TECHNISCHE UNIVERSITÄT
BERGAKADEMIE FREIBERG

Die Ressourcenuniversität. Seit 1765.

Programming of Neural Network for the Classification of Images with custom libraries (without Machine Learning Libraries)

Personal Programming Project for winter term 2020

Submitted by:

VIDIT GUPTA: 61442

Under the valuable guidance of

P.D. Dr.-Ing. habil. Geralf Hütter

&

Dr.-Ing. Arun Prakash

Computational Material Science

Winter Semester of 2020

Mar 27, 2020

Acknowledgment

It would be my immense pleasure to extend my sincere thanks to Dr. Geralf Hütter and Dr. Arun Prakash for their valuable guidance, supervision and feedback through the entire course of this project.

VIDIT GUPTA: 61442

Contents

Introduction:

The objective of the Personal Programming Project is Programming of Neural Network for the Classification of MNIST image dataset by using custom libraries in python. (Not using any high-level machine learning library). The task of the project includes designing 3 Fully Connected Neural Network for classification with 0 hidden layers,1 hidden layer, and 2 hidden layers and comparing them based on their performance .i.e. the accuracy of the test dataset and selecting the best performing model.

The motivation behind the project is to learn more about deep learning techniques used for classification. Neural Networks are a pretty powerful tool for the classification of data (especially images). The same technique can also be implemented in other data related to computational material science as well. Knowledge of the functioning of the neural network is pretty general can be implemented in different fields as well.

Milestones at the time of initial presentation were set to program the 3 fully connected Neural Networks with 0 hidden layers, 1 hidden layer and 2 hidden layers with relu activation for the hidden layer, softmax activation in the final layer and use of stochastic gradient descent as an optimizer.

During the project, I was able to achieve all objectives set for completion of the project. Apart from that, I have added other features in the code like Momentum optimizer, L1 regression, Learning rate decay, sigmoid activation for the hidden layer, dropout and also carried out the study in the latter part of the report to analyze the models with these techniques and their effect on the accuracy of the models.

Brief Introduction about Neural Networks:

Construction of Neural Network is inspired by the biological neural network in the human brain. In recent days Neural Network has achieved significant benchmarks in speech recognition, computer vision, pattern recognition, text translation.

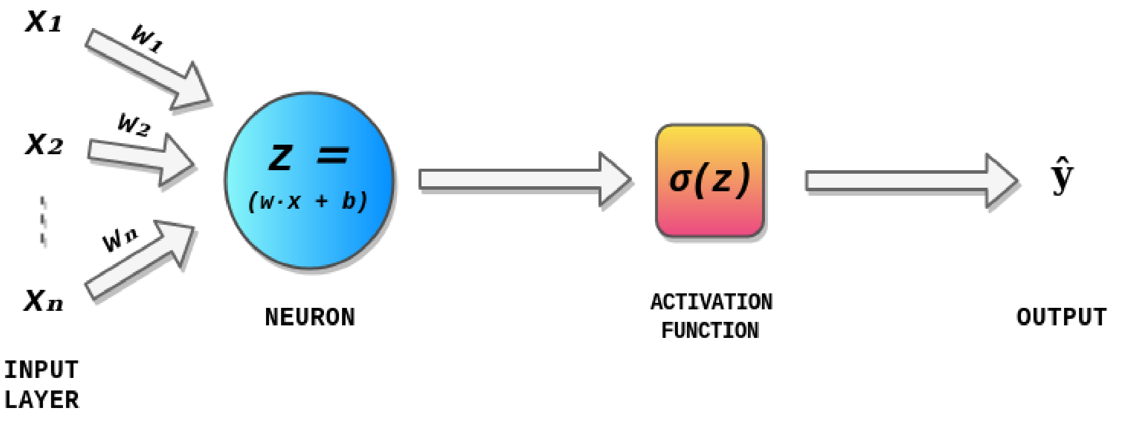


*Image is taken from the internet link https://hackernoon.com/a-6ur13zzx

A Single Neuron

The neuron is a basic component of the neural network. These are also called the building block of the network. Neuron gets its value from some other node or an external source and computes an output. The neuron is always associated with weights and biases which is responsible for the activation of the neuron. If we consider above example then the value of neuron will be

Calculate the value of neuron i.e.

$$Z = W1 \text{ x } X1 + W2 \text{ x } X2 + W3 \text{ x } X3 + \text{Biases}$$

And then apply the activation function to the value of Z. Activation function boost the performance of the neural network. (Reason: Activation functions are non linear functions. As with addition of non linear function in every layer neural network become non linear function overall which is much better than linear classifier).

There are many activation function used in Deep Learning field. Some of them are tan h function, sigmoid function, rectified linear unit activation function (relu), different version of rectified linear unit function like (leaky relu activation), softmax activation function

Activation function to the value neuron

$$A = \text{activation function } (Z)$$

Within the model I have designed in the project I have used three types of the activation function.

1. Rectified Linear unit (Relu) activation function:

$$A = \max(0, Z)$$

2. Sigmoid Activation function:

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}.$$

*image taken from Wikipedia

3. Softmax activation function

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \dots, K \text{ and } \mathbf{z} = (z_1, \dots, z_K) \in \mathbb{R}^K$$

*image is taken from Wikipedia

Relu and sigmoid activation functions are used for the hidden layer whereas the softmax activation function is used for the final output layer. The softmax activation function works well with a cross-entropy loss function. Within our model I have used cross-entropy loss function with multiclass with L1 regression. As an output from the neural network is 10 for each input we have use cross-entropy loss function with multiclass variant.

Cross entropy loss function:

$$\boxed{CE = -\sum_{i}^{C} t_i log(s_i)}$$

*Image is taken from link https://gombru.github.io/2018/05/23/cross_entropy_loss/

Within the model to update the weights and biases in each training step, I have used two optimizer.

1. Stochastic Gradient descent optimizer:
   Weights = Weights – learning_rate*dloss_dWeights
   Biases = Biases – Learning_rate* dloss_dBiases

2. Momentum optimizer:

   mov_Weight = beta * mov_Weight +(1 - beta)* dloss_dWeight
   Weights = Weights – learning_rate * mov_Weights

   mov_Biases = beta * mov_Biases +(1 - beta) * dloss_dBiases

Biases = Biases – learning_rate * mov_Biases

These optimizers use different algorithms to update weights and biases and reduce the test loss associated with the model.

Architecture of Models in Project:

In project work, I have designed the code which can create and train 3 fully connected neural network to classify images in MNIST dataset. (One network can be trained in single run).

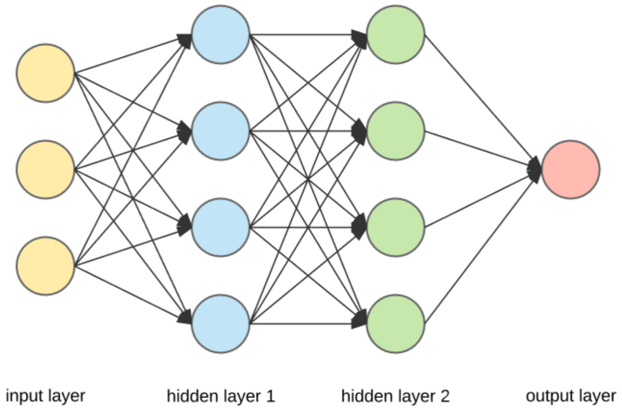


*image is taken form link https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

- Model with 2 hidden layer

  784 neurons in input layer
  64 neurons in 1st hidden layer
  64 neurons in 2nd hidden layer
  10 neurons in last layer

- Model with 1 hidden layer

  784 neurons in input layer
  64 neurons in hidden layer
  10 neurons in last layer

- Model with 0 hidden layer

  784 neurons in input layer
  10 neurons in last layer

Project code allows training and testing of one single model at a time. User can be creative and train its own model on MNIST dataset by using option for other techniques like.

- Choice for number of layer in neural Network [0 or 1 or 2]
- Choice for optimizer [SGD or Momentum]
- Choice for activation function inn hidden layers [Relu or sigmoid]
- Choice for data normalization algorithm [Simple or Normal]
- Choice for weights initialization algorithm [Gaussian or Xavier initialization]
- Choice for implementing L1 regression Loss [y / n]
- Choice for implementing decaying learning rate [y/n]
- Choice for number of epochs to train the model.

Project code in end displays and save the results in form of accuracy history graph per epoch, Loss history graph per epoch and learning rate history graph per epoch and also in form of .csv format

Project code saves trained weights and biases in end of training in working directory in .csv format.

Hyper parameters kept constant in the project are

- Batch size = 4
- Learning rate = 0.001
- Learning rate decay constant= 0.98
- Momentum parameter = 0.9
- Regression parameter = 0.00001
- Neurons in hidden layer = 64

MNIST dataset split used in the project and study in latter part of report is

Training dataset = 58,000 images

Validation dataset = 6000 images

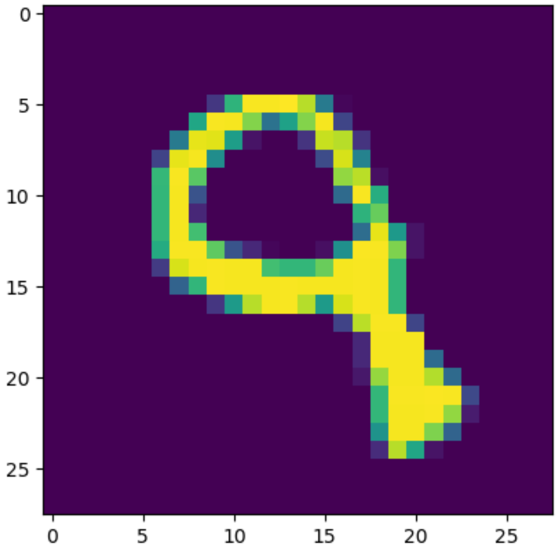
Test dataset = 6000 images

Some results from predict.py:

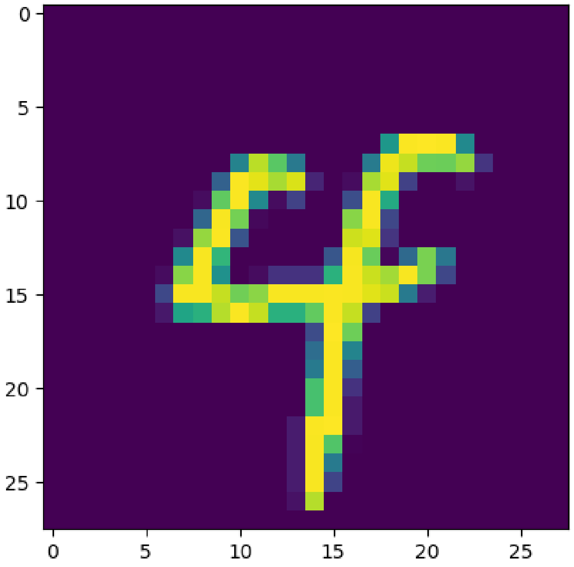
These images I get after running predict.py

Total Accuracy of the test dataset selected on the trained model is 96.25 %.

1. Image is an example of a wrong prediction by the model. Label predicted is 8 by model but the true label is 9.



2. The image is an example of correct prediction by the model. Label predicted is 4 by model but the true label is also 4.



Test results from verify.py:

Verify.py is a library of functions that carry out testing and validation operations on functions in train.py by comparing the output of the tested function with the final calculated values in valid_excel.xlsx file for a single input.

I have divided the test case used in verify.py in 3 categories.

1. Test Cases check individual functionality of function in train.py

   Digit associated with the cases from 1 to 16 falls in this category. In these test cases individual output of the function is tested with the know calculated output in valid_excel.xlsx file. I have run all the test cases associated with digits from 1 to 16 output of tested function matches with known calculated value in the excel sheet and all functions are tested okay.

2. Test Cases check specific functionality of train.py (involves output and input to more than one function.) for eg. Calculation of loss, Calculation of accuracy, Data pre-processing, the forward prop for loss.

   Digits associated with the cases from 17 to 35 falls in this category. In these test cases different functionality of train.py file is tested which involves more than one function and more than one option to execute the same task. I have run all the test cases associated with digits from 17 to 36 outputs of tested function matches with known calculated value in the excel sheet and all functions are tested okay.

3. Test Cases check the overall functionality of Neural Network (involves running the neural network for 5 epochs and checking the updated weights and biases, loss, accuracy of the network)

   Digits associated with cases from 37 to 50 falls in this category. These all test case execute the reduced down version of neural network (with 10 input layer, 10 hidden layers each and 10 output layer) for 5 epochs and then compare the parameter like weights, biases, loss and accuracy for the network from the known value in valid_excel.xlsx file. These test cases include training all 3 neural networks with the different optimizer, with L1 regression, with decaying learning rate, with different activation functions. I have run all test cases and all functions tested okay.

To know about each scenario of Test Case in detail please check Structure of verify.py on page 15 (page next to list). Each test case scenario is explained with function which is tested, with inputs and expected outputs and expected output comparison with calculated value in excel with same input to test the functionality of function is okay.

*List of verify.py functions with a digit associated with each function is on the next page

List of verify.py functions with a digit and excel sheet used to input the data:

Digit Associated with function	Name of the test function of verify.py file	Spreadsheet used to test the functionality of function in train.py
1	Relu Activation Test Case	Sheet1
2	Relu Activation back Test Case	Sheet2
3	Sigmoid Activation Test Case	Sheet3
4	Sigmoid Activation back Test Case	Sheet4
5	SGD optimizer for 2 hidden layers Test Case	Sheet5
6	SGD optimizer for 1 hidden layers Test Case	Sheet5
7	SGD optimizer for 0 hidden layers Test Case	Sheet5
8	Momentum optimizer for 2 hidden layers Test Case	Sheet6
9	Momentum optimizer for 1 hidden layers Test Case	Sheet6
10	Momentum optimizer for 0 hidden layers Test Case	Sheet6
11	Regression Loss 2 hidden layer Test Case	Sheet7
12	Regression Loss 1 hidden layer Test Case	Sheet7
13	Regression Loss 0 hidden layer Test Case	Sheet7
14	Learning rate decay Test Case	Sheet8
15	Normal Normalization Test Case	Sheet9
16	Simple Normalization Test Case	Sheet10
17	Loss function with reg 2 hidden layer Test Case	Sheet14
18	Loss function with reg 1 hidden layer Test Case	Sheet14
19	Loss function with reg 0 hidden layer Test Case	Sheet14
20	Loss function with no regression Test Case	Sheet14
21	Forward_prop_for_loss 2 hidden layer with relu activation with regression Test Case	Sheet15
22	Forward_prop_for_loss 2 hidden layer with relu activation with no	Sheet15

	regression Test Case	
23	Forward_prop_for_loss 2 hidden layer with sigmoid activation with regression Test Case	Sheet16
24	Forward_prop_for_loss 2 hidden layer with sigmoid activation with no regression Test Case	Sheet16
25	Forward_prop_for_loss 1 hidden layer with relu activation with regression Test Case	Sheet103
26	Forward_prop_for_loss 1 hidden layer with relu activation with no regression Test Case	Sheet103
27	Forward_prop_for_loss 1 hidden layer with sigmoid activation with no regression Test Case	Sheet104
28	Forward_prop_for_loss 1 hidden layer with sigmoid activation with no regression Test Case	Sheet104
29	Forward_prop_for_loss 0 hidden layer with regression Test Case	Sheet105
30	Forward_prop_for_loss 0 hidden layer with no regression Test Case	Sheet105
31	Accuracy with 2 hidden layer relu activation Test Case	Sheet17
32	Accuracy with 2 hidden layers sigmoid activation Test Case	Sheet18
33	Accuracy with 1 hidden layer relu activation Test Case	Sheet106
34	Accuracy with 1 hidden layer sigmoid activation Test Case	Sheet107
35	Accuracy with 0 hidden layer Test Case	Sheet108
36	Data Pre Processing Test case	-
37	Neural Network with 2 hidden layers with relu activation Test Case	Sheet19-24
38	Neural Network with 1 hidden layer with relu activation Test Case	Sheet 49-54
39	Neural Network with 0 hidden layer Test Case	Sheet 79-84
40	Neural Network with 2 hidden layers with sigmoid activation Test Case	Sheet25-30
41	Neural Network with 1 hidden layer with sigmoid activation Test Case	Sheet55-60
42	Neural Network with 2 hidden layers with relu activation with Momentum optimizer Test Case	Sheet31-36

43	Neural Network with 1 hidden layer with relu activation with Momentum optimizer Test Case	Sheet 61-66
44	Neural Network with 0 hidden layers with Momentum optimizer Test Case	Sheet85-90
45	Neural Network with 2 hidden layers with relu activation with decaying learning rate Test Case	Sheet 37-42
46	Neural Network with 1 hidden layer with relu activation with decaying learning rate Test Case	Sheet 67-72
47	Neural Network with 0 hidden layers with decaying learning rate Test Case	Sheet 91-96
48	Neural Network with 2 hidden layers with relu activation with L1 regression Test Case	Sheet43-48
49	Neural Network with 1 hidden layer with relu activation with L1 regression Test Case	Sheet73-78
50	Neural Network with zero hidden layers with L1 regression Test Case	Sheet 97-102

Structure of verify.py:

Verify.py file is the library of functions which get input data from the valid_excel.xlsx file which is feed to testing function (in train.py) and then the output of the function is compared to the output of calculations done in excel file (the calculations which the function in train.py file are supposed to-do as its functionality).

