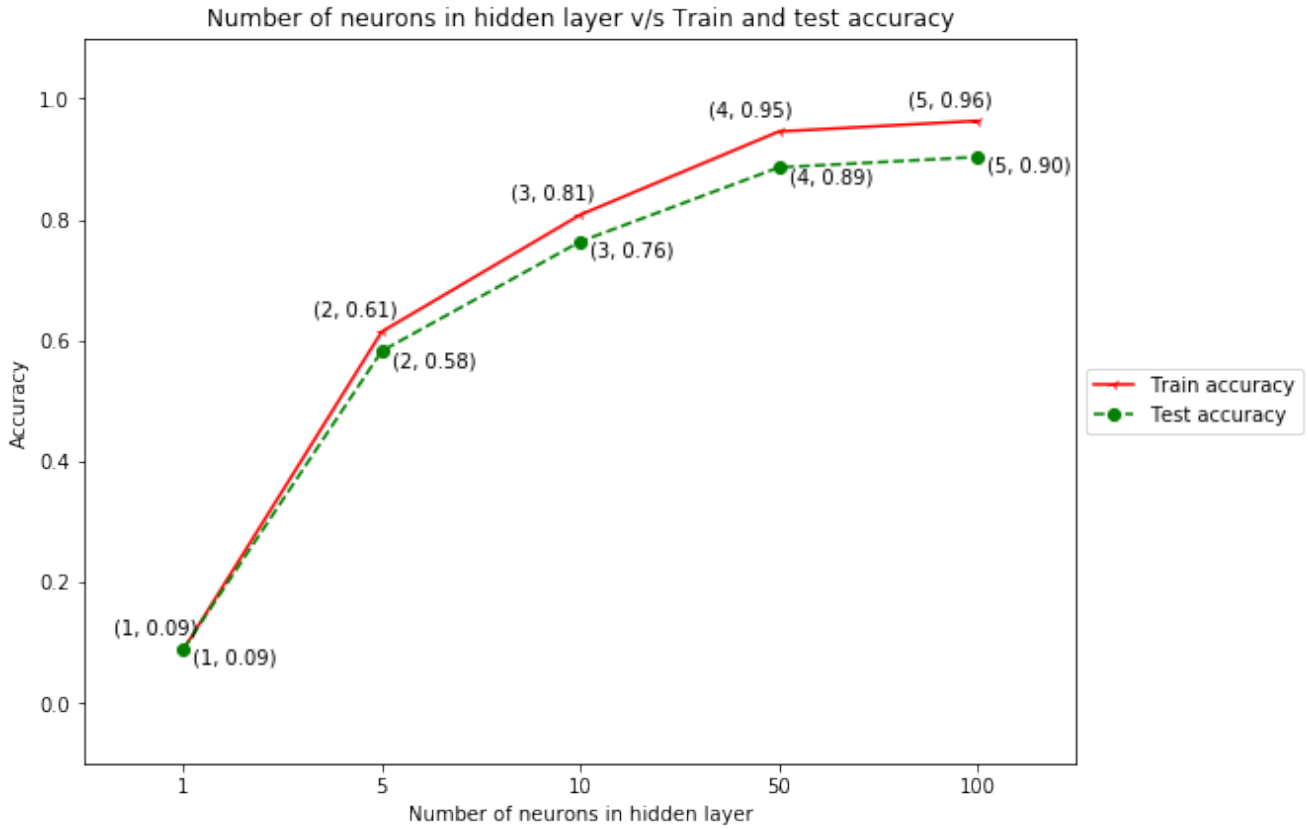


Figure 2: Number of neurons v/s Train & Test Accuracy (with 0.1 constant learning rate)

3.4 Observations

- Here we can see that MLP with 5 neurons is converging very slowly this might be due to saddle point or local minima as in this model we are solving a non-convex problem.
- As number of neurons increase we are getting better Train and Test accuracy.

4 Part C : Adaptive Learning Rate

In this part I have used adaptive learning rate to train 5 MLP models that we trained in the previous part. I have selected η_0=0.5 which means at any epoch e learning rate will be $\frac{0.5}{\sqrt{e}}$. Adaptive learning should help us meet the same convergence criteria in less number of epochs.

4.1 Output

Here I have trained 5 MLP models with same convergence criteria that we finalized in previous part i.e. tol=1e-5 n_iter_no_chnage=50 with early_stopping=False and now adaptive_lr=True. By keeping the same stopping criteria as in previous part I got nice accuracy and making this stopping

criteria tight will make our model to take more epoch to converge but also note that our learning rate will also be very small So running for more epochs was not helping in reducing loss much and it can be also shown as loss is not decreasing by more than 1e-5 for 50 consecutive epochs, which is a pretty tight constraint on its own.
In table below I have summarized the output.

