بسمه تعالی

HW2- Deep Neural Networks

Supervisor:

Prof. Johari Majd

Student:

Morteza Bigdeli - 40261662001

Tarbiat Modares University

Spring 2024

Table of Contents

Table if Figures

1. Pen and Paper Exercises

a) Analytic Gradient Calculation

a) Analytic Gradient Calculation

i) ReLU $(\mathbf{x}) = \max(0, \mathbf{x}), \mathbf{x} \in \mathbb{R}$

ii) ELU $(\mathrm{x}) = \begin{cases} \mathrm{x} & \mathrm{x} \geq 0 \\ c(e^{\mathbf{x}} - 1) & \mathrm{x} < 0 \end{cases}, \mathbf{x} \in \mathbb{R}$ and c=constant.

iii) Tanh $(\mathbf{x}) = \frac{2}{1+e^{-2\mathbf{x}}} - 1, \mathbf{x} \in \mathbb{R}$

iv) Softmax $(\mathbf{x}) = \frac{e^{\mathbf{x}}}{\sum_{i=1}^{n} e^{x_i}}, \mathbf{x} \in \mathbb{R}^n$

i. For the ReLU function, the gradient is:

$$\frac{\partial}{\partial x}\mathrm{ReLU}(x) = \begin{cases} 1, & \text{if } x > 0, \\ 0, & \text{if } x \leq 0. \end{cases}$$

ii. For the ELU function, the gradient is:

$$\frac{\partial}{\partial x}\mathrm{ELU}(x) = \begin{cases} 1, & \text{if } x \geq 0, \\ ce^{x}, & \text{if } x < 0. \end{cases}$$

iii. For the Tanh function, the gradient is:

$$\frac{\partial}{\partial x}\tanh(x) = 1 - (\tanh(x))^2 = 1 - \left(\frac{2}{1+e^{-2x}} - 1\right)^2.$$

iv. For the Softmax function, the gradient with respect to x_i is:

$$\frac{\partial}{\partial x_j}\mathrm{Softmax}(x)_i = \begin{cases} \mathrm{Softmax}(x)_i(1 - \mathrm{Softmax}(x)_i), & \text{if } i = j, \\ -\mathrm{Softmax}(x)_i\,\mathrm{Softmax}(x)_j, & \text{if } i \neq j. \end{cases}$$

v) *Given* the predicted probability $\mathbf{p} \in \mathbb{R}^n$ and ground-truth label $\mathbf{y} \in \mathbb{R}^n$, calculate the gradient of cross entropy loss $H(\mathbf{y}, \mathbf{p}) = -\sum_{i=1}^{n} y_i \log p_i$ wrt. every element of $\mathbf{p}$, i.e. $\frac{\partial H}{\partial p_i}(\mathbf{y}, \mathbf{p})$.

$$\frac{\partial H}{\partial p_i}(\mathbf{y}, \mathbf{p}) = -\frac{y_i}{p_i}.$$

b) Prove

Prove that: Softmax ($\mathbf{x}$) = Softmax ($\mathbf{x}$ + c), where c is a constant.

$$\mathrm{Softmax}(x+c)_i = \frac{e^{x_i+c}}{\sum_{k=1}^{n} e^{x_k+c}} = \frac{e^{x_i}e^{c}}{e^{c}\sum_{k=1}^{n} e^{x_k}} = \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}} = \mathrm{Softmax}(x)_i.$$

c) Numerical Calculation

c) Consider a one-layer neural network $\mathbf{y} = g(\mathbf{AX})$ with input $\mathbf{X}$, output $\mathbf{y}$, network weight $\mathbf{A}$ and output function g. Let's first assume the input and the network weights are

$$\mathbf{A} = \begin{pmatrix} 3.0 & 2.0 \end{pmatrix} \in \mathbb{R}^{1\times 2} \quad \mathbf{X} = \begin{pmatrix} 2.0 & 1.0 & -1.0 \\ 4.0 & -2.0 & 0.0 \end{pmatrix} \in \mathbb{R}^{2\times 3}$$

where $\mathbf{x}$ are 2D points with a minibatch size of 3 .