1. Relu Activation Test Case

   In Relu activation test case relu_activation () function is tested for its functionality in train.py file. In this test case input to the function is
   - the random value array of dimension [10 x 4] represent the output from the layer i.e. Z_array

   Value from valid_excel.xlsx is feed to relu_activation() function in train.py file and function output value

   - relu activated output array i.e. A_array of dim [10 X 4]

   Following output value is expected from the function which is then compared to the calculated array value in excel. If relu activated array matches with excel calculated array then function tested okay. (Excel Sheet1 is used for calculation)

2. Relu Activation back Test Case

   In Relu activation backtest case relu_activation_back() function is tested for its functionality in train.py file. In this test case input to the function is
   - 2 random value arrays of dimension [10 x 4] (represent the dloss_dA array and Z_array)

   Values from valid_excel.xlsx is feed to relu_activation_back() function in train.py file and function output values

   - Z_back array of dim [10 X 4]

   The following value is expected from the function which is then compared to the calculated array value in excel. If Z_back array matches with excel calculated array then function tested okay. (Excel Sheet2 is used for calculation)

3. Sigmoid Activation Test Case

   In Sigmoid activation test case sigmoid_activation() function is tested for its functionality in train.py file. In this test case, the input to the function is
   - the random value array of dimension [10 x 4]

Values from valid_excel.xlsx is feed to sigmoid_activation() and function in train.py file function output

- sigmoid activated output array of dim [10 X 4]

The following value is expected from the function which is then compared to the calculated array value in excel. If sigmoid activated array matches with excel calculated array then function tested okay. (Excel Sheet3 is used for calculation)

4. Sigmoid Activation back Test Case

In Sigmoid activation backtest case sigmoid_activation_back() function is tested for its functionality in train.py file. In this test case inputs to the function are

- 2 random value arrays of dimension [10 x 4] (represent the dloss_dA array and Z_array)

Values from valid_excel.xlsx is feed to sigmoid_activation_back() function in train.py file and function output

- Z_back array of dim [10 X 4]

Following output, array is expected from the function which is then compared to the calculated array value in excel. If Z_back array matches with excel calculated array then function tested okay. (Excel Sheet4 is used for calculation)

5. SGD optimizer for 2 hidden layers Test Case

In SGD optimizer for 2 hidden layer test case SGD_optimizer() function is tested for its functionality in train.py file. In this test case input to function is

- 3 random value array of dimension [10 x 10] (represent Weights for layer1, layer2 and layer 3)
- 3 random value array of dimension [10 x 1] (represent Biases for layer1, layer2 and layer 3)
- 3 random value array of dimension [10 x 10] (represent dloss_Weights for layer1, layer2 and layer 3)
- 3 random value array of dimension [10 x 1] (represent dloss_Biases for layer1, layer2 and layer 3)
- value for learning rate = 0.001
- num =2

Values from valid_excel.xlsx is feed to SGD_optimizer() function in train.py file and function output value

- SGD optimized weights for layer1 , layer2, layer3 of dimension [10 X 10]
- SGD optimized Biases for layer1 , layer2, layer3 of dimension [10 X 1]

Following output array are expected from the function which is then compared to the calculated array value in excel. If 6 output array [3 weights and 3 biases of each layer] matches with excel calculated arrays then function tested okay.(Excel Sheet5 is used for calculation)

6. SGD optimizer for 1 hidden layers Test Case

In SGD optimizer for 1 hidden layer test case SGD_optimizer() function is tested for its functionality in train.py file. In this test case input to the function

- 2 random value array of dimension [10 x 10] (represent Weights for layer1 and layer2)
- 2 random value array of dimension [10 x 1] (represent Biases for layer1 and layer2)
- 2 random value array of dimension [10 x 10] (represent dloss_Weights for layer1 and layer2)
- 2 random value array of dimension [10 x 1] (represent dloss_Biases for layer1 and layer2)
- value for learning rate = 0.001
- num = 1

Values from valid_excel.xlsx is feed to SGD_optimizer() function in train.py file and function output values

- SGD optimized weights for layer1 and layer2 of dimension [10 X 10]
- SGD optimized Biases for layer1 and layer2 of dimension [10 X 1]

Following output arrays are expected from the function which is then compared to the calculated array values in excel. If 4 output array [2 weights and 2 biases of each layer] matches with excel calculated arrays then function tested okay. (Excel Sheet5 is used for calculation)

7. SGD optimizer for zero hidden layers Test Case

In SGD optimizer for 1 hidden layer test case SGD_optimizer() function is tested for its functionality in train.py file. In this test case input to the function is

- 1 random value array of dimension [10 x 10] (represent Weights for layer1)
- 1 random value array of dimension [10 x 1] (represent Biases for layer1)
- 1 random value array of dimension [10 x 10] (represent dloss_Weights for layer1)
- 1 random value array of dimension [10 x 1] (represent dloss_Biases for layer1)
- value for learning rate = 0.001
- num =0

Values from valid_excel.xlsx is feed to SGD_optimizer() function in train.py file and function output values

- SGD optimized weights for layer1 of dimension [10 X 10]

- SGD optimized Biases for layer1 of dimension [10 X 1]

Following output arrays are expected from the function which is then compared to the calculated array value in excel. If 2 output array[1 weights and 1 biases] matches with excel calculated arrays then function tested okay.(Excel Sheet5 is used for calculation)

8. Momentum optimizer for 2 hidden layers Test Case

In Momentum optimizer for 2 hidden layer test case Momentum_optimizer()
function is tested for its functionality in train.py file. In this test case input array to the function is

- 3 random value array of dimension [10 x 10] (represent Weights for layer1, layer2 and layer 3)
- 3 random value array of dimension [10 x 1] (represent Biases for layer1, layer2 and layer 3) ,
- 3 random value array of dimension [10 x 10] (represent mov_Weights for layer1, layer2 and layer 3) ,
- 3 random value array of dimension [10 x 1] (represent mov_Biases for layer1, layer2 and layer 3)
- 3 random value array of dimension [10 x 10] (represent dloss_Weights for layer1, layer2 and layer 3)
- 3 random value array of dimension [10 x 1] (represent dloss_Biases for layer1, layer2 and layer 3)
- value for beta = 0.9
- value for learning rate = 0.001
- num =2

Values from valid_excel.xlsx is feed to Momentum_optimizer() function in train.py file and function output values

- Momentum optimized weights for layer1 , layer2, layer3 of dimension [10 X 10]
- Momentum optimized Biases for layer1 , layer2, layer3 of dimension [10 X 1]
- Momentum optimized mov_weights for layer1 , layer2, layer3 of dimension [10 X 10],
- Momentum optimized mov_Biases for layer1 , layer2, layer3 of dimension [10 X 1]

Following output arrays are expected from the function which is then compared to the calculated array value in excel. If all 12 output array [3 weights ,3 biases, 3 mov_weights and 3 mov_ biases of each layer] matches with excel calculated arrays then function tested okay.(Excel Sheet6 is used for calculation)

9. Momentum optimizer for 1 hidden layers Test Case

In Momentum optimizer for 1 hidden layer test case Momentum_optimizer() function is tested for its functionality in train.py file. In this test case input to the function is

- 2 random value array of dimension [10 x 10] (represent Weights for layer1 and layer2)
- 2 random value array of dimension [10 x 1] (represent Biases for layer1 and layer2)
- 2 random value array of dimension [10 x 10] (represent mov_Weights for layer1 and layer2)
- 2 random value array of dimension [10 x 1] (represent mov_Biases for layer1 and layer2)
- 2 random value array of dimension [10 x 10] (represent dloss_Weights for layer1 and layer2)
- 2 random value array of dimension [10 x 1] (represent dloss_Biases for layer1 and layer2)
- value for beta = 0.9
- value for learning rate = 0.001
- num = 1

Values from valid_excel.xlsx is feed to Momentum_optimizer() function in train.py file and function output values

- Momentum optimized weights for layer1 and layer2 of dimension [10 X 10]
- Momentum optimized Biases for layer1 and layer2 of dimension [10 X 1]
- Momentum optimized mov_weights for layer1 and layer2 of dimension [10 X 10]
- Momentum optimized mov_Biases for layer1 and layer2 of dimension [10 X 1]

Following arrays are expected from the function which is then compared to the calculated array value in excel. If all 8 output array [2 weights ,2 biases, 2 mov_weights and 2 mov_ biases of each layer] matches with excel calculated arrays then function tested okay.(Excel Sheet6 is used for calculation)

10. Momentum optimizer for zero hidden layers Test Case

In Momentum optimizer for zero hidden layer test case Momentum_optimizer() function is tested for its functionality in train.py file. In this test case input to the function are

- 1 random value array of dimension [10 x 10] (represent Weights for layer1)
- 1 random value array of dimension [10 x 1] (represent Biases for layer1)
- 1 random value array of dimension [10 x 10] (represent mov_Weights for layer1)
- 1 random value array of dimension [10 x 1] (represent mov_Biases for layer1)
- 1 random value array of dimension [10 x 10] (represent dloss_Weights for layer1)
- 1 random value array of dimension [10 x 1] (represent dloss_Biases for layer1)
- value for beta = 0.9
- value for learning rate 0.001

- num = 0

Values from valid_excel.xlsx is feed to Momentum_optimizer() function in train.py file and function output values

- Momentum optimized weights for layer1 of dimension [10 X 10]
- Momentum optimized Biases for layer1 of dimension [10 X 1]
- Momentum optimized mov_weights for layer1 of dimension [10 X 10]
- Momentum optimized mov_Biases for layer1 of dimension [10 X 1]

Following arrays are expected from the function which is then compared to the calculated array value in excel. If all 4 output array [1 weights ,1 biases, 1 mov_weights and 1 mov_ biases of each layer] matches with excel calculated arrays then function tested okay.(Excel Sheet6 is used for calculation)

11. Regression Loss for 2 hidden layer Test Case

In Regression loss for 2 hidden layer test case reg_loss() function is tested for its functionality in train.py file. In this test case inputs to the function are

- random value array of dimension [10 x 10] (represent Weights for layer1, layer2 and layer 3)
- value of alpha = 0.00001
- num =2

Values from valid_excel.xlsx are feed to reg_loss() function in train.py file and output the values

- single digit regression loss

The following value is expected from the function which is then compared to the calculated regression loss value in excel. If regression loss value matches with excel calculated regression value then function tested okay. (Excel Sheet7 is used for calculation)

12. Regression Loss for 1 hidden layer Test Case

In Regression loss for 1 hidden layer test case reg_loss() function is tested for its functionality in train.py file. In this test case inputs to the function are

- the random value array of dimension [10 x 10] (represent Weights for layer1 and layer2)
- value of alpha = 0.00001
- num = 1

Values from valid_excel.xlsx are feed to reg_loss() function in train.py file and output values

- Single-digit regression loss

The following value is expected from the function which is then compared to the calculated regression loss value in excel. If regression loss value matches with excel calculated regression value then function tested okay. (Excel Sheet7 is used for calculation)

13. Regression Loss for zero hidden layer Test Case

In Regression loss for zero hidden layer test case reg_loss() function is tested for its functionality in train.py file. In this test case inputs to the functions are

- random value array of dimension [10 x 10] (represent Weights for layer1)
- value of alpha = 0.00001
- num = 0

Values from valid_excel.xlsx are feed to reg_loss() function in train.py file and output the values

- single digit regression loss

The following value is expected from the function which is then compared to the calculated regression loss value in excel. If regression loss value matches with excel calculated regression value then function tested okay. (Excel Sheet7 is used for calculation)

14. Learning rate decay Test Case

In Learning rate decay test case learning_rate_decay() function is tested for its functionality in train.py file. In this case input to the function

- learning rate value = 0.001
- decay rate = 0.98

Values from valid_excel.xlsx is feed to learning_rate_decay() function in train.py file and function output values

- decayed learning rate in each step

Following output value is expected from the function which is then compared to the calculated decayed learning rate value in excel. If decay learning rate value matches with excel calculated regression value then function tested okay. (Excel Sheet8 is used for calculation)

15. Normal normalization Test Case

In Normal Normalized test case Normal_normalization() function is tested for its functionality in train.py file. In this case, input to the function is

- random image array of dim [10 x10]

Values from valid_excel.xlsx is feed to Normal_normalization() function in train.py file and function output value

- Normalized image array

Following output, the array is expected from the function which is then compared to the calculated normalized image array in excel. If normalized image array matches with excel calculated normalized image array then function tested okay. (Excel Sheet9 is used for calculation)

16. Simple normalization Test Case

In Simple Normalized test case Simple_normalization() function is tested for its functionality in train.py file. In this case, the input to the function is

- random image array of dim [10 x10]

Values from valid_excel.xlsx is feed to Simple_normalization() function in train.py file and function output values

- Normalized image array

Following output, an array is expected from the function which is then compared to the calculated normalized image array in excel. If normalized image array matches with excel calculated normalized image array then function tested okay. (Excel Sheet10 is used for calculation)

17. Loss function with regression 2 hidden layers Test Case

In Loss function with regression 2 hidden layers test case Loss_function() function is tested for its functionality in train.py file. In this case, input to the function is

- random pred array of dim [10 x4]
- predefined true label array of dimension [10x4]
- 3 random array of weights from layer 1, layer2, layer3 of dimension [10 X10] each
- value of alpha = 0.0001
- num = 2
- reg = “y”

Values from valid_excel.xlsx is feed to Loss_function() function in train.py file and function output value

- single loss value

Following output value is expected from the function which is then compared to the calculated loss value in excel. If loss value matches with excel calculated loss value then function tested okay. (Excel Sheet14 is used for calculation)

18. Loss function with regression 1 hidden layers Test Case

In Loss function with regression 1 hidden layers test case Loss_function() function is tested for its functionality in train.py file. In this case, input to the function is

- random pred array of dim [10 x4]
- predefined true label array of dimension [10x4]
- 2 random array of weights from layer 1 and layer2 of dimension [10 X10] each
- value of alpha = 0.00001
- reg = “y”
- num = 1

Values from valid_excel.xlsx is feed to Loss_function() function in train.py file and function output values

- single loss value

The following output is expected from the function which is then compared to the calculated loss value in excel. If loss value matches with excel calculated loss value then function tested okay. (Excel Sheet14 is used for calculation)

19. Loss function with regression 0 hidden layers Test Case

In Loss function with regression zero hidden layers test case Loss_function() function is tested for its functionality in train.py file. In this case input to the function

- random pred array of dim [10 x4]
- predefined true label array of dimension [10x4]
- 1 random array of weights from layer 1 of dimension [10 X10]
- value of alpha = 0.0001
- reg = “y”
- num = 0

Values from valid_excel.xlsx is feed to Loss_function() function in train.py file and function output

- single loss value

The following output is expected from the function which is then compared to the calculated loss value in excel. If loss value matches with excel calculated loss value then function tested okay. (Excel Sheet14 is used for calculation)

20. Loss function with no regression Test Case

In Loss function with no regression test case Loss_function() function is tested for its functionality in train.py file. In this case inputs to the function

- random pred array of dim [10 x4]
- predefined true label array of dimension [10x4]

Values from valid_excel.xlsx is feed to Loss_function() function in train.py file and function output

- single loss value

The following output is expected from the function which is then compared to the calculated loss value in excel. If loss value matches with excel calculated loss value then function tested okay. (Excel Sheet14 is used for calculation).

21. Forward prop for loss with Relu activation with regression for 2 hidden layer Test Case

In Forward prop for loss with Relu activation with regression for 2 hidden layer Test Case forward_prop_for_loss() function is tested for its functionality in train.py file. In this case input to the function

- random image array of dim [10 x4]
- pre -define true label array of dimension [10x4],
- 3 random array of dimension [10x10] representing weights in layer 1, layer2, layer 3,
- 3 random array of dimension [10x1] representing biases in layer 1, layer2, layer 3,
- activation function value = "relu",
- reg value = "y"
- value of alpha = 0.00001
- num = 2

Values from valid_excel.xlsx is feed to Loss_function() function in train.py file and function output

- single loss value

Following output is expected from function which is then compared to the calculated loss value in excel. If loss value matches with excel calculated loss value then function tested okay. (Excel Sheet15 is used for calculation).

22. Forward prop for loss with Relu activation with no regression for 2 hidden layer Test Case

In Forwarding prop for loss with Relu activation with no regression for 2 hidden layer Test Case forward_prop_for_loss() function is tested for its functionality in train.py file. In this case inputs to the function are

- random image array of dim [10 x4] and pre-defined true label array of dimension [10x4]
- 3 random array of dimension [10x10] representing weights in layer 1, layer2, layer 3
- 3 random array of dimension [10x1] representing biases in layer 1, layer2, layer 3
- activation function value = "relu"
- reg value = "n"
- a value of alpha = 0.00001
- num =2

Values from valid_excel.xlsx is feed to Loss_function() function in train.py file and function output value

- single loss value

The following output is expected from the function which is then compared to the calculated loss value in excel. If loss value matches with excel calculated loss value then function tested okay. (Excel Sheet15 is used for calculation).