Number of neurons	**Test Accuracy**	**Train Accuracy**	**Time(in minutes)**	**number of epochs**
1	7.57	7.57	0.37	330
5	23.37	23.38	1.22	964
10	71.00	73.26	1.75	1270
50	58.29	88.31	1.64	778
100	85.09	88.19	1.43	574

Table 3: Output with adaptive_lr=True & $\eta_0 = 0.5$

Here in below table I have compared number of epoch that took to converge in part with and without adaptive learning rate

Number of neurons	**Number of epochs without adaptive learning rate**	**Number of epochs with adaptive learning rate**
1	576	330
5	2198	964
10	968	1270
50	1193	778
100	1399	574

Table 4: Comparison of number of epochs with and without adaptive learning rate

4.2 Plots

These are the plots of number of neurons v/s time and number of neurons v/s train, test accuracy when I used Adaptive Learning rate.

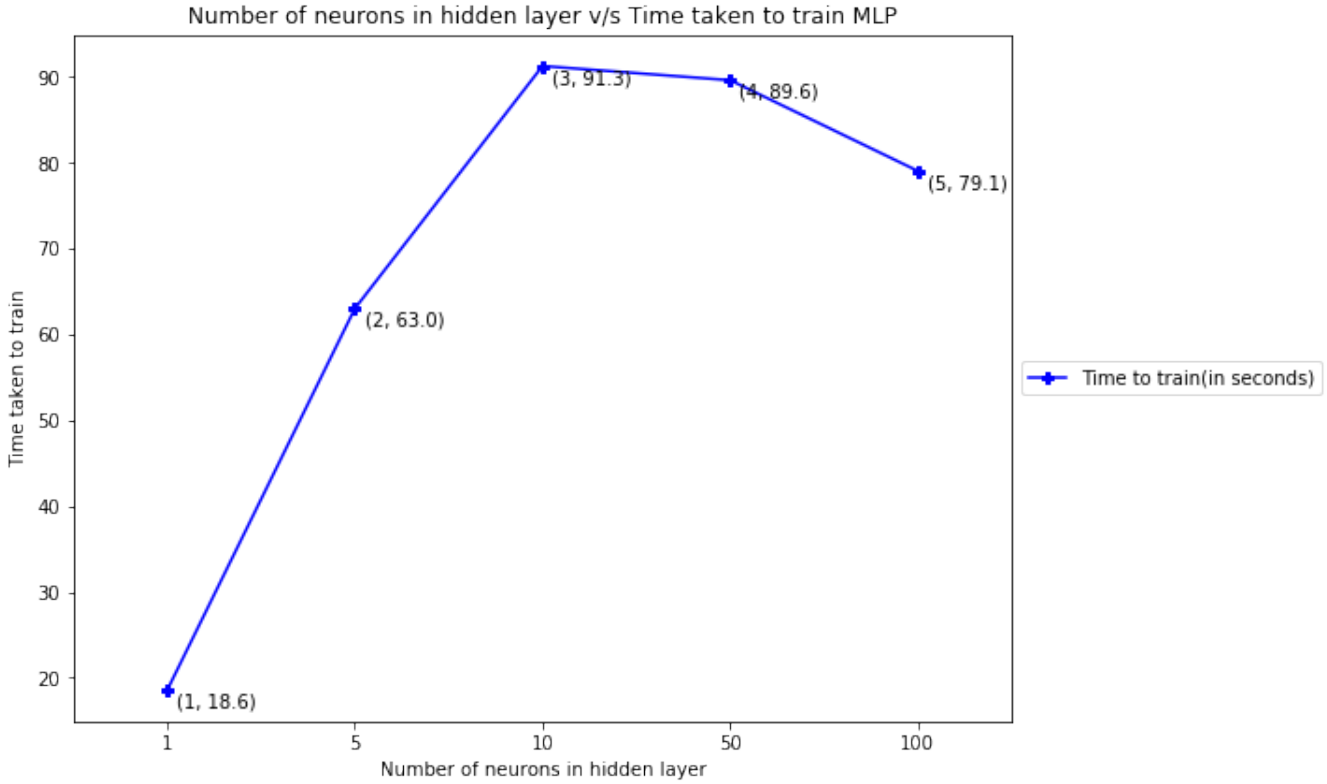


Figure 3: Number of neurons v/s time(in Seconds) to train model (with adaptive learning rate and $\eta_0 = 0.5$)