Ex C.

i) $A = [3 \quad 2]$ $\quad X = \begin{bmatrix} 2 & 1 & -1 \\ 4 & -2 & 0 \end{bmatrix}$

$t = [15 \quad 3 \quad 1]$ $\quad L = 1/2 \sum_j (y_j - t_j)^2$

forward pass:

$$y = AX = [3 \quad 2] \begin{bmatrix} 2 & +1 & -1 \\ 4 & -2 & 0 \end{bmatrix}$$

$\Rightarrow y = [14 \quad -1 \quad -3]$

the loss before the update:

$$L = 1/2 \left[(14-15)^2 + (-1-3)^2 + (-3-1)^2 \right] = 1/2 [1 + 16 + 16] = \underline{16.5}$$

Gradients: $y = a_1 x_j + a_2 x_j$

$$\frac{\partial L}{\partial a_1} = \frac{\partial L}{\partial a_1} \cdot \frac{\partial y_1}{\partial a_1} + \frac{\partial L}{\partial y_2} \cdot \frac{\partial y_2}{\partial a_1} + \frac{\partial L}{\partial y_3} \frac{\partial y_3}{\partial a_1}$$

$$\Rightarrow \begin{cases} y_1 = a_1 x_{11} + a_2 x_{21} \\ y_2 = a_1 x_{12} + a_2 x_{22} \\ y_3 = a_1 x_{13} + a_2 x_{23} \end{cases}$$

$$\Rightarrow \frac{\partial L}{\partial a_1} = (y_1 - t_1) x_{11} + (y_2 - t_2) x_{12} + (y_3 - t_3) x_{13}$$

$$\Rightarrow \frac{\partial L}{\partial a_1} = (14-15) \times 2 + (3-(-1)) \times 1 + (1-(-3)) \times -1 \; \underline{= 2}$$

$$\frac{\partial L}{\partial a_2} = \frac{\partial L}{\partial y_1} \cdot \frac{\partial y_1}{\partial a_2} + \frac{\partial L}{\partial y_2} \cdot \frac{\partial y_2}{\partial a_2} + \frac{\partial L}{\partial y_3} \cdot \frac{\partial y_3}{\partial a_2} =$$

$$(14-15)m_{21} + (3-(-1))m_{21} + (1-(-3))m_{23} = 4 - 4\times 2 + 4\times 0$$

$$\boxed{= -4}$$

Considering the learning rate 1, the update will be:

$A_N = A_{old}$ - Learning rate × gradient

$$\Rightarrow A_N = \begin{pmatrix} 3 \\ 2 \end{pmatrix} - 1\times\begin{pmatrix} 2 \\ -4 \end{pmatrix} = \begin{bmatrix} 1 \\ 6 \end{bmatrix}$$

the forward pass will be as follows:

$$Y_{new} = \begin{bmatrix} 1 \\ 6 \end{bmatrix} \begin{bmatrix} 2 & -1 & -1 \\ 4 & -2 & 0 \end{bmatrix} = \begin{bmatrix} 26 & -11 & -1 \end{bmatrix}$$

$$\Rightarrow L_{new} = \frac{1}{2}\left[(26-15)^2 + (-11-3)^2 + (-1-1)^2\right] =$$

$$\frac{1}{2}\left[121 + 196 + 4\right] = \boxed{160.5}$$

---

i) It is obvious that the loss is gained worse from first step after update, the reason is the Learning rate which has been considered high and does not lead to convergence.

---

iii) $y = g(Ax) = \text{relu}(Ax)$

$$\frac{\partial L}{\partial a_i} = \frac{\partial L}{\partial y} \cdot \frac{\partial g}{\partial a_i} \rightsquigarrow \begin{cases} \dfrac{\partial g}{\partial a_i} = \dfrac{\partial (Ax)}{\partial a_i} = \begin{cases} \neq 0 & \text{if } y_i > 0 \\ = 0 & \text{if } y_i = 0 \end{cases} \\ \dfrac{\partial L}{\partial y} = \dfrac{\partial L}{\partial y} = (y_i - t_i) \end{cases}$$

$$\Longrightarrow \begin{cases} y_1 = \text{relu}(a_1 x_{11} + a_2 x_{21}) \\ y_2 = \text{relu}(a_1 x_{12} + a_2 x_{22}) \\ y_3 = \text{relu}(a_1 x_{13} + a_2 x_{23}) \end{cases} \Longrightarrow \begin{cases} y_1 = a_1 x_1 + a_2 x_{21} = 14 \\ y_2 = 0 \\ y_3 = 0 \end{cases} \underline{\text{considering relu}}$$