23. Forward prop for loss with sigmoid activation with regression for 2 hidden layer Test Case

In Forward prop for loss with sigmoid activation with regression for 2 hidden layer Test Case forward_prop_for_loss() function is tested for its functionality in train.py file. In this case inputs to the function are

- random image array of dim [10 x4]
- predefined true label array of dimension [10x4]
- 3 random array of dimension [10x10] representing weights in layer 1, layer2, layer 3
- 3 random array of dimension [10x1] representing biases in layer 1, layer2, layer 3,
- activation function value = "sigmoid"
- reg value = "y"
- a value of alpha = 0.00001
- num = 2

Value from valid_excel.xlsx is feed to Loss_function() function in train.py file and function output values

- single loss value

The following output is expected from the function which is then compared to the calculated loss value in excel. If loss value matches with excel calculated loss value then function tested okay. (Excel Sheet16 is used for calculation).

24. Forward prop for loss with sigmoid activation with no regression for 2 hidden layer Test Case

In Forward prop for loss with sigmoid activation with no regression for 2 hidden layer Test Case forward_prop_for_loss() function is tested for its functionality in train.py file. In this case inputs to the function are

- random image array of dim [10 x4]
- pre -define true label array of dimension [10x4]
- 3 random array of dimension [10x10] representing weights in layer 1, layer2, layer 3
- 3 random array of dimension [10x1] representing biases in layer 1, layer2, layer3
- activation function value = "sigmoid"
- reg value = "n"
- a value of alpha = 0.00001
- num = 2

Values from valid_excel.xlsx is feed to Loss_function() function in train.py file and function output values

- single loss value

Following output is expected from function which is then compared to the calculated loss value in excel. If loss value matches with excel calculated loss value then function tested okay. (Excel Sheet16 is used for calculation).

25. Forward prop for loss with Relu activation with regression for 1 hidden layer Test Case

In Forward prop for loss with Relu activation with regression for 1 hidden layer Test Case forward_prop_for_loss() function is tested for its functionality in train.py file. In this case inputs to the function are

- random image array of dim [10 x4]
- pre -define true label array of dimension [10x4],
- 2 random array of dimension [10x10] representing weights in layer 1 and layer2,
- 2 random array of dimension [10x1] representing biases in layer 1 and layer2,
- activation function value = "relu"
- reg value = "y"
- a value of alpha = 0.00001
- num = 1

Values from valid_excel.xlsx is feed to Loss_function() function in train.py file and function output values

- single loss value

Following value is expected from function which is then compared to the calculated loss value in excel. If loss value matches with excel calculated loss value then function tested okay. (Excel Sheet103 is used for calculation).

26. Forward prop for loss with Relu activation with no regression for 1 hidden layer Test Case

In Forward prop for loss with Relu activation with no regression for 1 hidden layer Test Case forward_prop_for_loss() function is tested for its functionality in train.py file. In this case inputs to the function are

- random image array of dim [10 x4]
- pre -define true label array of dimension [10x4],
- 2 random array of dimension [10x10] representing weights in layer 1 and layer2,
- 2 random array of dimension [10x1] representing biases in layer 1 and layer2,
- activation function value = "relu"
- reg value = "n"
- a value of alpha = 0.00001
- num = 1

Values from valid_excel.xlsx is feed to Loss_function() function in train.py file and function output values

- single loss value

Following value is expected from function which is then compared to the calculated loss value in excel. If loss value matches with excel calculated loss value then function tested okay. (Excel Sheet103 is used for calculation).

27. Forward prop for loss with sigmoid activation with regression for 1 hidden layer Test Case

In Forward prop for loss with sigmoid activation with regression for 1 hidden layer Test Case forward_prop_for_loss() function is tested for its functionality in train.py file. In this case inputs to the function are

- random image array of dim [10 x4]
- pre -define true label array of dimension [10x4],
- 2 random array of dimension [10x10] representing weights in layer 1 and layer2,
- 2 random array of dimension [10x1] representing biases in layer 1 and layer2,
- activation function value = "sigmoid"
- reg value = "y"
- a value of alpha = 0.00001
- num = 1

Values from valid_excel.xlsx is feed to Loss_function() function in train.py file and function output values

- single loss value

Following value is expected from function which is then compared to the calculated loss value in excel. If loss value matches with excel calculated loss value then function tested okay. (Excel Sheet104 is used for calculation).

28. Forward prop for loss with sigmoid activation with no regression for 1 hidden layer Test Case

    In Forward prop for loss with sigmoid activation with no regression for 1 hidden layer Test Case forward_prop_for_loss() function is tested for its functionality in train.py file. In this case inputs to the function are
    - random image array of dim [10 x4]
    - pre -define true label array of dimension [10x4],
    - 2 random array of dimension [10x10] representing weights in layer 1 and layer2
    - 2 random array of dimension [10x1] representing biases in layer 1 and layer2
    - activation function value = "sigmoid"
    - reg value = "n"
    - a value of alpha = 0.00001
    - num = 1

    Values from valid_excel.xlsx is feed to Loss_function() function in train.py file and function output values

    - single loss value

    Following value is expected from function which is then compared to the calculated loss value in excel. If loss value matches with excel calculated loss value then function tested okay. (Excel Sheet104 is used for calculation).

29. Forward prop for loss with regression for zero hidden layer Test Case

    In Forward prop for loss with regression for zero hidden layer Test Case forward_prop_for_loss() function is tested for its functionality in train.py file. In this case inputs to the function are
    - random image array of dim [10 x4]
    - pre -define true label array of dimension [10x4],
    - 1 random array of dimension [10x10] representing weights in layer 1
    - 1 random array of dimension [10x1] representing biases in layer 1
    - activation function value = "n"
    - reg value = "y"
    - a value of alpha = 0.00001
    - num = 1

Values from valid_excel.xlsx is feed to Loss_function() function in train.py file and function output values

- single loss value

Following value is expected from function which is then compared to the calculated loss value in excel. If loss value matches with excel calculated loss value then function tested okay. (Excel Sheet105 is used for calculation).

30. Forward prop for loss with no regression for zero hidden layer Test Case

In Forward prop for loss no with regression for zero hidden layer Test Case forward_prop_for_loss() function is tested for its functionality in train.py file. In this case inputs to the function are

- random image array of dim [10 x4]
- pre -define true label array of dimension [10x4],
- 1 random array of dimension [10x10] representing weights in layer 1
- 1 random array of dimension [10x1] representing biases in layer 1
- activation function value = "n"
- reg value = "n"
- a value of alpha = 0.00001
- num = 1

Values from valid_excel.xlsx is feed to Loss_function() function in train.py file and function output values

- single loss value

Following value is expected from function which is then compared to the calculated loss value in excel. If loss value matches with excel calculated loss value then function tested okay. (Excel Sheet105 is used for calculation).

31. Accuracy with relu activation 2 hidden layer Test Case

Accuracy with relu activation 2 hidden layer Test Case accuracy () function is tested for its functionality in train.py file. In this case, input to function are

- random image array of dim [10 x4]
- predefined true label array of dimension [10x4]
- 3 random array of dimension [10x10] representing weights in layer 1, layer2, layer 3
- 3 random array of dimension [10x1] representing biases in layer 1, layer2, layer 3
- Activation function value = "relu"
- num = 2

Values from valid_excel.xlsx is feed to Loss_function() function in train.py file and function output value

- single accuracy value

Following output value is expected from the function which is then compared to the calculated accuracy value in excel. If accuracy value matches with excel calculated accuracy value then function tested okay. (Excel Sheet17 is used for calculation).

32. Accuracy with sigmoid activation 2 hidden layer Test Case

Accuracy with sigmoid activation 2 hidden layer Test Case accuracy () function is tested for its functionality in train.py file. In this case, inputs to function are

- random image array of dim [10 x4]
- predefined true label array of dimension [10x4]
- 3 random array of dimension [10x10] representing weights in layer 1, layer2, layer 3
- 3 random array of dimension [10x1] representing biases in layer 1, layer2, layer 3
- Activation function value = "sigmoid"
- num = 2

Values from valid_excel.xlsx is feed to Loss_function() function in train.py file and function output value

- single accuracy value

Following output value is expected from the function which is then compared to the calculated accuracy value in excel. If accuracy value matches with excel calculated accuracy value then function tested okay. (Excel Sheet18 is used for calculation).

33. Accuracy with relu activation 1 hidden layer Test Case

Accuracy with relu activation 1 hidden layer Test Case accuracy () function is tested for its functionality in train.py file. In this case inputs to the function are

- random image array of dim [10 x4]
- predefined true label array of dimension [10x4]
- 2 random array of dimension [10x10] representing weights in layer 1 and layer2
- 2 random array of dimension [10x1] representing biases in layer 1 and layer2
- Activation function value = "relu"
- num = 2

Values from valid_excel.xlsx is feed to Loss_function() function in train.py file and function output value

- single accuracy value

Following output value is expected from the function which is then compared to the calculated accuracy value in excel. If accuracy value matches with excel calculated accuracy value then function tested okay. (Excel Sheet105 is used for calculation).

34. Accuracy with sigmoid activation 1 hidden layer Test Case

Accuracy with sigmoid activation 1 hidden layer Test Case accuracy () function is tested for its functionality in train.py file. In this case inputs to the function are

- random image array of dim [10 x4]
- predefined true label array of dimension [10x4]
- 2 random array of dimension [10x10] representing weights in layer 1 and layer2
- 2 random array of dimension [10x1] representing biases in layer 1 and layer2
- Activation function value = "sigmoid"
- num = 2

Values from valid_excel.xlsx is feed to Loss_function() function in train.py file and function output value

- single accuracy value

Following output value is expected from the function which is then compared to the calculated accuracy value in excel. If accuracy value matches with excel calculated accuracy value then function tested okay. (Excel Sheet106 is used for calculation).

35. Accuracy with 0 hidden layer Test Case

Accuracy with zero hidden layer Test Case accuracy () function is tested for its functionality in train.py file. In this case inputs to the function are

- random image array of dim [10 x4]
- predefined true label array of dimension [10x4]
- 1 random array of dimension [10x10] representing weights in layer 1
- 1 random array of dimension [10x1] representing biases in layer 1
- Activation function value = "n"
- num = 2

Values from valid_excel.xlsx is feed to Loss_function() function in train.py file and function output value

- single accuracy value

Following output value is expected from the function which is then compared to the calculated accuracy value in excel. If accuracy value matches with excel calculated accuracy value then function tested okay. (Excel Sheet107 is used for calculation).

36. Data pre-processing Test Case

In Data Preprocessing Test Case Data pre-processing () function is tested for its functionality in train.py file. The approach used to test the working of the function is manual testing in this approach the Data pre-processing () called with the input of normal= "simple" and the testing function select the random 10 digit image for display and convert the hot label of randomly selected images as window title of image display. When test function is executed 10 randomly selected images are displayed for manual testing with their label on window name.

37. Neural Network with 2 hidden layers with relu activation Test Case

In Neural Network with 2 hidden layer with relu activation test case NN_2_layer_test () function is tested for its functionality in train.py file. NN_2_layer_test () function is reduced down version of NN_2_layer () function in train.py. NN_2_layer_test () function train neural network with 10 input neuron, 10 neuron in 1st hidden layer, 10 neuron in 2nd hidden layer and 10 neuron in output layer.
In this case, input to a function is

- random image array of dim [10 x4]
- pre -define true label array of dimension [10x4]
- 3 random array of dimension [10x10] representing weights in layer 1, layer2, layer 3
- 3 random array of dimension [10x1] representing biases in layer 1, layer2, layer 3
- activation function value = “relu”
- L1 reg value = “n”
- Decay = “n”
- value of beta = 0.9
- batch = 4
- decay_rate =0.98
- number of hidden neuron in layer is 10 each
- optimizer = “sgd”
- a value of alpha = 0.00001
- number epoch to train = 5
- num= 2

Output to the function in this case is

- 3 trained array of dimension [10x10] representing trained weights of layer 1, layer2, layer 3 for network after 5 epochs.
- 3 trained array of dimension [10x1] representing trained biases in layer 1, layer2, layer 3 for network after 5 epoch.
- Loss value of the neural network
- Accuracy value of the neural network.

Input values are feed to NN_2_layers_test () function from valid_excel.xlsx file. The output value of the function is then compared to the calculated value of arrays in excel. If function output values match with excel calculated value then function tested okay otherwise not okay. (Excel Sheet 19-24 is used for calculation).

38. Neural Network with 1 hidden layer with relu activation Test Case

In Neural Network with 1 hidden layer with relu activation test case NN_1_layer_test () function is tested for its functionality in train.py file. NN_1_layer_test () function is reduced down version of NN_1_layer () function in train.py. NN_1_layer_test () function train neural network with 10 input neuron, 10 neuron in hidden layer and 10 neuron in output layer.
In this case, input to a function is

- random image array of dim [10 x4]
- pre -define true label array of dimension [10x4]
- 2 random array of dimension [10x10] representing weights in layer 1 and layer2
- 2 random array of dimension [10x1] representing biases in layer 1 and layer2
- activation function value = “relu”
- L1 reg value = “n”
- Decay = “n”
- value of beta = 0.9
- batch = 4
- decay_rate =0.98
- number of hidden neuron in layer is 10 each
- optimizer = “sgd”
- a value of alpha = 0.00001
- number epoch to train = 5
- num= 1

Output to the function in this case is

- 2 trained array of dimension [10x10] representing trained weights of layer 1 and layer2 for network after 5 epochs.
- 2 trained array of dimension [10x1] representing trained biases in layer 1 and layer2 for network after 5 epochs.
- Loss value of the neural network
- Accuracy value of the neural network.

Input values are feed to NN_1_layers_test () function from valid_excel.xlsx file. The output value of the function is then compared to the calculated value of arrays in excel. If function output values match with excel calculated value then function tested okay otherwise not okay. (Excel Sheet 49-54 is used for calculation).

39. Neural Network with zero hidden layer Test Case

In Neural Network with no hidden layer test case NN_0_layer_test () function is tested for its functionality in train.py file. NN_0_layer_test () function is reduced down version of NN_0_layer () function in train.py. NN_0_layer_test () function train neural network with 10 input neuron and 10 neuron in output layer.
In this case, input to a function is

- random image array of dim [10 x4]
- pre -define true label array of dimension [10x4]
- 1 random array of dimension [10x10] representing weights in layer 1
- 1 random array of dimension [10x1] representing biases in layer 1
- activation function value = “n”
- L1 reg value = “n”
- Decay = “n”
- value of beta = 0.9
- batch = 4
- decay_rate =0.98
- number of hidden neuron in layer is 10 each
- optimizer = “sgd”
- a value of alpha = 0.00001
- number epoch to train = 5
- num= 0

Output to the function in this case is

- 1 trained array of dimension [10x10] representing trained weights of layer 1 for network after 5 epochs.
- 1 trained array of dimension [10x1] representing trained biases in layer 1 for network after 5 epochs.
- Loss value of the neural network
- Accuracy value of the neural network.

Input values are feed to NN_0_layers_test () function from valid_excel.xlsx file. The output value of the function is then compared to the calculated value of arrays in excel. If function output values match with excel calculated value then function tested okay otherwise not okay. (Excel Sheet 79-84 is used for calculation).

40. Neural Network with 2 hidden layers with sigmoid activation Test Case

In Neural Network with 2 hidden layer with sigmoid activation test case NN_2_layer_test () function is tested for its functionality in train.py file. NN_2_layer_test () function is reduced down version of NN_2_layer () function in train.py. NN_2_layer_test () function train neural network with 10 input neuron, 10 neuron in 1st hidden layer, 10 neuron in 2nd hidden layer and 10 neuron in output layer.

In this case, input to a function is

- random image array of dim [10 x4]
- pre -define true label array of dimension [10x4]
- 3 random array of dimension [10x10] representing weights in layer 1, layer2, layer 3
- 3 random array of dimension [10x1] representing biases in layer 1, layer2, layer 3
- activation function value = “sigmoid”
- L1 reg value = “n”
- Decay = “n”
- value of beta = 0.9
- batch = 4
- decay_rate =0.98
- number of hidden neuron in layer is 10 each
- optimizer = “sgd”
- a value of alpha = 0.00001
- number epoch to train = 5
- num= 2

Output to the function in this case is

- 3 trained array of dimension [10x10] representing trained weights of layer 1, layer2, layer 3 for network after 5 epochs.
- 3 trained array of dimension [10x1] representing trained biases in layer 1, layer2, layer 3 for network after 5 epochs.
- Loss value of the neural network
- Accuracy value of the neural network.

Input values are feed to NN_2_layers_test () function from valid_excel.xlsx file. The output value of the function is then compared to the calculated value of arrays in excel. If function output values match with excel calculated value then function tested okay otherwise not okay. (Excel Sheet 25-30 is used for calculation).