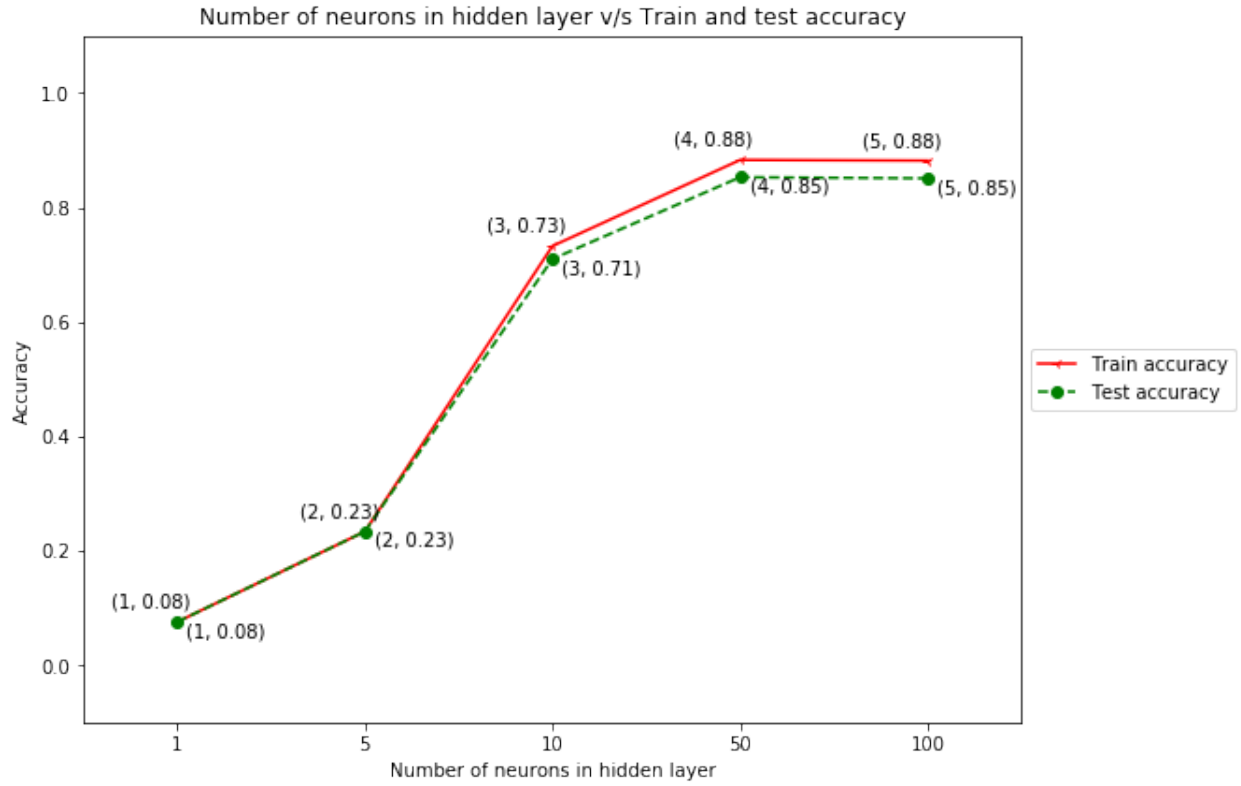


Figure 4: Number of neurons v/s Train & Test Accuracy (with adaptive learning rate and $\eta_0 = 0.5$)

4.3 Observations

- Here you can see that number of epochs to meet the same convergence is less than what we got in part B.
- We are seeing drop in accuracy though as learning rate is decreasing and becoming very small after some iterations so jumps SGD takes towards minima becomes smaller hence drop in loss becomes smaller and smaller.

5 Part D : ReLU activation function

In this part I have trained a MLP model with two hidden layers with 100 neurons each and ReLU(Rectified Linear Unit) as activation units for hidden layers and sigmoid as activation unit for output layer.

5.1 Output

I ran MLP model with 2 hidden layers with 100 neurons each with ReLU as activation function for hidden layers and also ran it as sigmoid as activation units to compare both activation functions. I have used adaptive learning rate with initial learning rate 0.5. I have kept the convergence criteria same as above two parts i.e. tol=1e-5, n_iter_no_change=50 and early_stopping=False. I have also tried training the same model using constant learning rate of 0.1 as we saw in the previous part that in keeping constant learning rate gives better accuracy.

Here I have summerized the output in the following table

Activation Function	**Test Acc.**	**Train Acc.**	**Time(in minutes)**	**n_epochs**
ReLU	91.25	97.15	2.35	750
Sigmoid	85.14	88.18	3.52	1227
ReLU (With constant lr=0.1)	91.09	97.61	1.31	417
Sigmoid (With constant lr=0.1)	91.14	97.17	4.58	1589

Table 5: Output for NN with two hidden layers with 100 neurons each

5.2 Observations

- Here we can see that we are getting slightly better accuracy than what we got in one hidden layer with 100 sigmoid neurons with two hidden layers with 100 neurons and ReLU activation units.

- When we use sigmoid activation units instead of ReLU and keeping every other parameters same we got much better accuracy with ReLU activation unit. And sigmoid is also taking much more epochs to converge.
- When we use constant learning rate instead of adaptive we get much better accuracy in case of sigmoid activation units. Whereas in case of ReLU accuracy got slightly dropped but it did converge in lesser epoch. So here we see constant learning rate performing better than adaptive learning rate.