41. Neural Network with 1 hidden layer with sigmoid activation Test Case

In Neural Network with 1 hidden layer with sigmoid activation test case NN_1_layer_test () function is tested for its functionality in train.py file. NN_1_layer_test () function is reduced down version of NN_1_layer () function in train.py. NN_1_layer_test () function train neural network with 10 input neuron, 10 neuron in hidden layer and 10 neuron in output layer.
In this case, input to a function is

- random image array of dim [10 x4]
- pre -define true label array of dimension [10x4]

- 2 random array of dimension [10x10] representing weights in layer 1 and layer2
- 2 random array of dimension [10x1] representing biases in layer 1 and layer2
- activation function value = “sigmoid”
- L1 reg value = “n”
- Decay = “n”
- value of beta = 0.9
- batch = 4
- decay_rate =0.98
- number of hidden neuron in layer is 10 each
- optimizer = “sgd”
- a value of alpha = 0.00001
- number epoch to train = 5
- num= 1

Output to the function in this case is

- 2 trained array of dimension [10x10] representing trained weights of layer 1 and layer2 for network after 5 epochs.
- 2 trained array of dimension [10x1] representing trained biases in layer 1 and layer2 for network after 5 epochs.
- Loss value of the neural network
- Accuracy value of the neural network.

Input values are feed to NN_1_layers_test () function from valid_excel.xlsx file. The output value of the function is then compared to the calculated value of arrays in excel. If function output values match with excel calculated value then function tested okay otherwise not okay. (Excel Sheet 55-60 is used for calculation).

42. Neural Network with 2 hidden layers with relu activation with Momentum optimizer Test Case

In Neural Network with 2 hidden layer with relu activation with momentum optimizer test case NN_2_layer_test () function is tested for its functionality in train.py file. NN_2_layer_test () function is reduced down version of NN_2_layer () function in train.py. NN_2_layer_test () function train neural network with 10 input neuron, 10 neuron in 1st hidden layer, 10 neuron in 2nd hidden layer and 10 neuron in output layer.
In this case, input to a function is

- random image array of dim [10 x4]
- pre -define true label array of dimension [10x4]
- 3 random array of dimension [10x10] representing weights in layer 1, layer2, layer 3
- 3 random array of dimension [10x1] representing biases in layer 1, layer2, layer 3
- 3 zero array of dimension [10x10] representing mov_weights in layer 1, layer2, layer 3

- 3 zero array of dimension [10x1] representing mov_biases in layer 1, layer2, layer 3
- activation function value = “relu”
- L1 reg value = “n”
- Decay = “n”
- value of beta = 0.9
- batch = 4
- decay_rate =0.98
- number of hidden neuron in layer is 10 each
- optimizer = “Momentum”
- a value of alpha = 0.00001
- number epoch to train = 5
- num= 2

Output to the function in this case is

- 3 trained array of dimension [10x10] representing trained weights of layer 1, layer2, layer 3 for network after 5 epochs.
- 3 trained array of dimension [10x1] representing trained biases in layer 1, layer2, layer 3 for network after 5 epochs.
- Loss value of the neural network
- Accuracy value of the neural network.

Input values are feed to NN_2_layers_test () function from valid_excel.xlsx file. The output value of the function is then compared to the calculated value of arrays in excel. If function output values match with excel calculated value then function tested okay otherwise not okay. (Excel Sheet 31-36 is used for calculation).

43. Neural Network with 1 hidden layer with relu activation with Momentum optimizer Test Case

In Neural Network with 1 hidden layer with relu activation momentum optimizer test case NN_1_layer_test () function is tested for its functionality in train.py file. NN_1_layer_test () function is reduced down version of NN_1_layer () function in train.py. NN_1_layer_test () function train neural network with 10 input neuron, 10 neuron in hidden layer and 10 neuron in output layer.
In this case, input to a function is

- random image array of dim [10 x4]
- pre -define true label array of dimension [10x4]
- 2 random array of dimension [10x10] representing weights in layer 1 and layer2
- 2 random array of dimension [10x1] representing biases in layer 1 and layer2
- 2 zero array of dimension [10x10] representing mov_weights in layer 1 and layer2
- 2 zero array of dimension [10x1] representing mov_biases in layer 1 and layer2
- activation function value = “relu”

- L1 reg value = “n”
- Decay = “n”
- value of beta = 0.9
- batch = 4
- decay rate =0.98
- number of hidden neuron in layer is 10 each
- optimizer = “momentum”
- a value of alpha = 0.00001
- number epoch to train = 5
- num= 1

Output to the function in this case is

- 2 trained array of dimension [10x10] representing trained weights of layer 1 and layer2 for network after 5 epochs.
- 2 trained array of dimension [10x1] representing trained biases in layer 1 and layer2 for network after 5 epochs.
- Loss value of the neural network
- Accuracy value of the neural network.

Input values are feed to NN_1_layers_test () function from valid_excel.xlsx file. The output value of the function is then compared to the calculated value of arrays in excel. If function output values match with excel calculated value then function tested okay otherwise not okay. (Excel Sheet 61--66 is used for calculation).

44. Neural Network with zero hidden layers with momentum optimizer Test Case

In Neural Network with no hidden layer with momentum optimizer test case NN_0_layer_test () function is tested for its functionality in train.py file. NN_0_layer_test () function is reduced down version of NN_0_layer () function in train.py. NN_0_layer_test () function train neural network with 10 input neuron and 10 neuron in output layer.
In this case, input to a function is

- random image array of dim [10 x4]
- pre -define true label array of dimension [10x4]
- 1 random array of dimension [10x10] representing weights in layer 1
- 1 random array of dimension [10x1] representing biases in layer 1
- 1 zero array of dimension [10x10] representing mov_weights in layer 1
- 1 zero array of dimension [10x1] representing mov_biases in layer 1
- activation function value = “n”
- L1 reg value = “n”
- Decay = “n”
- value of beta = 0.9

- batch = 4
- decay_rate =0.98
- number of hidden neuron in layer is 10 each
- optimizer = “momentum”
- a value of alpha = 0.00001
- number epoch to train = 5
- num= 0

Output to the function in this case is

- 1 trained array of dimension [10x10] representing trained weights of layer 1 for network after 5 epochs.
- 1 trained array of dimension [10x1] representing trained biases in layer 1 for network after 5 epochs.
- Loss value of the neural network
- Accuracy value of the neural network.

Input values are feed to NN_0_layers_test () function from valid_excel.xlsx file. The output value of the function is then compared to the calculated value of arrays in excel. If function output values match with excel calculated value then function tested okay otherwise not okay. (Excel Sheet 85-90 is used for calculation).

45. Neural Network with 2 hidden layers with relu activation with decaying learning rate Test Case

In Neural Network with 2 hidden layer with relu activation with decaying learning rate test case NN_2_layer_test () function is tested for its functionality in train.py file. NN_2_layer_test () function is reduced down version of NN_2_layer () function in train.py. NN_2_layer_test () function train neural network with 10 input neuron, 10 neuron in 1st hidden layer, 10 neuron in 2nd hidden layer and 10 neuron in output layer.
In this case, input to a function is

- random image array of dim [10 x4]
- pre -define true label array of dimension [10x4]
- 3 random array of dimension [10x10] representing weights in layer 1, layer2, layer 3
- 3 random array of dimension [10x1] representing biases in layer 1, layer2, layer 3
- activation function value = “relu”
- L1 reg value = “n”
- Decay = “y”
- value of beta = 0.9
- batch = 4
- decay_rate =0.98
- number of hidden neuron in layer is 10 each
- optimizer = “sgd”

- a value of alpha = 0.00001
- number epoch to train = 5
- num= 2

Output to the function in this case is

- 3 trained array of dimension [10x10] representing trained weights of layer 1, layer2, layer 3 for network after 5 epochs.
- 3 trained array of dimension [10x1] representing trained biases in layer 1, layer2, layer 3 for network after 5 epochs.
- Loss value of the neural network
- Accuracy value of the neural network.

Input values are feed to NN_2_layers_test () function from valid_excel.xlsx file. The output value of the function is then compared to the calculated value of arrays in excel. If function output values match with excel calculated value then function tested okay otherwise not okay. (Excel Sheet 37-42 is used for calculation).

46. Neural Network with 1 hidden layer with relu activation with decaying learning rate Test Case

In Neural Network with 1 hidden layer with relu activation decaying learning rate test case NN_1_layer_test () function is tested for its functionality in train.py file. NN_1_layer_test () function is reduced down version of NN_1_layer () function in train.py. NN_1_layer_test () function train neural network with 10 input neuron, 10 neuron in hidden layer and 10 neuron in output layer.
In this case, input to a function is

- random image array of dim [10 x4]
- pre -define true label array of dimension [10x4]
- 2 random array of dimension [10x10] representing weights in layer 1 and layer2
- 2 random array of dimension [10x1] representing biases in layer 1 and layer2
- activation function value = “relu”
- L1 reg value = “n”
- Decay = “y”
- value of beta = 0.9
- batch = 4
- decay rate =0.98
- number of hidden neuron in layer is 10 each
- optimizer = “sgd”
- a value of alpha = 0.00001
- number epoch to train = 5
- num= 1

Output to the function in this case is

- 2 trained array of dimension [10x10] representing trained weights of layer 1 and layer2 for network after 5 epochs.
- 2 trained array of dimension [10x1] representing trained biases in layer 1 and layer2 for network after 5 epochs.
- Loss value of the neural network
- Accuracy value of the neural network.

Input values are feed to NN_1_layers_test () function from valid_excel.xlsx file. The output value of the function is then compared to the calculated value of arrays in excel. If function output values match with excel calculated value then function tested okay otherwise not okay. (Excel Sheet 67--72 is used for calculation).

47. Neural Network with zero hidden layers with decaying learning rate Test Case

In Neural Network with no hidden layer with decaying learning rate test case NN_0_layer_test () function is tested for its functionality in train.py file. NN_0_layer_test () function is reduced down version of NN_0_layer () function in train.py. NN_0_layer_test () function train the neural network with 10 input neurons and 10 neurons in the output layer.
In this case, input to a function is

- random image array of dim [10 x4]
- pre -define true label array of dimension [10x4]
- 1 random array of dimension [10x10] representing weights in layer 1
- 1 random array of dimension [10x1] representing biases in layer 1
- activation function value = “n”
- L1 reg value = “n”
- Decay = “y”
- value of beta = 0.9
- batch = 4
- decay_rate =0.98
- number of hidden neuron in layer is 10 each
- optimizer = “sdg”
- a value of alpha = 0.00001
- number epoch to train = 5
- num= 0

Output to the function in this case is

- 1 trained array of dimension [10x10] representing trained weights of layer 1 for network after 5 epochs.
- 1 trained array of dimension [10x1] representing trained biases in layer 1 for network after 5 epochs.
- Loss value of the neural network

- Accuracy value of the neural network.

Input values are feed to NN_0_layers_test () function from valid_excel.xlsx file. The output value of the function is then compared to the calculated value of arrays in excel. If function output values match with excel calculated value then function tested okay otherwise not okay. (Excel Sheet 91-96 is used for calculation).

48. Neural Network with 2 hidden layers with relu activation with L1 regression Test Case

In Neural Network with 2 hidden layer with relu activation with L1 regression test case NN_2_layer_test () function is tested for its functionality in train.py file. NN_2_layer_test () function is reduced down version of NN_2_layer () function in train.py. NN_2_layer_test () function train neural network with 10 input neuron, 10 neuron in 1st hidden layer, 10 neuron in 2nd hidden layer and 10 neuron in output layer.
In this case, input to a function is

- random image array of dim [10 x4]
- pre -define true label array of dimension [10x4]
- 3 random array of dimension [10x10] representing weights in layer 1, layer2, layer 3
- 3 random array of dimension [10x1] representing biases in layer 1, layer2, layer 3
- activation function value = “relu”
- L1 reg value = “y”
- Decay = “n”
- value of beta = 0.9
- batch = 4
- decay_rate =0.98
- number of hidden neuron in layer is 10 each
- optimizer = “sgd”
- a value of alpha = 0.00001
- number epoch to train = 5
- num= 2

Output to the function in this case is

- 3 trained array of dimension [10x10] representing trained weights of layer 1, layer2, layer 3 for network after 5 epoch.
- 3 trained array of dimension [10x1] representing trained biases in layer 1, layer2, layer 3 for network after 5 epochs.
- Loss value of the neural network
- Accuracy value of the neural network.

Input values are feed to NN_2_layers_test () function from valid_excel.xlsx file. The output value of the function is then compared to the calculated value of arrays in excel. If function output values match with excel calculated value then function tested okay otherwise not okay. (Excel Sheet 43-48 is used for calculation).

49. Neural Network with 1 hidden layer with relu activation with L1 regression Test Case

In Neural Network with 1 hidden layer with relu activation with L1 regression test case NN_1_layer_test () function is tested for its functionality in train.py file. NN_1_layer_test () function is reduced down version of NN_1_layer () function in train.py. NN_1_layer_test () function train neural network with 10 input neuron, 10 neuron in hidden layer and 10 neuron in output layer.
In this case, input to a function is

- random image array of dim [10 x4]
- pre -define true label array of dimension [10x4]
- 2 random array of dimension [10x10] representing weights in layer 1 and layer2
- 2 random array of dimension [10x1] representing biases in layer 1 and layer2
- activation function value = “relu”
- L1 reg value = “y”
- Decay = “n”
- value of beta = 0.9
- batch = 4
- decay rate =0.98
- number of hidden neuron in layer is 10 each
- optimizer = “sgd”
- a value of alpha = 0.00001
- number epoch to train = 5
- num= 1

Output to the function in this case is

- 2 trained array of dimension [10x10] representing trained weights of layer 1 and layer2 for network after 5 epochs.
- 2 trained array of dimension [10x1] representing trained biases in layer 1 and layer2 for network after 5 epochs.
- Loss value of the neural network
- Accuracy value of the neural network.

Input values are feed to NN_1_layers_test () function from valid_excel.xlsx file. The output value of the function is then compared to the calculated value of arrays in excel. If function output values match with excel calculated value then function tested okay otherwise not okay. (Excel Sheet 73--78 is used for calculation).

50. Neural Network with zero hidden layers with L1 regression Test Case

In Neural Network with no hidden layer with L1 regression test case NN_0_layer_test () function is tested for its functionality in train.py file. NN_0_layer_test () function is

reduced down version of NN_0_layer () function in train.py. NN_0_layer_test () function train neural network with 10 input neuron and 10 neuron in output layer.
In this case, input to a function is

- random image array of dim [10 x4]
- pre -define true label array of dimension [10x4]
- 1 random array of dimension [10x10] representing weights in layer 1
- 1 random array of dimension [10x1] representing biases in layer 1
- activation function value = “n”
- L1 reg value = “y”
- Decay = “n”
- value of beta = 0.00001
- batch = 4
- decay_rate = 0.98
- number of hidden neuron in layer is 10 each
- optimizer = “sdg”
- a value of alpha = 0.00001
- number epoch to train = 5
- num= 0

Output to the function in this case is

- 1 trained array of dimension [10x10] representing trained weights of layer 1 for network after 5 epochs.
- 1 trained array of dimension [10x1] representing trained biases in layer 1 for network after 5 epochs.
- Loss value of the neural network
- Accuracy value of the neural network.

Input values are feed to NN_0_layers_test () function from valid_excel.xlsx file. The output value of the function is then compared to the calculated value of arrays in excel. If function output values match with excel calculated value then function tested okay otherwise not okay. (Excel Sheet 97-102 is used for calculation)

Structure of train.py:

1. Main Function ():

   Main functions ask for the number of hidden layers from the user and call the neural network function based on user input. The main function receives the Loss history list, Accuracy history list and learning rate history list for training, validation and test data set as the output from neural network function and saves them into .csv file and print the notification after saving the .csv file for a specific list.

2. NN_2_layers ():

   Neural Network with 2 hidden layers ask for input from a user for

   - Type of normalization [Simple or Normal]
   - Type weights initialization [Gauss distribution or Xavier initialization]
   - Activation function to be used in hidden layers [Relu or |Sigmoid]
   - Option to implement L1 regression [y/n]
   - Option to implement decaying learning rate [y/n]
   - Type of optimizer to be implemented during training [Stochastic Gradient descent or Momentum optimizer]
   - Number of Epoch for training the model.

   Using input from the user function creates the neural network of 2 hidden layers with 64 neurons each with 784 input neurons and 10 output neurons and trains them for a given number of epochs. After training function saves the weights and biases of the neural network in individual .csv file with notification and returns Loss history list, Accuracy history list, and learning rate history list for training, validation and test data set to the main function.

3. NN_1_layers ():

   Neural Network with 1 hidden layer ask for input from the user for

   - Type of normalization [Simple or Normal]
   - Type weights initialization [Gauss distribution or Xavier initialization]
   - Activation function to be used in hidden layers [Relu or |Sigmoid]
   - Option to implement L1 regression [y/n]
   - Option to implement decaying learning rate [y/n]
   - Type of optimizer to be implemented during training [Stochastic Gradient descent or Momentum optimizer]
   - Number of Epoch for training the model.

Using input from the user function creates the neural network of 1 hidden layer with 64 neurons with 784 input neurons and 10 output neurons and trains them for a given number of epochs. After training function saves the weights and biases of the neural network in individual .csv file with notification and returns Loss history list, Accuracy history list, and learning rate history list for training, validation and test data set to the main function.

4. NN_0_layers ():

   Neural Network with 0 hidden layers ask for input from the user for

   - Type of normalization [Simple or Normal]
   - Type weights initialization [Gauss distribution or Xavier initialization]
   - Option to implement L1 regression [y/n]
   - Option to implement decaying learning rate [y/n]
   - Type of optimizer to be implemented during training [Stochastic Gradient descent or Momentum optimizer]
   - Number of Epoch for training the model.