6 Part E : Sklearn's MLPClassifier

Here in this part I have used scikit-learn's implementation of MLPClassifier to train a model with 2 hidden layers with 100 neurons each. I used the following parameters to train sklearn's model. `(hidden_layer_sizes=(100,100), solver='sgd', learning_rate='invscaling', learning_rate_init=0.5 max_iter=2000, batch_size=100, tol=1e-4, n_iter_no_change=10, alpha=0, momentum=0)` This trained a model with with ReLU activation units without any regularizer and momentum used in SGD so now the only difference should be the loss function.

6.1 Output

When I trained model by parameters given above I gave decent performance for ReLU activation units but sigmoid activation units just didn't converged and got very low accuracy. So I tried it again by changing learning_rate='adaptive' and keeping everything same and I got much better results for both activation functions. I have summarized output generate by running all four models in the table below.

Activation Function	**Test Acc.**	**Train Acc.**	**Time(in minutes)**	**n_epochs**
ReLU	84.40	87.38	4.64	1104
Sigmoid	27.00	26.78	9.83	2000
ReLU (With learning_rate='adaptive')	91.45	100.00	0.68	160
Sigmoid (With learning_rate='adaptive')	92.55	100.00	1.45	305

Table 6: Output with scikit-learn's implementation

6.2 Observations

- When we used invscaling as learning rate both models with ReLU and sigmoid didn't perform well(Sigmoid model was not able to learn anything). After changing it to adaptive we got much better results.

- One more thing to notice is that when we used invscaling as learning rate this model took considerably more time and epochs to converge than our implementation in part D.
- In later two models when we used adaptive learning rate then we got 100% Train accuracy so this model is not very good generalized model as we can say that it learned noise from the train samples. But this is expected when we disable regularizer by setting alpha=0.

7 Part F : Using cross-entropy as loss function

In this part I have implemented binary cross-entropy loss in part A. You can use it by passing loss_fn='cross-entropy'.

7.1 Outputs

I repeated every experiments that we have done till now in parts B, C and D but with cross-entropy loss. I have summarized all the outputs from all three parts in the tables below.

Number of neurons	Test Accuracy	Train Accuracy	Time(in minutes)	number of epochs
1	9.94	10.33	0.17	151
5	55.82	60.54	0.27	207
10	74.94	81.45	0.21	145
50	88.60	94.25	0.30	141
100	91.32	99.01	0.82	331

Table 7: Output of Part B with loss_fn='cross-entropy'

Number of neurons	Test Accuracy	Train Accuracy	Time(in minutes)	number of epochs
1	3.88	3.92	0.02	21
5	49.71	52.28	0.43	331
10	74.17	77.88	0.12	83
50	87.94	93.02	0.30	141
100	89.45	94.45	0.34	136

Table 8: Output of Part C with loss_fn='cross-entropy'

Activation Function	Test Acc.	Train Acc.	Time(in minutes)	n_epochs
ReLU	90.40	100.00	1.17	324
Sigmoid	90.37	96.16	0.64	196
ReLU (With constant lr=0.1)	92.80	100.00	0.60	158
Sigmoid (With constant lr=0.1)	92.14	99.86	1.09	330

Table 9: Output of Part D with loss_fn='cross-entropy'