   Using input from the user function creates the neural network of 0 hidden layers with 784 input neurons and 10 output neurons and trains them for a given number of epochs. After training function saves the weights and biases of the neural network in individual .csv file with notification and returns Loss history list, Accuracy history list, and learning rate history list for training, validation and test data set to the main function.

5. NN_2_layers_test (weights_1, baises_1, weights_2, baises_2, weights_3, baises_3, image array, label array):

   NN_2_layers_test function is called by verify.py file to test the functionality of NN_2_layers function which has 784 neurons as input, 64 neurons in 1st hidden layer, 64 hidden neurons in 2nd hidden layer and 10 neurons as output and get weights from random initialization functions whereas NN_2_layers_test function has 10 neurons as input, 10 neurons in 1st hidden layer, 10 neurons in 2nd hidden layer and 10 neurons in the output layer and get predefined weights of dimension[10 x10] and biases of dimension [10 x 1], predefined image array of dimension [10 x 4] and label array of dimension [10 x 4] as input from verify.py file.

   NN_2_layers_test function ask for input from the verify.py file

   - Activation function to be used in hidden layers [Relu or |Sigmoid]
   - Option to implement L1 regression [y/n]
   - Option to implement decaying learning rate [y/n]

- Type of optimizer to be implemented during training [Stochastic Gradient descent or Momentum optimizer]

Using input from verify.py file function creates the neural network of 2 hidden layers with 10 neurons each with 10 input neurons and 10 output neurons and trains them for 5 epoch. After the training function returns weights and biases, training loss, training accuracy to verify.py file to compare the value of weights, biases, training loss and training accuracy values which are also calculated in excel file for validating the functionality of NN_2_layers function.

6. NN_1_layers_test (weights_1, baises_1, weights_2, baises_2, image array, label array):

NN_1_layers_test function is called by verify.py file to test the functionality of NN_1_layers function which has 784 neurons as input, 64 neurons in 1st hidden layer and 10 neurons as output and get weights from random initialization functions whereas NN_1_layers_test function has 10 neurons as input, 10 neurons in 1st hidden layer, and 10 neurons in the output layer and get predefined weights of dimension[10 x10] and biases of dimension [10 x 1], predefined image array of dimension [10 x 4] and label array of dimension [10 x 4] as input from verify.py file.

NN_1_layers_test function ask for input from the verify.py file for

- Activation function to be used in hidden layers [Relu or |Sigmoid]
- Option to implement L1 regression [y/n]
- Option to implement decaying learning rate [y/n]
- Type of optimizer to be implemented during training [Stochastic Gradient descent or Momentum optimizer]

Using input from verify.py file function creates the neural network of 1 hidden layer 10 neurons with 10 input neurons and 10 output neurons and trains them for 5 epoch. After the training function returns weights and biases, training loss, training accuracy to verify.py file to compare the value of weights, biases, training loss and training accuracy values which are also calculated in excel file for validating the functionality of NN_1_layers function.

7. NN_0_layers_test (weights_1, baises_1, image array, label array):

NN_0_layers_test function is called by verify.py file to test the functionality of NN_0_layers function which has 784 neurons as input (zero hidden neurons) 10 neurons as output and get weights from random initialization functions whereas

NN_0_layers_test function has 10 neurons as input (zero hidden neurons) 10 neurons in the output layer and get predefined weights of dimension[10 x10] and biases of dimension [10 x 1], predefined image array of dimension [10 x 4] and label array of dimension [10 x 4] as input from verify.py file.

NN_0_layers_test function ask for input from the verify.py

- Option to implement L1 regression [y/n]
- Option to implement decaying learning rate [y/n]
- Type of optimizer to be implemented during training [Stochastic Gradient descent or Momentum optimizer]

Using input from verify.py file function creates the neural network of zero hidden layers with 10 input neurons and 10 output neurons and trains them for 5 epoch. After the training function returns weights and biases, training loss, training accuracy to verify.py file to compare the value of weights, biases, training loss and training accuracy values which are also calculated in excel file for validating the functionality of NN_0_layers function.

8. Data Pre-Processing ():

Data pre-processing function reads the MNIST data from the files

- train-images-idx3-ubyte.gz
- train-labels-idx1-ubyte.gz
- t10k-images-idx3-ubyte.gz
- t10k-labels-idx1-ubyte.gz

And resize and split the image data and label data into

1. Training data set
- Training images data of dimension [58,000 x 784]
- Training one hot label data of dimension [58,000 x 10]

2. Validation data set
- Validation images data of dimension [6,000 x 784]
- Validation one hot label data of dimension [6,000 x 10]

3. Test data set
- Test images data of dimension [6,000 x 784]
- Test one hot label data of dimension [6,000 x 10]

Data Pre- Processing function carry out normalization (simple normalization or normal normalization) of the data on basis of option selected by user. Function return normalized training image data [58,000 x 784], training one hot label data [58,000 x 10], normalized validation image data [6,000 x 784],, validation one hot label data [6,000 x 10], normalized test image data [6,000 x 784], and test one hot label data [6,000 x 10] .

9. SGD_optimizer ():

   Stochastic Gradient Optimizer function updates the weights and biases in every training step. The function gets the input of weights, biases, dloss_weights, dloss_dbiases and learning rate value from the neural network function, update parameters and return the updated parameters to the neural network for every training step. Function return updated weights and biases.

10. Momentum_optimizer ():

    Momentum Optimizer function is an optimizer that uses momentum parameter beta to create uphill rolling down effect for weights and biases to reach the local minima of the loss function. Momentum optimizer updates the weights, biases and also moving weights and moving biases in every training step. The function gets the input of weights, biases, dloss_weights, dloss_dbiases, moving weights, moving biases and learning rate value from the neural network function, the update parameters and return the updated parameters to the neural network for every training step. Function return updated weights, biases and updated mov_weights and mov_biases.

11. Loss_function ():

    The loss function calculates the overall loss of the neural network at the end of each epoch. In this project I have implemented cross-entropy loss with regression loss. The function gets the input of predicted array, true array, weights and biases from the neural network it calculates the cross-entropy loss and regression loss for the model and returns the single loss value.

12. Reg_loss ():

    Reg_loss function calculates the regression loss of the neural network at the end of each epoch. Reg_loss function gets the input value of weights from the loss function and calculate regression loss for the model and return the single regression loss value.

13. Forward_prop_for_loss ():

Forward_prop_for_loss function create the neural network and calculate the prediction array for the model at the end of each epoch to calculate the loss for training, validation and test data set. Forward_prop_for_loss function gets the input of weights, biases, image array, label array and hyperparameter values required to create the model, function calculate the loss for the required data set and return the singe value of the loss.

14. Accuracy ():

    Accuracy function calculates the accuracy of the model by comparing the result predicted by the model and true results available from the data set label file. Accuracy gets the input of weights, biases, image array, label array and activation function from the neural network and compares the values predicted by the model to the true array, calculates the accuracy and returns the single value of accuracy to the neural network.

15. Normal Normalization ():

    Normal Normalization function converts the image array data (values from 0 to 255) into normally distributed data. The function calculates the mean value for the image data and subtracts it from all data values and then divides all data values by the standard deviation value of the data. The function returns the normalized image data to the data pre-processing function.

16. Simple Normalization ():

    Normal Normalization function converts the image array data (values from 0 to 255) into data distributed in between 0 to 1 by dividing all value by 255 i.e. the maximum pixel value. The function returns the normalized image data to the data pre-processing function.

17. Gaussian initialization ():

    Gaussian initialization function initialization of the weights for the neural network in the initial stage by using Gaussian distribution function available in python with mean zero and of standard deviation value 0.01. The function returns the randomly initialize weights to the neural network.

18. Xavier initialization ():

    Xavier initialization function initialization of the weights for the neural network in the initial stage by using Gaussian distribution function available in python with mean zero and the standard deviation value equal to √ (1/weight_array_row). The function returns the randomly initialize weights to the neural network.

19. Learning rate decay ():

    The learning rate decay function reduces the learning rate value at the end of an epoch for the neural network. Function calculates the new learning rate at end of each epoch by multiplying the learning rate with learning rate decay constant. The function returns the single learning rate value to the neural network.

20. Save fun ():

    Save function save the weights array and biases array to .csv file at the end of the training with the name of string input to the function by the neural network. Function return none value to the neural network

21. Save fun list ():

    Save function save the loss history list, accuracy list and learning rate history list to .csv file at end of the training with the name of string input to the function by the main function. The function returns none value to the main function.

Usage Manual for train.py

train.py in a single run

- Creates the neural network of 2 hidden layers or 1 hidden layer or 0 hidden layer
- Reads the MNIST data
- Initialize the weights and biases for the network
- Train the network for a given number of epochs.
- Display the results of accuracy and loss of network in the form of graphs.
- Save the trained weights, biases, loss history, accuracy history and learning rate history after each run.

Requirements to run train.py file

- The Cmd environment used to run train.py file should be installed with the following packages.

  1. Python 3.6 minimum
  2. Numpy
  3. Gzip
  4. matplotlib.pyplot
  5. tqdm
  6. CSV

- train.py file directory should contain the following MNIST dataset file. Following file act as input to the neural network during the training and testing the network

  1. t10k-images-idx3-ubyte.gz
  2. t10k-labels-idx1-ubyte.gz
  3. train-images-idx3-ubyte.gz
  4. train-labels-idx1-ubyte.gz

  If not available then the above-mentioned files can be downloaded from the website (link).

User input required to run train.py file

There are 8 user inputs required to run the train.py file. The required user input for each case is enlisted below.

1. Enter Number of Hidden layers in Model [**0 or 1 or 2**]:
   User input can be (**0, zero, Zero**) for zero hidden layers in a neural network or (**1, one, One**) for one hidden layer in a neural network with 64 neurons or (**2, Two, two**) for two hidden layers in a neural network with 64 neurons each.

2. Enter the type of normalization for Data [**Simple or Normal**]:

User input can be (**Simple, simple**) for simple normalization or (**Normal, normal**) for normal normalization of Data set.

3. Enter function for Weights value initialization [**Gauss or Xavier**]:
   User input can be (**Gauss, gauss)** for Gaussian initialization or (**Xavier, xavier**) for the Xavier initialization of weights in the neural network.

4. Enter Activation function for hidden layers [**Relu or Sigmoid**]:
   User input can be (**Relu, relu)** for rectified linear activation function or (**Sigmoid, sigmoid**) for the sigmoid activation function for the hidden layers.

5. Want to implement L1 Regression in the Model [**y / n**]:
   User input can be (**Y, y)** for implementing L1 regression or (**N,n**) for not implementing L1 regression during training the model.

6. Want to implement the decaying learning rate [**y / n**]:
   User input can be (**Y, y)** for implementing the decaying learning rate or (**N,n**) for not implementing the decaying learning rate during training the model.

7. Enter type of Optimizer want to use to train the model [**SGD or Momentum]:**
   User input can be (**sgd, SGD)** for implementing stochastic gradient descent or (**momentum, Momentum**) for implementing momentum optimizer during training the model.

8. Enter Number of Epochs for training the model:
   User input should be an **integer** which is equivalent to the number of epochs the user wants to train the selected neural network.

The output of train.py file

When the process of training the network ends the following output files will be generated in the working directory of the user.

- Trained weights and biases of the neural network will be saved in the working directory in the form of .csv file and the user will get a notification on the cmd window when files are saved.

- Graph for Loss history, Accuracy history and learning rate history for the training dataset, validation dataset and test dataset will be displayed and saved in the working directory.

- Loss history list, Accuracy history list, and learning rate history list for the training dataset, validation dataset and test dataset will be saved in the form of .csv file working directory and the user will get a notification on cmd window when files are saved.

Usage Manual for predict.py

- Predict.py file uses the trained weights and biases (which is output on train.py file) to predict the test dataset MNIST images.
- In predict.py file, I have used the 1 hidden layer models with trained weights and biases. (I reason for selecting this model for prediction is, this model has better accuracy on MNIST test set than other 0 hidden layer model or 2 hidden layer model. I have shown the study of the performance of models in the latter part of this report.)
- predict.py file read the MNIST test dataset and feed the image array of 6000 length in the model one by one and use the model to predict the label of images.
- predict.py display predicted images with the predicted label of the image by model and true label of an image on the display image window title.

Requirements to run train.py file

- The Cmd environment used to run train.py file should be installed with the following packages.

    1. Python 3.6 minimum
    2. Numpy
    3. Gzip
    4. matplotlib.pyplot
    5. CSV

- predict.py file directory should contain the following MNIST dataset file. Following file act as input to the neural network testing of MNIST images

    1. t10k-images-idx3-ubyte.gz
    2. t10k-labels-idx1-ubyte.gz
    3. train.py
    4. saved files folder (contain .csv for weights and biases)

The output of predict.py file

- The total accuracy of the model for the test dataset will display on the command line.
- Predicted Test data set images will be displayed in for loop with predicted labels and true label on image window title for comparison.

Usage Manual for verify.py

- Verify.py file consists of functions used to test different functionality of functions available in train.py file.
- Verify.py file can execute a single test function in one run.
- Verify.py takes a digit value as input associated with the test function to run that test function (for simplicity). (list of test functions is on the next page.)
- Verify.py file gets input data from the valid_excel.xlsx file which is feed to testing function and then the output of the function is compared to the output of calculations done in excel file.
- If the final calculation of the excel file matches to tested function output with the same input the function is tested okay otherwise function tested not okay.

Requirements to run verify.py file

- The Cmd environment used to run train.py file should be installed with the following packages.

  1. Python 3.6 minimum
  2. Numpy
  3. Gzip
  4. matplotlib.pyplot
  5. tqdm
  6. CSV
  7. xlrd

- verify.py file directory should contain the following files. Following files act as input to verify.py

  1. train.py
  2. valid_excel.xlsx

User input required to run verify.py file

- Verify.py file can execute a single test function in one run. To run a specific function user need to input the digit associated with the function (from the list on the next page).
- Digits associate with functions from 1 to 16 are test function testing the functionality of functions in train.py file individually. (Individual function from train.py get the input from the test function and output is checked against the values from excels files.)
- Digit associated with functions from 17- 36 are test functions testing the functionality of functions working with other functions. (not individual functions) In this group the

function (from train.py file) which is tested is also calling other functions for input in train.py file. In this group the same functionality is tested for different cases like layer0, layer1 and layer2.

- Digit associated with functions from 37 -50 are test functions testing the functionality of reduced down version of neural network for layer 0, layer 1 and layer2. Reduced down version of neural network has 10 input neurons and 10 output neurons and 10 neurons in the hidden layer. The number of hidden layers depends upon the user selection for the hidden layer. In this group testing is done for a different case like different activation functions, different optimizers, with or without decaying learning rate, with or without L1 regression for all three reduced down versions of neural network.

The output of verify.py file

The output of the verfy.py file is the notification displayed in the command line. If the output of the function match with the output of the excel file for the same input then the notification displayed will be "function tested okay" otherwise display will be "function tested not okay."

List of verify.py functions with a digit and excel sheet used to input the data:

Digit Associated with function	Name of the test function of verify.py file	Spreadsheet used to test the functionality of function in train.py
1	Relu Activation Test Case	Sheet1
2	Relu Activation back Test Case	Sheet2
3	Sigmoid Activation Test Case	Sheet3
4	Sigmoid Activation back Test Case	Sheet4
5	SGD optimizer for 2 hidden layers Test Case	Sheet5
6	SGD optimizer for 1 hidden layers Test Case	Sheet5
7	SGD optimizer for 0 hidden layers Test Case	Sheet5
8	Momentum optimizer for 2 hidden layers Test Case	Sheet6
9	Momentum optimizer for 1 hidden layers Test Case	Sheet6
10	Momentum optimizer for 0 hidden layers Test Case	Sheet6

11	Regression Loss 2 hidden layer Test Case	Sheet7
12	Regression Loss 1 hidden layer Test Case	Sheet7
13	Regression Loss 0 hidden layer Test Case	Sheet7
14	Learning rate decay Test Case	Sheet8
15	Normal Normalization Test Case	Sheet9
16	Simple Normalization Test Case	Sheet10
17	Loss function with reg 2 hidden layer Test Case	Sheet14
18	Loss function with reg 1 hidden layer Test Case	Sheet14
19	Loss function with reg 0 hidden layer Test Case	Sheet14
20	Loss function with no regression Test Case	Sheet14
21	Forward_prop_for_loss 2 hidden layer with relu activation with regression Test Case	Sheet15
22	Forward_prop_for_loss 2 hidden layer with relu activation with no regression Test Case	Sheet15
23	Forward_prop_for_loss 2 hidden layer with sigmoid activation with regression Test Case	Sheet16
24	Forward_prop_for_loss 2 hidden layer with sigmoid activation with no regression Test Case	Sheet16
25	Forward_prop_for_loss 1 hidden layer with relu activation with regression Test Case	Sheet103
26	Forward_prop_for_loss 1 hidden layer with relu activation with no regression Test Case	Sheet103
27	Forward_prop_for_loss 1 hidden layer with sigmoid activation with no regression Test Case	Sheet104
28	Forward_prop_for_loss 1 hidden layer with sigmoid activation with no regression Test Case	Sheet104
29	Forward_prop_for_loss 0 hidden layer with regression Test Case	Sheet105
30	Forward_prop_for_loss 0 hidden layer with no regression Test Case	Sheet105
31	Accuracy with 2 hidden layer relu activation Test Case	Sheet17
32	Accuracy with 2 hidden layers sigmoid activation Test Case	Sheet18

33	Accuracy with 1 hidden layer relu activation Test Case	Sheet106
34	Accuracy with 1 hidden layer sigmoid activation Test Case	Sheet107
35	Accuracy with 0 hidden layer Test Case	Sheet108
36	Data Pre Processing Test case	-
37	Neural Network with 2 hidden layers with relu activation Test Case	Sheet19-24
38	Neural Network with 1 hidden layer with relu activation Test Case	Sheet 49-54
39	Neural Network with 0 hidden layer Test Case	Sheet 79-84
40	Neural Network with 2 hidden layers with sigmoid activation Test Case	Sheet25-30
41	Neural Network with 1 hidden layer with sigmoid activation Test Case	Sheet55-60
42	Neural Network with 2 hidden layers with relu activation with Momentum optimizer Test Case	Sheet31-36
43	Neural Network with 1 hidden layer with relu activation with Momentum optimizer Test Case	Sheet 61-66
44	Neural Network with 0 hidden layers with Momentum optimizer Test Case	Sheet85-90
45	Neural Network with 2 hidden layers with relu activation with decaying learning rate Test Case	Sheet 37-42
46	Neural Network with 1 hidden layer with relu activation with decaying learning rate Test Case	Sheet 67-72
47	Neural Network with 0 hidden layers with decaying learning rate Test Case	Sheet 91-96
48	Neural Network with 2 hidden layers with relu activation with L1 regression Test Case	Sheet43-48
49	Neural Network with 1 hidden layer with relu activation with L1 regression Test Case	Sheet73-78
50	Neural Network with zero hidden layers with L1 regression Test Case	Sheet 97-102

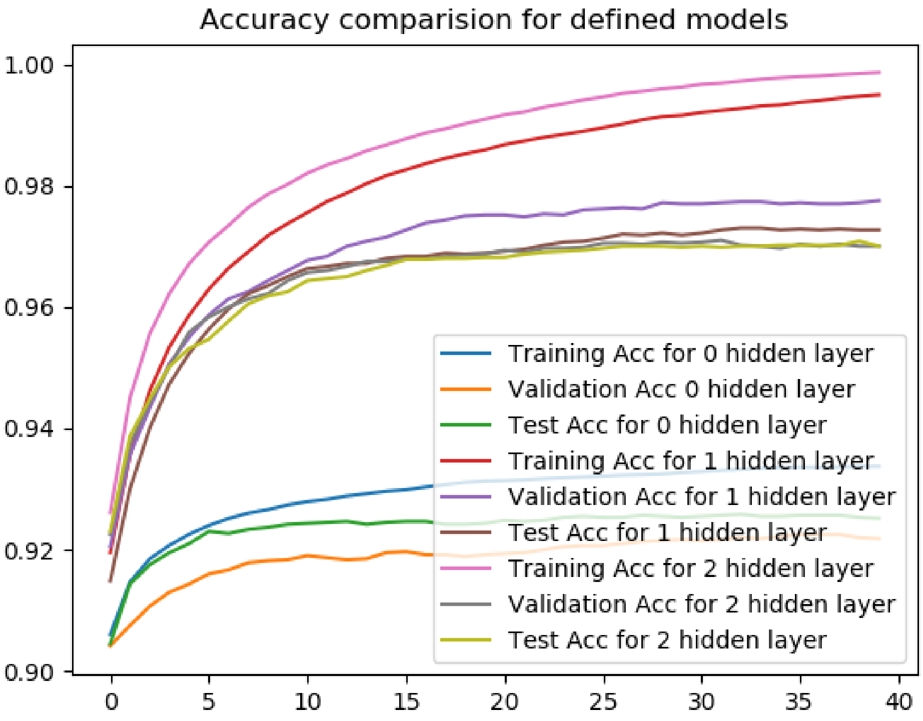
Model performance Study based on the result from train.py

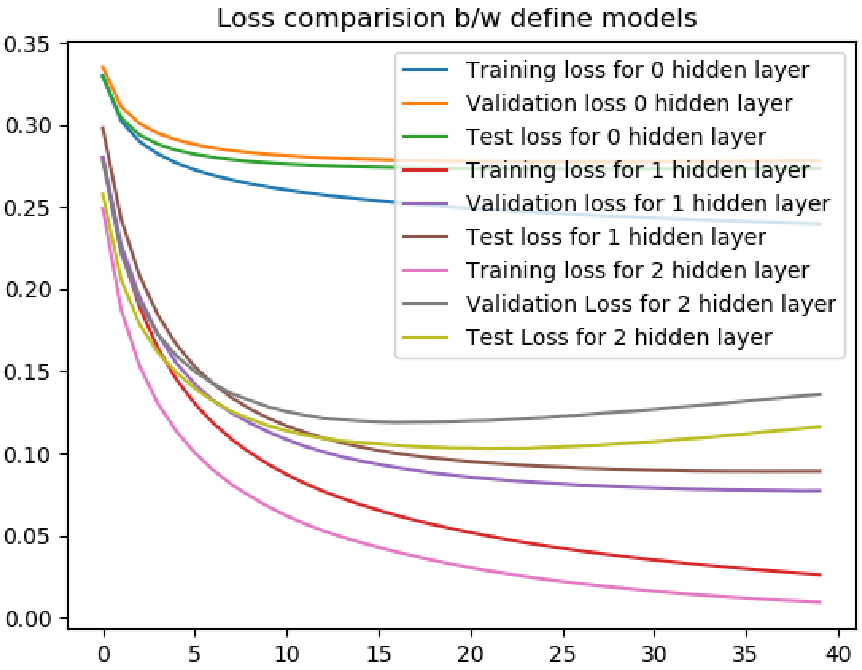
I have carries out a comparative analysis of the basis of the accuracy of different models. I carried out a training run for the model with 0 hidden layers, 1 hidden layer and with 2 hidden layers with the algorithm used to update weights, with different activation functions, different learning rates to find the best fit for the MNIST dataset.

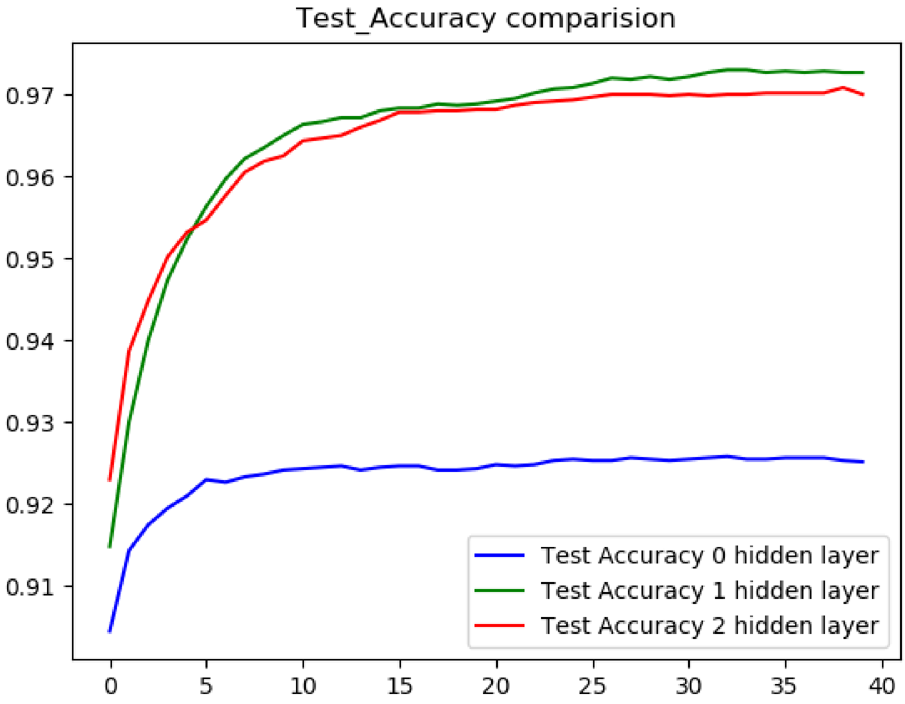
I have created the comparison graphs for accuracy and loss curve for different runs of 3 different layer models. I have divided them into 5 different categories.

1. Performance comparison of Models with Relu activation functions in the hidden layer with stochastic gradient descent optimizer.(for 3 models)
2. Performance comparison of Models with Sigmoid activation function in the hidden layer with stochastic gradient descent optimizer (for 3 models)
3. Performance comparison of Models with Relu activation function in the hidden layer and with Momentum optimizer (for 3 models)
4. Performance comparison of Models with Relu activation function in the hidden layer, with decaying learning rate and with stochastic gradient descent optimizer.(for 3 models)
5. Performance comparison of Models with Relu activation function in the hidden layer, with L1 regression and with stochastic gradient descent optimizer.(for 3 models).

1. Results for Models with Relu activation functions in the hidden layer with stochastic gradient descent optimizer.

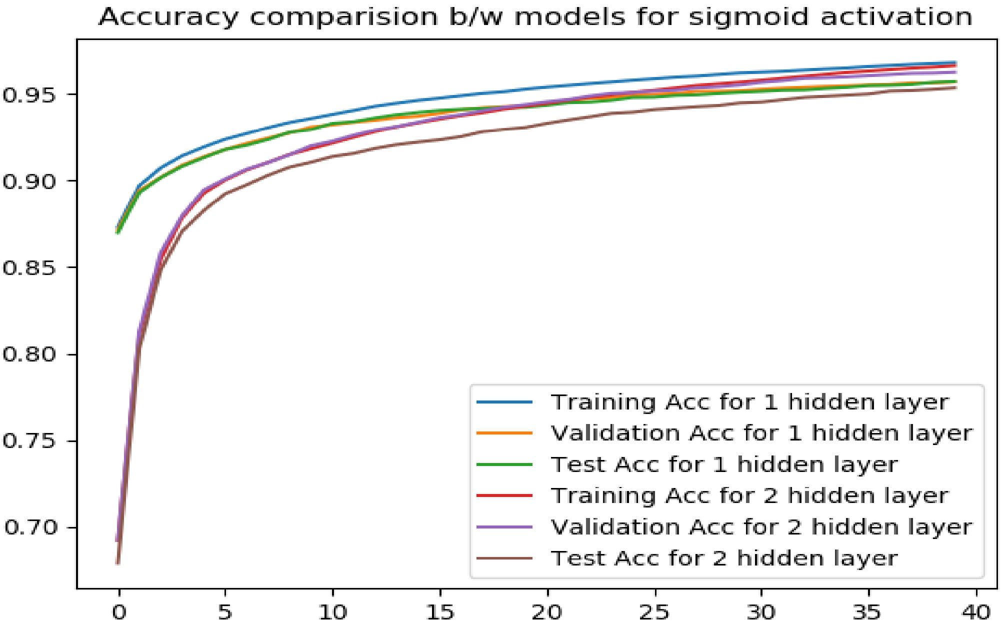


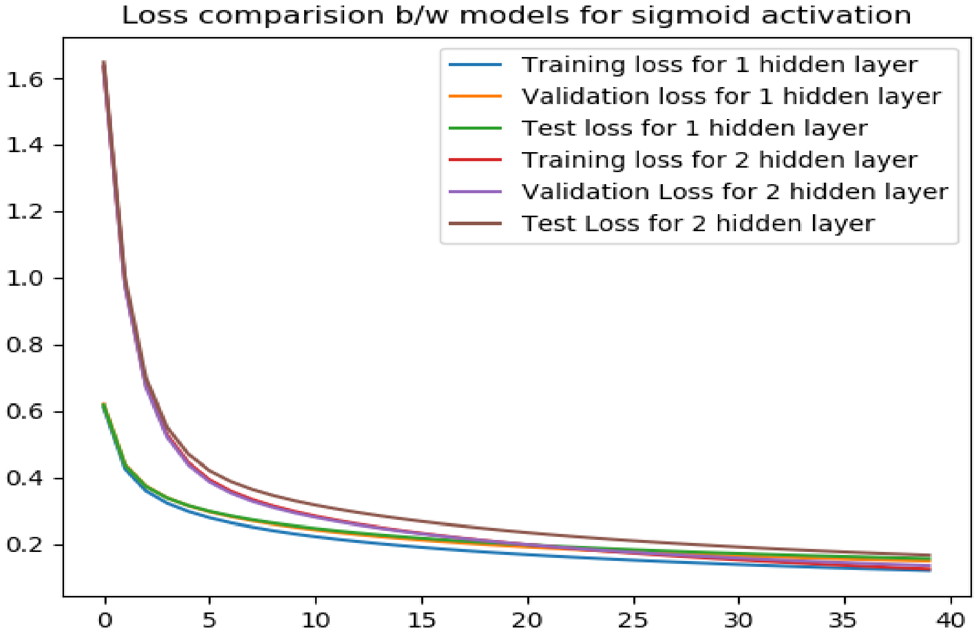




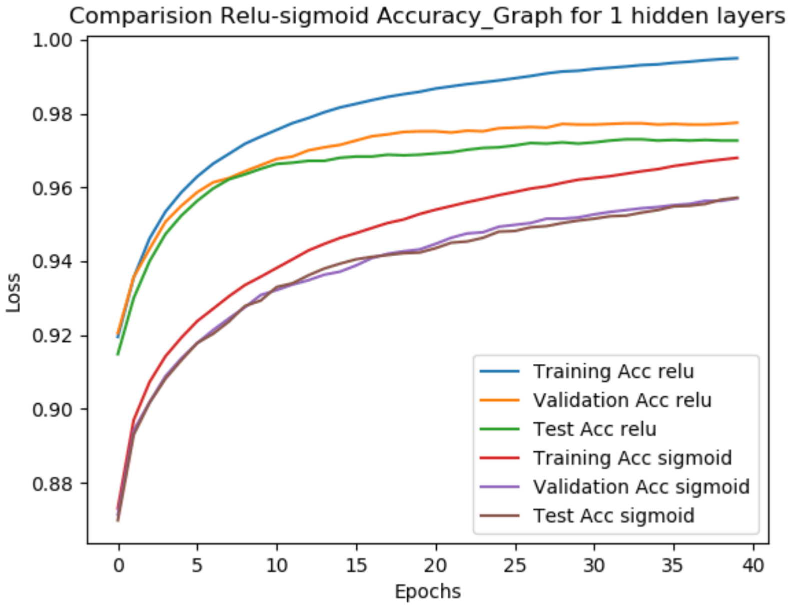
From the graphs we can see the model with 1 hidden layer and 64 neurons in the hidden layer is a better performing model than models with 2 hidden layers with 64 neurons in each layer and the model with 0 hidden layers.

2. Result of Models with Sigmoid activation function in the hidden layer with stochastic gradient descent optimizer.
   In the graph below, we see that the model with 1 hidden layer (64 neurons) with sigmoid activation function performs better than the model with 2 hidden layers (64 neurons) with sigmoid activation.