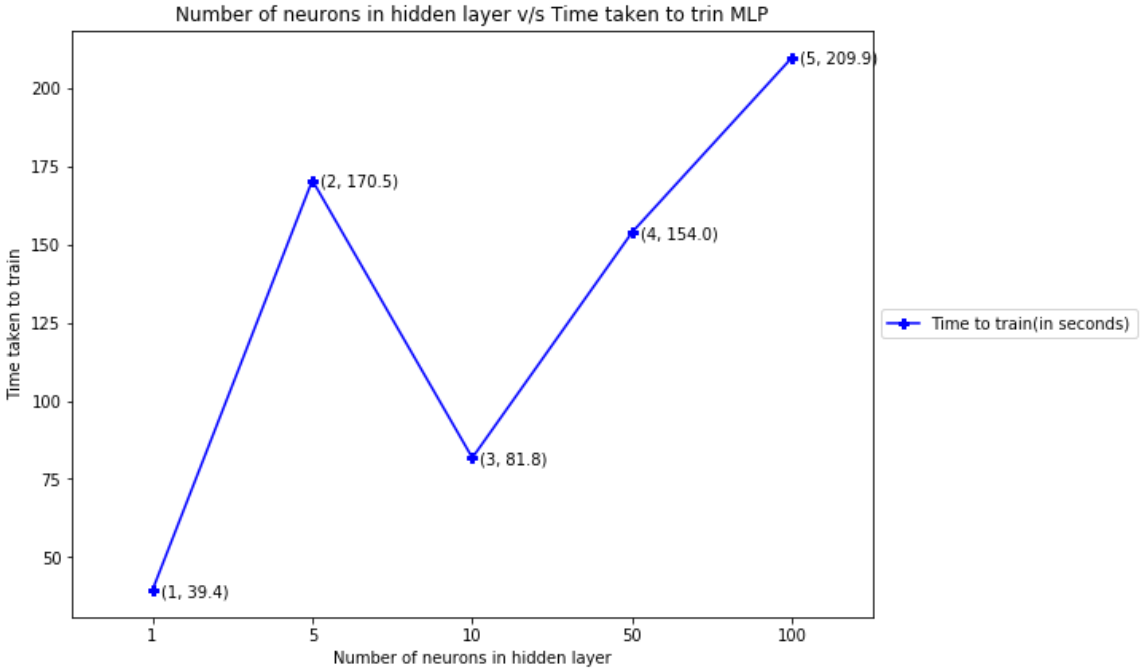
7.2 Observations

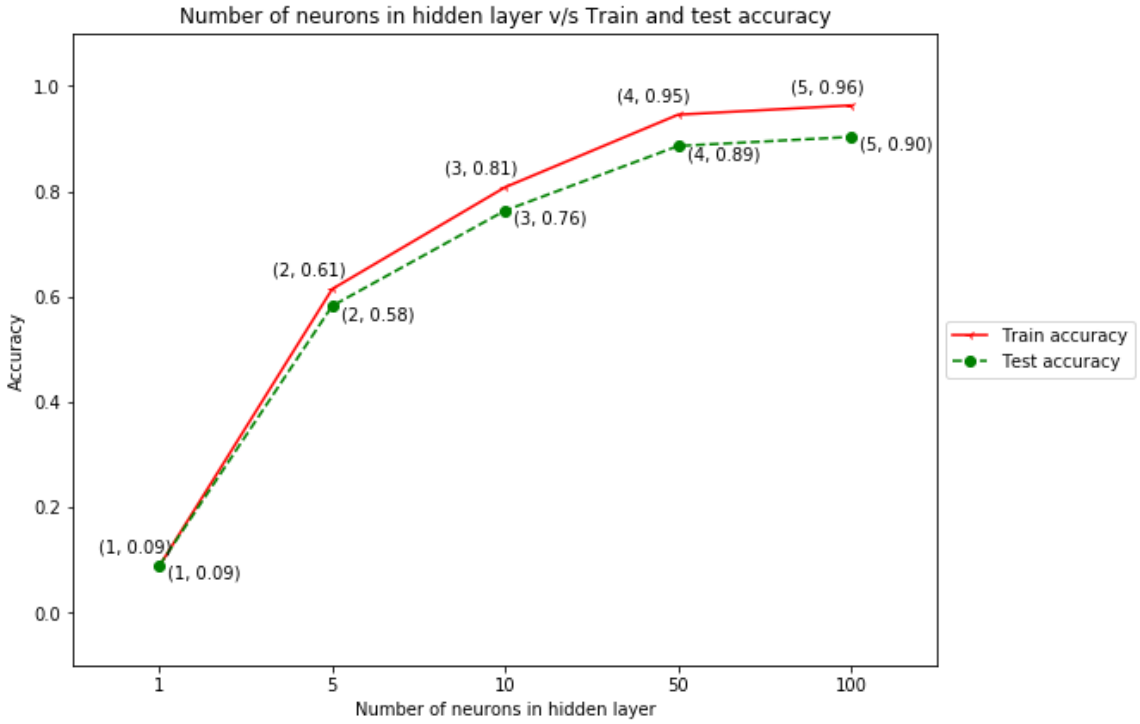
- Here by looking at output we can see that it is performing very well as compared to output that we got got in part B C and D both in terms of speed of convergence and accuracy.
- Here we can see that we are getting comparable or even better results than sklearn's implementation both in terms of accuracy as well as in terms of speed. We can see this especially in case of sigmoid unit with invscaling learning rate which was performing very poorly in sklearn's implementation but in out implementation we got much better accuracy.
- One more thing I noticed when I ran classifier in part D multiple times with squared loss, that our model's convergence is sensitive to weight initialization. Some times accuracy dropped significantly as model converged early as for 50 consecutive epochs loss was not reducing with adaptive learning rate. When I tried the same thing with cross-entropy loss instead of squared-loss in part F I got consistent results throughout with constant learning rate. So we can say that cross-entropy loss is also better in that sense.