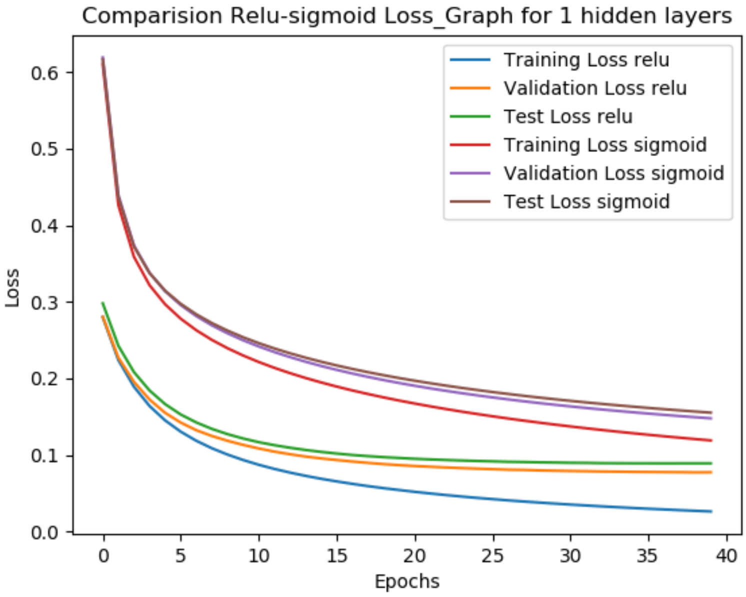




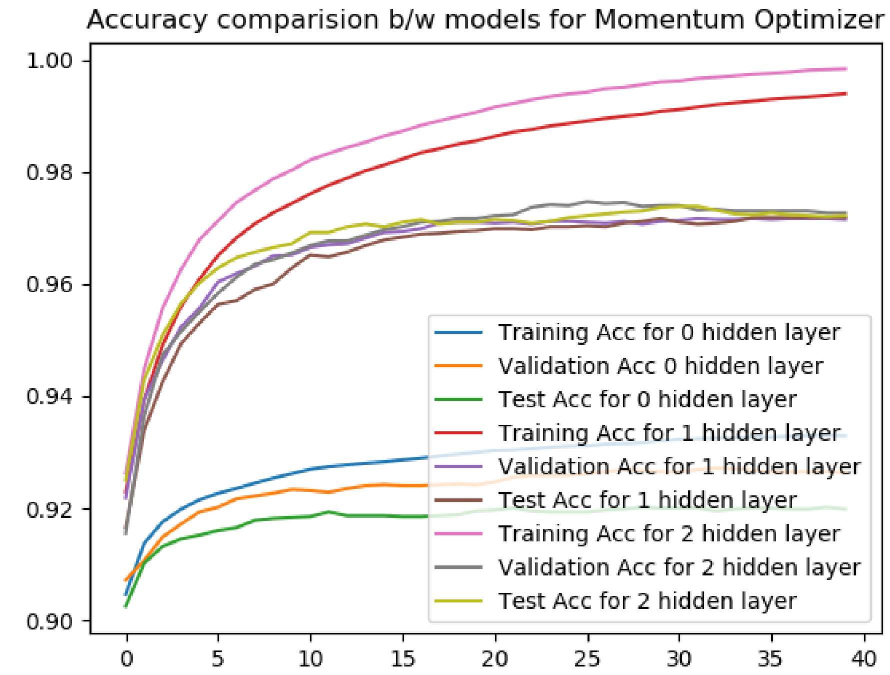
In the graph below we see that model with 1 hidden layer (64 neurons) with relu activation function perform better than the model with 1 hidden layer (64 neurons) with sigmoid activation

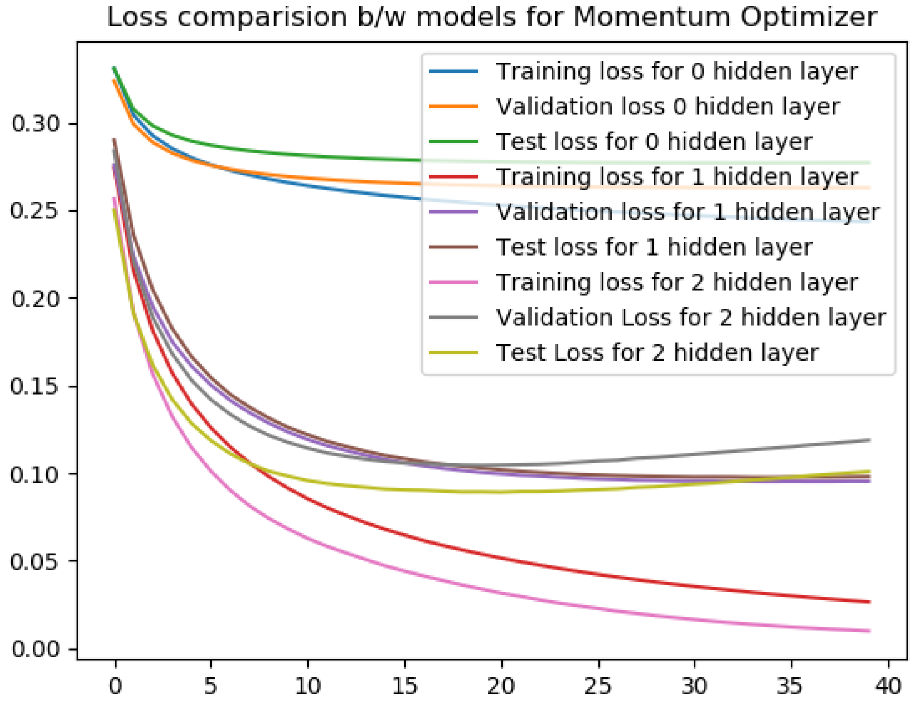


And the graph below we can see that the loss function for relu reduces much faster than the loss function of the sigmoid. This helps the relu activation models to train faster and takes less time to achieve higher accuracy as compared to sigmoid for the same availability of resources.

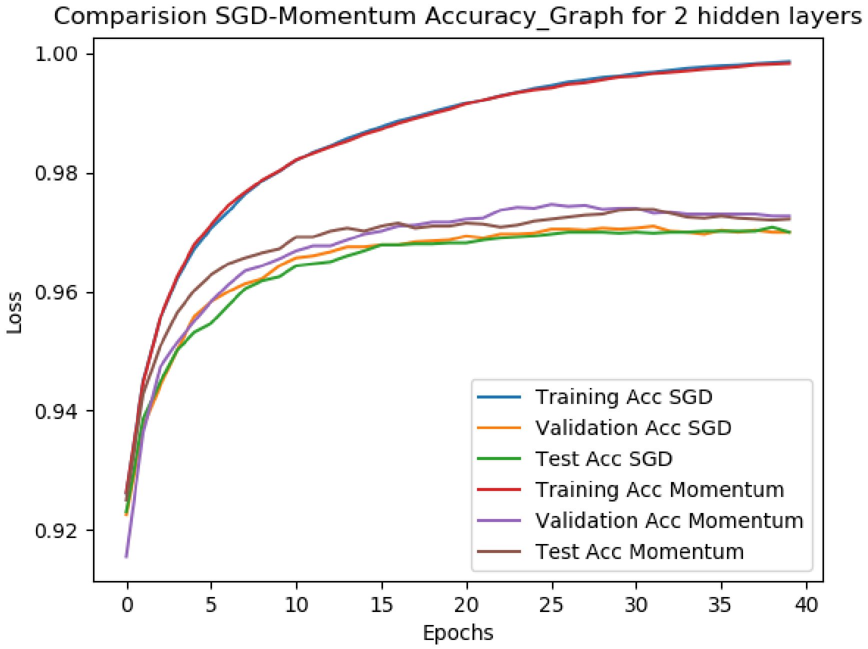


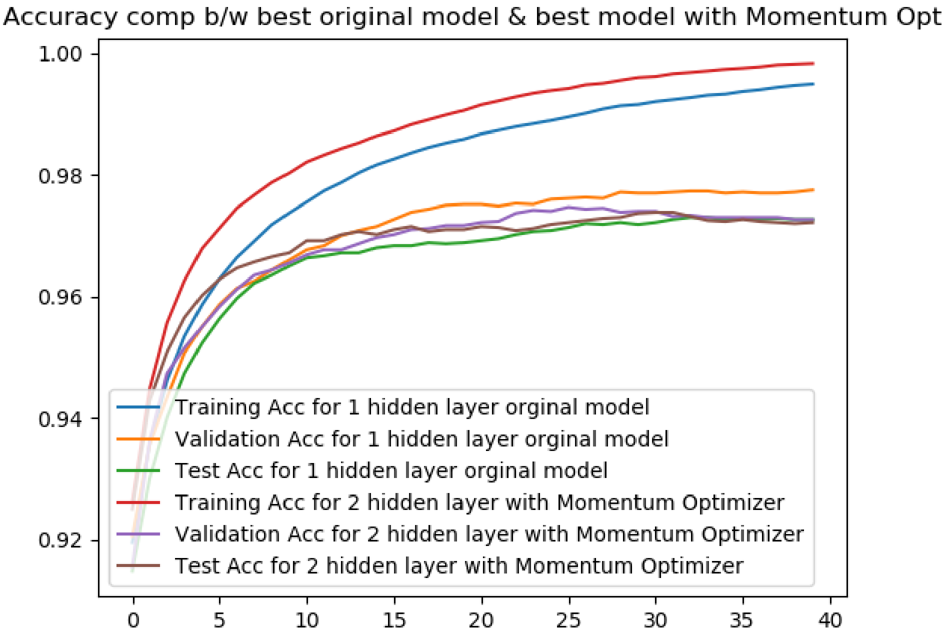
3. Models with Relu activation function in the hidden layer and with Momentum optimizer.



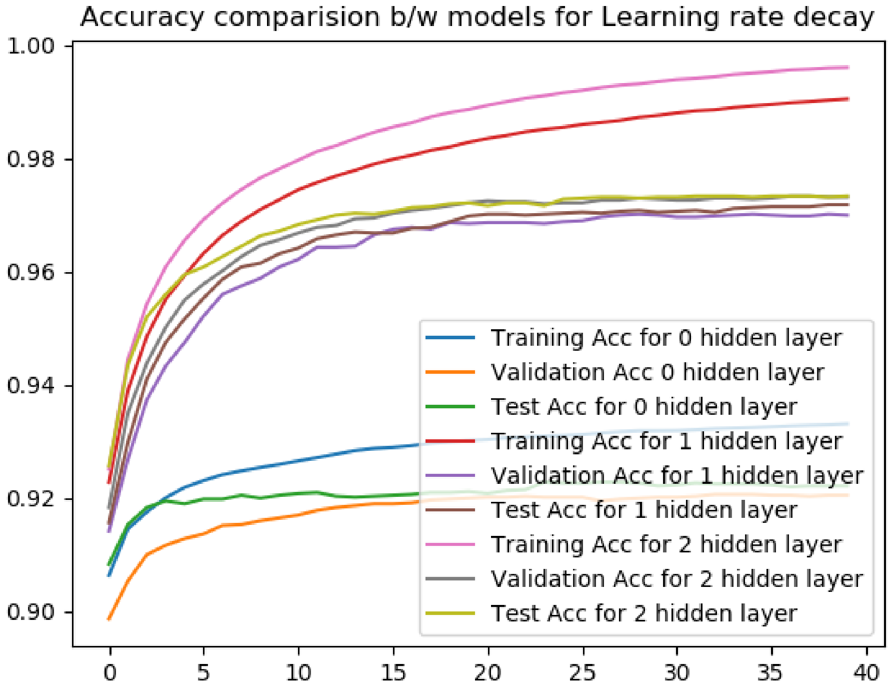


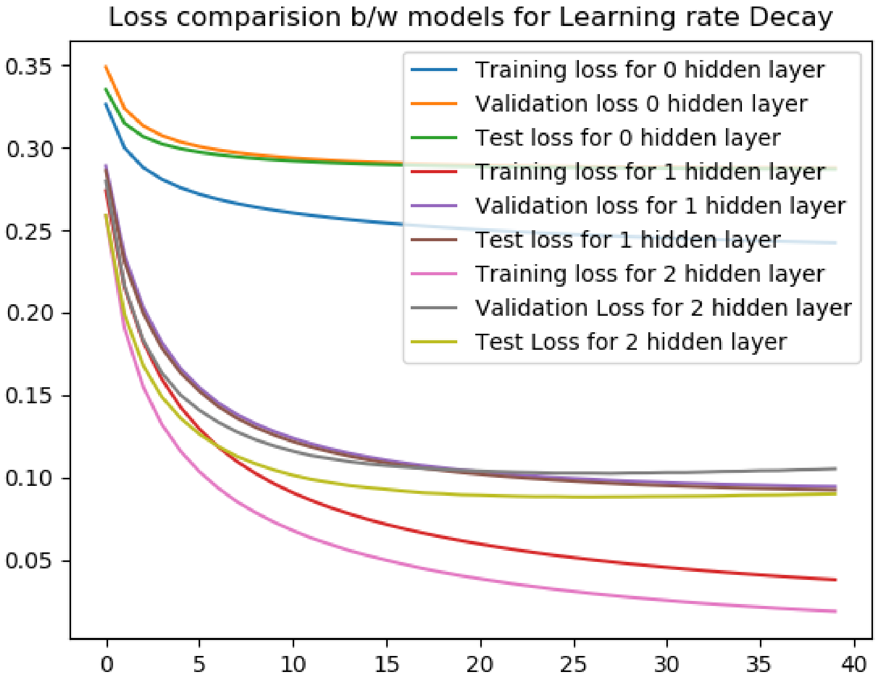
In the above accuracy graph we can see the test accuracy curve for 1 hidden layer model and 2 hidden layer model is pretty close. In another graph below I compare the model with SGD optimizer relu activation 2 hidden layers and model with momentum optimizer relu activation in hidden layers. In the curve below we can see with the use of momentum optimizer test accuracy of 2 hidden layer model become comparable to the original 1 hidden layer model.

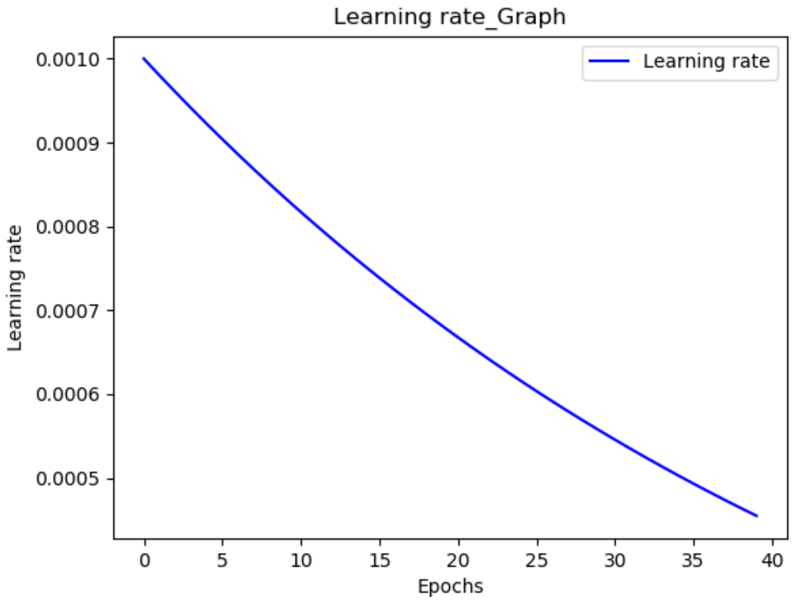




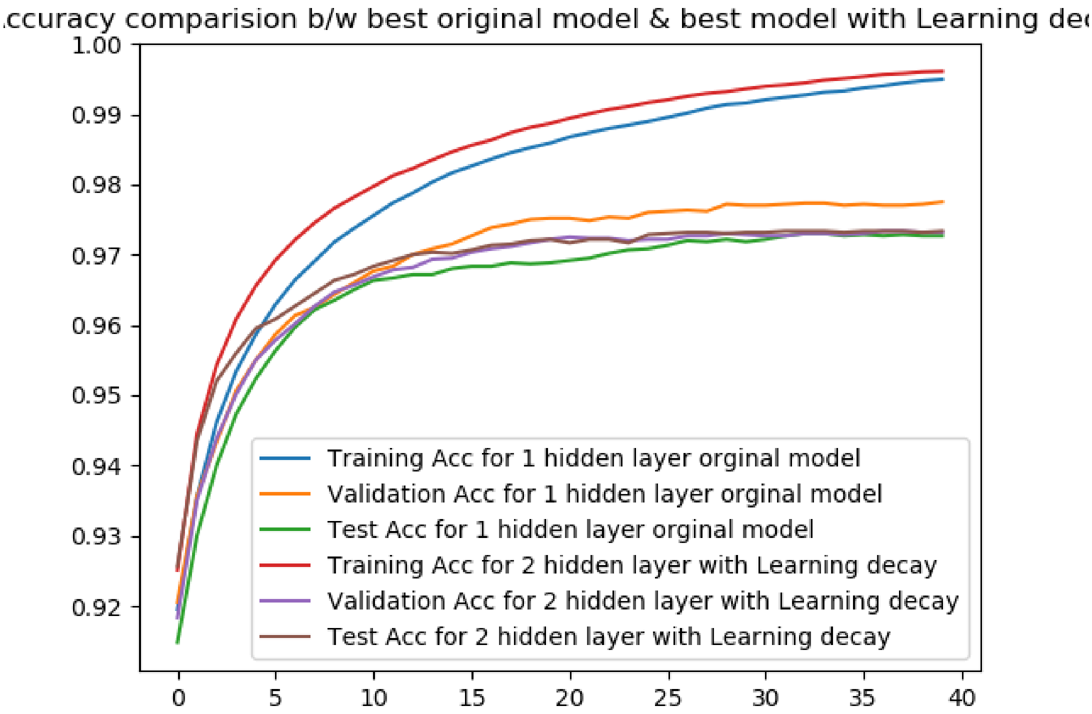
4. Models with Relu activation function in the hidden layer, with decaying learning rate and with stochastic gradient descent optimizer.



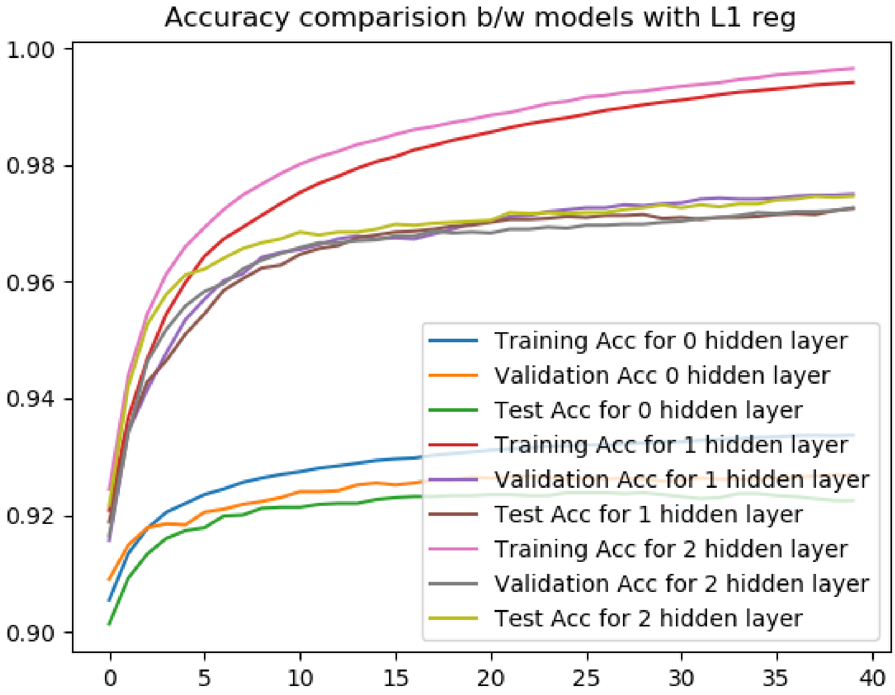


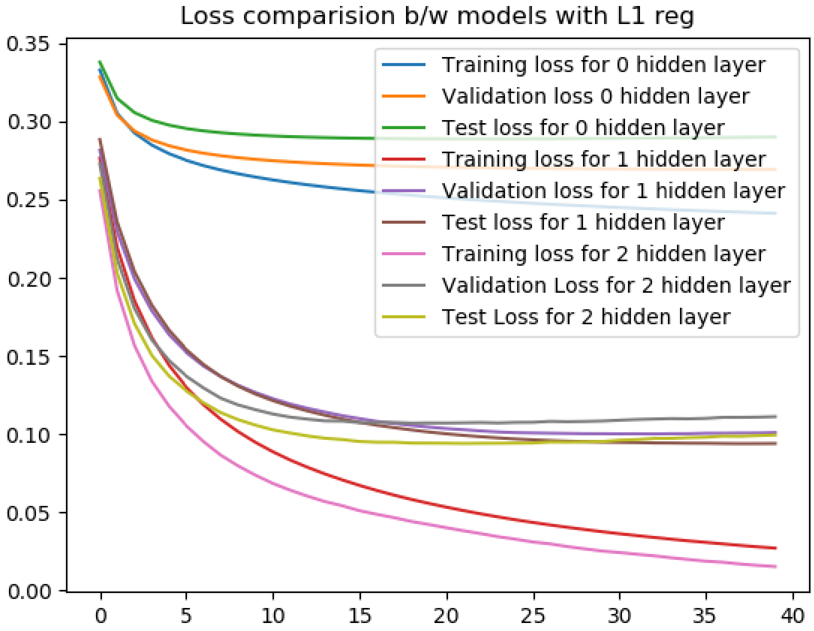


In the above graphs, we can see that the Test accuracy curve for 1 hidden layer model and 2 hidden layer models get very close. This means that decay in the learning rate also has an effect on in increase performance of the 2 hidden layer model test accuracy.

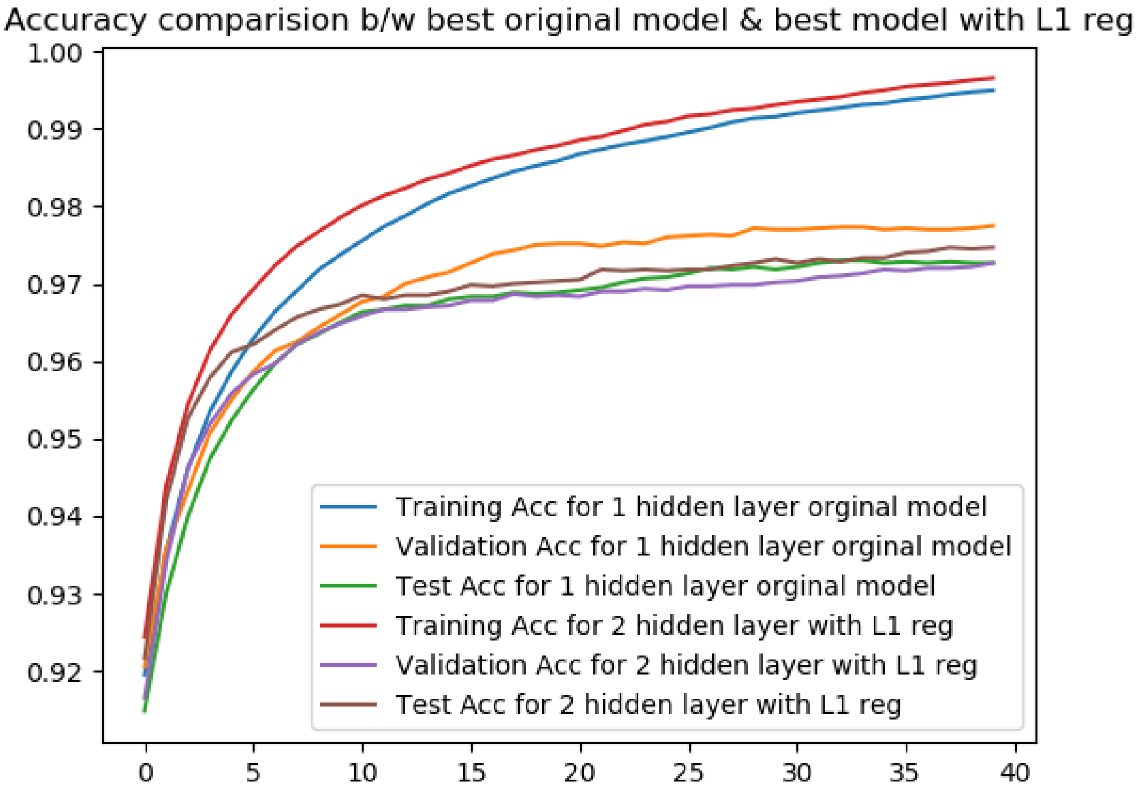


5. Models with Relu activation function in the hidden layer, with L1 regression and with stochastic gradient descent optimizer.





In the above graph, we can see that L1 regression can lead to an increase in performance similar to the best performing 1 hidden layer model. In the below graph, we can see it is pretty difficult to decide to choose a better performing model. But the lighter model with the same efficiency is always a better choice. So according to my analysis based on the data I collected. I think the model with 1 hidden layer with relu activation with stochastic gradient descent perform better than other available models.



Git log:

```
commit fba618df5bedcd089a79e14cd617b9fff79a3d20
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Mar 26 01:53:48 2020 +0530

    Add files via upload

commit cc5c24a4bd42d4b1c23ee89dd19836e1a1fec18e
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Mar 26 01:52:35 2020 +0530

    Add files via upload

commit 1a2c139040403f0eb6ef4de24ca502b2f2d499c2
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Mar 26 01:44:19 2020 +0530

    Add files via upload

commit bf643bb54fdb02530a5f01a0d01c3b887f39f41f
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Tue Mar 24 12:39:16 2020 +0530

    Add files via upload

commit 0ad43bc6cb945b6d9aa6d3c983019b00071cb6aa
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Tue Mar 24 12:19:07 2020 +0530

    Add files via upload

commit d6616a9443e3ef14cb8650787283a5abf6ad28a2
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Tue Mar 24 10:53:04 2020 +0530
```

```
    Add files via upload

commit ce5fd88611df355f7d24f6caec889a3c0495d536
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Tue Mar 24 04:54:55 2020 +0530

    Add files via upload

commit 11ce1b4ecb8f346198ae1653946cdd75e99c86ee
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Mon Mar 23 15:15:36 2020 +0530

    Add files via upload

commit 8efae306a3649b8e5b674c520b743155841fd5c2
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Mon Mar 23 15:14:50 2020 +0530

    Add files via upload

commit 680748f322cecb4726de9cd37a002a66b96e2d7d
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Sun Mar 22 07:26:20 2020 +0530

    Add files via upload

commit dabe1970e1b6b894cab85ee84a2073aeabb64b2b
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Sun Mar 22 07:25:38 2020 +0530

    Add files via upload

commit f055cc988f6b43d46ff51ef9a7901146fbbc865e
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Sat Mar 21 16:27:29 2020 +0530

    Add files via upload
```

```
commit 595985919456d2d005a4a0dd45a66eb28c5e058a
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Sat Mar 21 16:26:33 2020 +0530

    Add files via upload

commit 3342be285d4235934cf74109f9a1b8bb0278263c
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Sat Mar 21 16:24:58 2020 +0530

    Add files via upload

commit 62759fca7c37851a0a2afe70bfb02fe1a6d5e383
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Sat Mar 21 16:24:40 2020 +0530

    Add files via upload

commit 979ab0df424ada6b499ff6ad47f68b2c106ecd84
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Sat Mar 21 15:45:26 2020 +0530

    Add files via upload

commit c0747bdcb877b229d3dae724e87fc9e8b502676e
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Sat Mar 21 15:45:03 2020 +0530

    Add files via upload

commit e5b17a32e9492fcbd519d37ab41df0a2f210733e
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Sat Mar 21 14:52:26 2020 +0530

    Add files via upload

commit d39f2e14c4c32de03b4b201d6358050445f31e53
Author: Vidit
```

```
<54926937+vidit04@users.noreply.github.com>
Date:   Sat Mar 21 14:18:00 2020 +0530

    Add files via upload

commit 1c4dbe2fb262285d704229fb16ba80ec3ef5fae8
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Sat Mar 21 14:15:06 2020 +0530

    Add files via upload

commit 26b18398d4a8a79406fdf1bde6240a6cf0102c3e
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Sat Mar 21 13:28:11 2020 +0530

    Add files via upload

commit b75d0f182f3f778fb2030c819c01407bd2b1a4a2
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Sat Mar 21 13:27:49 2020 +0530

    Add files via upload

commit b182942545a47d034cab90831c32eec77b16635b
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Sat Mar 21 13:27:13 2020 +0530

    Add files via upload

commit a6359958f443b8832db7fff3793f7ad41fbae5ee
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Wed Mar 18 17:29:53 2020 +0530

    Add files via upload

commit c9e07070e681babd65def5116cf172480c27678d
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Wed Mar 18 16:21:14 2020 +0530
```

```
    Add files via upload

commit e2688a1fde51d083b1829dfc6f8941ca2205206b
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Wed Mar 18 15:41:38 2020 +0530

    Add files via upload

commit 53f2cc43a95536addda46bde81b71742b5b37d54
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Wed Mar 18 11:06:07 2020 +0530

    Add files via upload

commit 62f88e3d47dc78c91b180c697151cdb9a1a064b4
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Wed Mar 18 10:39:51 2020 +0530

    Add files via upload

commit 54a947d773b715c2d0ceb399f50c26ce96ecd929
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Wed Mar 18 10:25:59 2020 +0530

    Add files via upload

commit f6f337f2c5511fc1199f234bb596844fb52daee3
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Wed Mar 18 10:17:20 2020 +0530

    Add files via upload

commit b0619bce120d6ba8b32a049c5b884c022b8be8dd
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Wed Mar 18 09:05:06 2020 +0530

    Add files via upload
```

```
commit 11deee492fc656797d8f0f99468fb0f418d84ff1
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Wed Mar 18 08:59:23 2020 +0530

    Add files via upload

commit 94c2c1e4abfb43c7bdf73a2c6ea9cabb3b3e747c
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Wed Mar 18 08:49:47 2020 +0530

    Add files via upload

commit 66aa20f237ce245f811d30130329493c008ece37
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Wed Mar 18 07:27:26 2020 +0530

    Add files via upload

commit 03988f8284bfabc69f90a25b0bf260deb753320c
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Wed Mar 18 07:26:25 2020 +0530

    Add files via upload

commit 141ae4b2b6ea79d1a9d135bb18851a716af4505c
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Mon Mar 16 12:03:30 2020 +0530

    Add files via upload

commit 9b97e791ef6d35d374f4bca9b7afe61dd7dded4e
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Mon Mar 16 07:31:08 2020 +0530

    Add files via upload

commit 14d7d5236b461e8bdfeff4c61d41b2eddfa39ef4
```

```
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Mon Mar 9 10:10:18 2020 +0530

    Add files via upload

commit a5581c756ccd1f34699cdeddabd34546aaceb23b
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Mon Mar 9 09:51:30 2020 +0530

    Add files via upload

commit 9992e68767adf805be3f8c71fa4604a8a92b809b
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Fri Mar 6 09:44:51 2020 +0530

    Add files via upload

commit cbddaf3da8739bac7b5fa30958595b78c69c45c5
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Fri Mar 6 09:28:43 2020 +0530

    Add files via upload

commit 79248a8cde4b7d855e6147db4abdff1c59cb5a06
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Fri Mar 6 06:15:58 2020 +0530

    Add files via upload

commit 6b6311956b18faa46eeb246b5ef5faa964257093
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Fri Mar 6 06:10:32 2020 +0530

    Add files via upload

commit 25ec018161edd2b83bce508941dc8b11e9cd333e
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
```

```
Date:   Thu Mar 5 12:21:54 2020 +0530

    Add files via upload

commit 7f3b1e3729b381f00af7f9bed5dad310c301b80a
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Mar 5 12:02:28 2020 +0530

    Add files via upload

commit dae8b8764e3baa8e137bc666c97bec6cb530eaae
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Mar 5 11:04:24 2020 +0530

    Add files via upload

commit 16fcee80484722bda22a6e21e18194a9879ac5b5
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Mar 5 10:35:22 2020 +0530

    Add files via upload

commit 160f794b64a4939172d835d98f430489384b538b
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Mar 5 08:28:14 2020 +0530

    Add files via upload

commit ac5101564ee943753a1ae6da2273ca703dfadfbc
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Mar 5 08:24:40 2020 +0530

    Add files via upload

commit fbed652a62220efcfe9bbd7ace18cd4720c645e6
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Mar 5 07:46:43 2020 +0530
```

```
    Add files via upload

commit 90e0dd76c80aa0aca98857ac980dbb7f36408493
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Mar 5 07:11:11 2020 +0530

    Add files via upload

commit 395405500f4bb073c5bf94dc39c7dabca244bac7
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Mar 5 06:01:48 2020 +0530

    Add files via upload

commit 31fed7cd56f96d2d8642d3e12560f619d6e3d489
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Mar 5 05:53:24 2020 +0530

    Add files via upload

commit de8dac034f184c4e50771698e6602c9890046a6f
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Mar 5 05:21:11 2020 +0530

    Add files via upload

commit 77d3d2fe62d3ff595799e7dd994318b9715fe308
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Mar 5 05:20:48 2020 +0530

    Add files via upload

commit 5c70712305b09ca339147c9d906ffc590fe140d8
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Mar 5 04:47:22 2020 +0530

    Add files via upload
```

```
commit 3000f036e98830f1abe6a4f8ecbaa8601579e0e8
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Mar 5 04:47:09 2020 +0530

    Add files via upload

commit 7145c2bd0d45a5b3415d2eaa53250ff9b9e60414
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Mar 5 04:38:14 2020 +0530

    Add files via upload

commit d1a445cf617b153d70e1731bf18226a5c746af98
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Mar 5 04:36:48 2020 +0530

    Add files via upload

commit 2e8317d08ed8ffbf72c3149e66998059cc120c69
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Mon Mar 2 09:57:35 2020 +0530

    Add files via upload

commit 8ab2997c8eade5becaab1a887bb284da10bb0afb
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Mon Mar 2 09:47:38 2020 +0530

    Add files via upload

commit aae498f0a847e4904ca99e2fdce866e230387b66
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Mon Mar 2 05:56:25 2020 +0530

    Add files via upload

commit 1179d25e257e595da7836af0899baeec4361744c
Author: Vidit
```

```
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Feb 20 08:22:09 2020 +0530

    Add files via upload

commit f04c8d9e2496ac05abdc19357b32256f0becef95
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Feb 20 05:42:04 2020 +0530

    Add files via upload

commit 675993b2932f9a5693a3a6fc170a47f5b040d165
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Feb 20 04:14:00 2020 +0530

    Add files via upload

commit e6538fc413d9cce1583bf73ea1f34e5f898f6703
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Feb 20 04:10:00 2020 +0530

    Delete train_1.py

commit 6ea4ec8809f2ba88301ae6fc8bb6774819f46c6b
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Feb 20 04:09:02 2020 +0530

    Add files via upload

commit 4c7abbb7352234a2c008abf0086208ec10d4e7e2
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Thu Feb 20 00:47:03 2020 +0530

    Add files via upload

commit 59272606fe29cf4f6fb1d36ef920f87e577886bc
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Tue Feb 18 23:59:02 2020 +0530
```

```
    Add files via upload

commit 58315db1b55fa0c3bd028185180b66a200e886c2
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Tue Feb 18 23:22:30 2020 +0530

    Add files via upload

commit 60a89ba66a63307b6b43c5cf5aa1ad9a4a83c945
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Wed Feb 12 03:51:23 2020 +0530

    Add files via upload

commit 5bde93e9c39f0d8ae9b8d05f19873be198f65c72
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Wed Feb 12 02:48:22 2020 +0530

    Add files via upload

commit f5b04e8315cb459be59cf0b5295b6e48ed4edc4d
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Wed Feb 12 01:02:08 2020 +0530

    Create test.py

commit 183aeab52d862b29e0d82a5a57dc058d5867249d
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Wed Feb 12 01:01:41 2020 +0530

    Add files via upload

commit ac994513486cd057aad52b802bbb8113e692f84c
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Wed Feb 12 00:59:34 2020 +0530

    Create train.py
```

```
commit dd857a5f28f83062673cbd02a255a4cba2e4b17f
Author: Vidit
<54926937+vidit04@users.noreply.github.com>
Date:   Wed Feb 12 00:54:40 2020 +0530

    Initial commit
```