8 Output of the submitted file

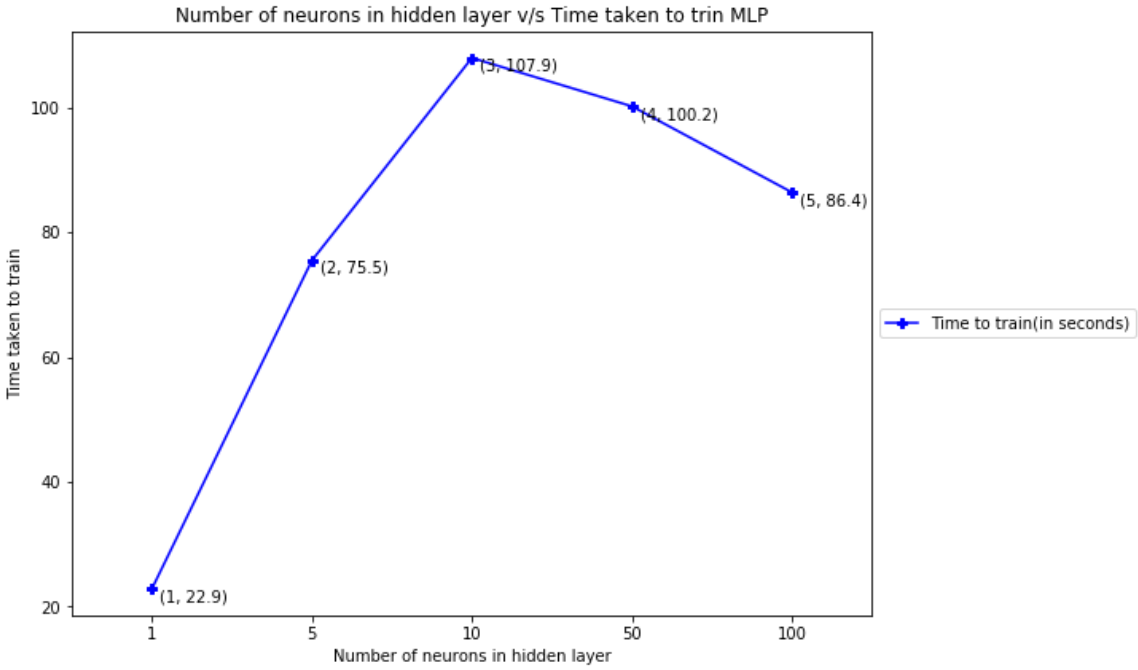
Here I have also attached the how the output looked like when I ran `python Q1.py -f ./data --part bcdef` command.

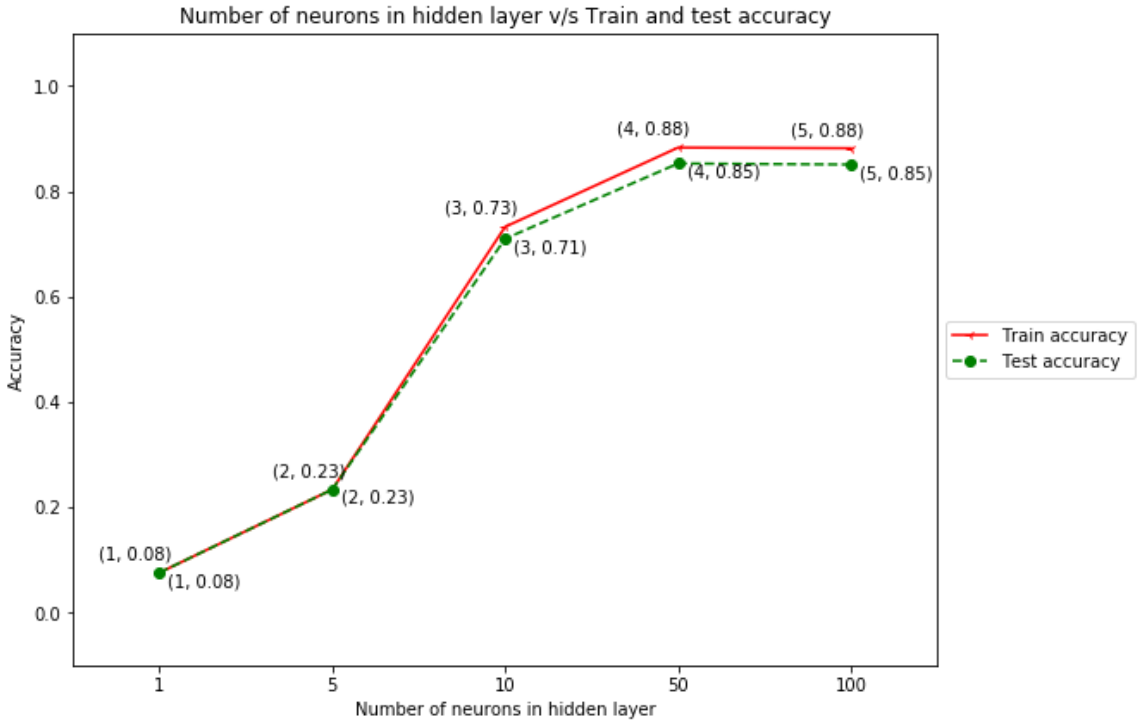
```
**********Loading Datasets**********


Shape of X_train (13000, 784)
Shape of X_test (6500, 784)


**********Running Part B**********


Running MLP with following parameters:

        input_size=784
        batch_size=100
        output_size=26
        lr=0.1
        adaptive_learning=False
        activation_fn='sigmoid'
        max_epoch=3000
        tol=1e-5
        n_iter_no_change=50
        initialization='he-uniform'
        early_stopping=False
        loss_fn='squared-loss'


Training MLP model with 1 neurons in hidden layer:
Converged in 576 epochs
Trained in 0.66 minutes
Test Accuracy : 8.91 %
Train Accuracy : 8.83 %


Training MLP model with 5 neurons in hidden layer:
Converged in 2198 epochs
Trained in 2.84 minutes
Test Accuracy : 58.25 %
Train Accuracy : 61.45 %


Training MLP model with 10 neurons in hidden layer:
Converged in 968 epochs
Trained in 1.36 minutes
Test Accuracy : 76.35 %
Train Accuracy : 80.78 %


Training MLP model with 50 neurons in hidden layer:
Converged in 1193 epochs
Trained in 2.57 minutes
Test Accuracy : 88.62 %
Train Accuracy : 94.56 %


Training MLP model with 100 neurons in hidden layer:
Converged in 1399 epochs
Trained in 3.50 minutes
Test Accuracy : 90.34 %
Train Accuracy : 96.33 %
```


Number of neurons in hidden layer v/s Time taken to trin MLP
Time taken to train
Number of neurons in hidden layer
Time to train(in seconds)
(1, 39.4)
(2, 170.5)
(3, 81.8)
(4, 154.0)
(5, 209.9)
50
75
100
125
150
175
200
1
5
10
50
100


Number of neurons in hidden layer v/s Train and test accuracy
Accuracy
Number of neurons in hidden layer
Train accuracy
Test accuracy
(1, 0.09)
(1, 0.09)
(2, 0.61)
(2, 0.58)
(3, 0.81)
(3, 0.76)
(4, 0.95)
(4, 0.89)
(5, 0.96)
(5, 0.90)
0.0
0.2
0.4
0.6
0.8
1.0
1
5
10
50
100

```
**********Running Part C**********


Running MLP with following parameters:

        input_size=784
        batch_size=100
        output_size=26
        lr=0.5
        adaptive_learning=True
        activation_fn='sigmoid'
        max_epoch=3000
        tol=1e-5
        n_iter_no_change=50
        initialization='he-uniform'
        early_stopping=False
        loss_fn='squared-loss'


Training MLP model with 1 neurons in hidden layer:
Converged in 330 epochs
Trained in 0.38 minutes
Test Accuracy : 7.57 %
Train Accuracy : 7.57 %


Training MLP model with 5 neurons in hidden layer:
Converged in 964 epochs
Trained in 1.26 minutes
Test Accuracy : 23.37 %
Train Accuracy : 23.38 %


Training MLP model with 10 neurons in hidden layer:
Converged in 1270 epochs
Trained in 1.80 minutes
Test Accuracy : 71.00 %
Train Accuracy : 73.26 %


Training MLP model with 50 neurons in hidden layer:
Converged in 778 epochs
Trained in 1.67 minutes
Test Accuracy : 85.29 %
Train Accuracy : 88.31 %


Training MLP model with 100 neurons in hidden layer:
Converged in 574 epochs
Trained in 1.44 minutes
Test Accuracy : 85.09 %
Train Accuracy : 88.19 %
```


Number of neurons in hidden layer v/s Time taken to trin MLP
Time taken to train
Number of neurons in hidden layer
(1, 22.9)
(2, 75.5)
(3, 107.9)
(4, 100.2)
(5, 86.4)
Time to train(in seconds)


Number of neurons in hidden layer v/s Train and test accuracy
Accuracy
Number of neurons in hidden layer
(1, 0.08)
(1, 0.08)
(2, 0.23)
(2, 0.23)
(3, 0.73)
(3, 0.71)
(4, 0.88)
(4, 0.85)
(5, 0.88)
(5, 0.85)
Train accuracy
Test accuracy

**********Running Part D**********

        Running MLP with following parameters:

        input_size=784
        n_layers=[100,100]
        batch_size=100
        output_size=26
        lr=0.5
        adaptive_learning=True
        activation_fn='sigmoid'
        max_epoch=3000
        tol=1e-5
        n_iter_no_change=50
        initialization='he-uniform'
        early_stopping=False
        loss_fn='squared-loss'

Converged in 1227 epochs
Trained in 3.80 minutes
Test Accuracy : 85.14 %
Train Accuracy : 88.18 %

        Running with same parameters as above except activation func
tion as relu in hidden layers

Converged in 750 epochs
Trained in 2.51 minutes
Test Accuracy : 91.25 %
Train Accuracy : 97.15 %

        Runnig both of above parts with learning rate as 0.1 constan
t

Running with activation function ReLU
Converged in 417 epochs
Trained in 1.40 minutes
Test Accuracy : 91.09 %
Train Accuracy : 97.61 %

Running with activation function sigmoid
Converged in 1589 epochs
Trained in 4.91 minutes
Test Accuracy : 91.14 %
Train Accuracy : 97.17 %

**********Running Part E**********

Training MLPClassifier with same architecture in part D and learning
_rate_init=0.5 and learning_rate='invscaling'

Trained in 4.82 minutes
Test accuracy 84.40 %
Train accuracy 87.38 %
Number of epochs till convergence 1104
Output activation logistic

```
Number of neurons in output layer 26
Loss type : log_loss
Loss Value : 0.8205594115010088

Running same thing as above with activation_fn='logistic'


/home/hetpatel/anaconda3/lib/python3.6/site-packages/sklearn/neural_
network/multilayer_perceptron.py:562: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (2000) reached and the optimization ha
sn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

```
Trained in 9.98 minutes
Test accuracy 27.00 %
Train accuracy 26.78 %
Number of epochs till convergence 2000
Output activation logistic
Number of neurons in output layer 26
Loss type : log_loss
Loss Value : 3.716546498809679

Running both part that I ran above with learning_rate='adaptive' and
learning_rate_init=0.5

With ReLU Activation function
Trained in 0.71 minutes
Test accuracy 91.45 %
Train accuracy 100.00 %
Number of epochs till convergence 160
Output activation logistic
Number of neurons in output layer 26
Loss type : log_loss
Loss Value : 0.001562106272307983

Running same thing as above with activation_fn='logistic'

Trained in 1.52 minutes
Test accuracy 92.55 %
Train accuracy 100.00 %
Number of epochs till convergence 305
Output activation logistic
Number of neurons in output layer 26
Loss type : log_loss
Loss Value : 0.00860725272302886


**********Running Part F**********


In this part I will be running part B,C and D with binary cross-entr
opy loss. Everything else remains same

Running Part B with binary cross-entropy loss(n_iter_no_change=10)

Training MLP model with 1 neurons in hidden layer:
Converged in 151 epochs
Trained in 0.20 minutes
Test Accuracy : 9.94 %
Train Accuracy : 10.33 %


Training MLP model with 5 neurons in hidden layer:
Converged in 207 epochs
Trained in 0.31 minutes
Test Accuracy : 55.82 %
Train Accuracy : 60.54 %


Training MLP model with 10 neurons in hidden layer:
Converged in 145 epochs
Trained in 0.23 minutes
Test Accuracy : 75.94 %
Train Accuracy : 81.45 %
```

```
Training MLP model with 50 neurons in hidden layer:
Converged in 141 epochs
Trained in 0.33 minutes
Test Accuracy : 88.60 %
Train Accuracy : 94.25 %


Training MLP model with 100 neurons in hidden layer:
Converged in 331 epochs
Trained in 0.89 minutes
Test Accuracy : 91.32 %
Train Accuracy : 99.01 %



Running Part C with binary cross-entropy loss(n_iter_no_change=10)

Training MLP model with 1 neurons in hidden layer:
Converged in 21 epochs
Trained in 0.03 minutes
Test Accuracy : 3.88 %
Train Accuracy : 3.92 %


Training MLP model with 5 neurons in hidden layer:
Converged in 331 epochs
Trained in 0.50 minutes
Test Accuracy : 49.71 %
Train Accuracy : 52.28 %


Training MLP model with 10 neurons in hidden layer:
Converged in 83 epochs
Trained in 0.13 minutes
Test Accuracy : 74.17 %
Train Accuracy : 77.88 %


Training MLP model with 50 neurons in hidden layer:
Converged in 141 epochs
Trained in 0.33 minutes
Test Accuracy : 87.94 %
Train Accuracy : 93.02 %


Training MLP model with 100 neurons in hidden layer:
Converged in 136 epochs
Trained in 0.37 minutes
Test Accuracy : 89.45 %
Train Accuracy : 94.45 %



Running Part D with binary cross-entropy loss(n_iter_no_change=10, t
ol=1e-4)


Two hidden layers with 100 neurons each with relu activation functio
n and adaptive learning rate with lr=0.5
```

```
Converged in 324 epochs
Trained in 1.20 minutes
Test Accuracy : 90.40 %
Train Accuracy : 100.00 %

Two hidden layers with 100 neurons each with sigmoid activation func
tion and adaptive learning rate with lr=0.5

Converged in 196 epochs
Trained in 0.65 minutes
Test Accuracy : 90.37 %
Train Accuracy : 96.16 %

Two hidden layers with 100 neurons each with relu activation functio
n and 0.1 learning rate

Converged in 158 epochs
Trained in 0.61 minutes
Test Accuracy : 92.80 %
Train Accuracy : 100.00 %

Two hidden layers with 100 neurons each with sigmoid activation func
tion and 0.1 learning rate

Converged in 330 epochs
Trained in 1.10 minutes
Test Accuracy : 92.14 %
Train Accuracy : 99.86 %
```