$$\frac{\partial L}{\partial a_1} = \underbrace{\frac{\partial L}{\partial y_1} \cdot \frac{\partial y_1}{\partial a_1}}_{} + \overset{0}{\cancel{\frac{\partial L}{\partial y_2} \cdot \frac{\partial y_2}{\partial a_1}}} + \overset{0}{\cancel{\frac{\partial L_3}{\partial y_3} \cdot \frac{\partial y_3}{\partial a_1}}}$$

$$\Longrightarrow \begin{cases} \dfrac{\partial L}{\partial a_1} = (y_1 - t_1) x_{11} = (14 - 15) 2 = \boxed{-2} \\ \dfrac{\partial L}{\partial a_2} = \dfrac{\partial L}{\partial y_1} \cdot \dfrac{\partial y_1}{\partial a_2} + \overset{0}{\cancel{\dfrac{\partial L}{\partial y_2} \cdot \dfrac{\partial y_2}{\partial a_2}}} + \overset{0}{\cancel{\dfrac{\partial L}{\partial y_3} \cdot \dfrac{\partial y_3}{\partial a_2}}} = \boxed{-4} \end{cases}$$

$A_{new} = A_{old} - \text{Learning\_rate} \times \text{gradients}$

$$\Longrightarrow A_{new} = \begin{bmatrix} 3 \\ 2 \end{bmatrix} - 1 \begin{bmatrix} -2 \\ -4 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$$

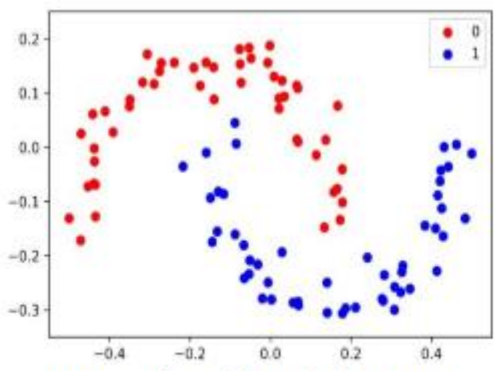
$$Y_{new} = \text{relu}\left[[5 \;\; 6] \begin{bmatrix} 2 & 1 & -1 \\ 4 & -2 & 0 \end{bmatrix} \right] = [54 \;\; 0 \;\; 0]$$

$$L_{new} = \frac{1}{2} \left[(34-15)^2 + (0-3)^2 + (0-1)^2 \right] = \frac{1}{2} [361 + 9 + 1]$$

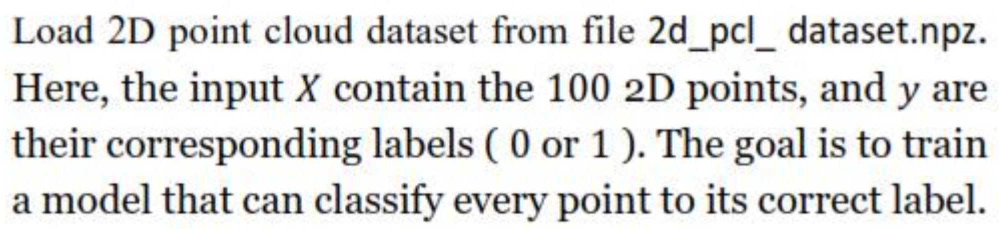
$$\boxed{= 185.5}$$

Github Adress for the codes: https://github.com/Mbigdeli2003/Deep-Learning/blob/main/DNN_HW2_Q2_Morteza_Bigdeli.ipynb

2. Binary Classification on 2D Point Cloud



2D point cloud dataset



Load 2D point cloud dataset from file 2d_pcl_ dataset.npz. Here, the input X contain the 100 2D points, and y are their corresponding labels (0 or 1). The goal is to train a model that can classify every point to its correct label.

2-1- Logistic Regression (a and b task)

As the first task, train a logistic regression model with CrossEntropyLoss. You can reuse use your code from Exercise 1 and visualize the boundary decision.

Bu using CrossEntropyLoss and train with logistic regression considering 100000 epochs and the learning rate of 0.001 for the data, the training loss will be converged to about 0.2, (Fig1), the decision boundary is also shown in Fig2 which is obviously is line that cannot specify the boundaries of the data well enough.

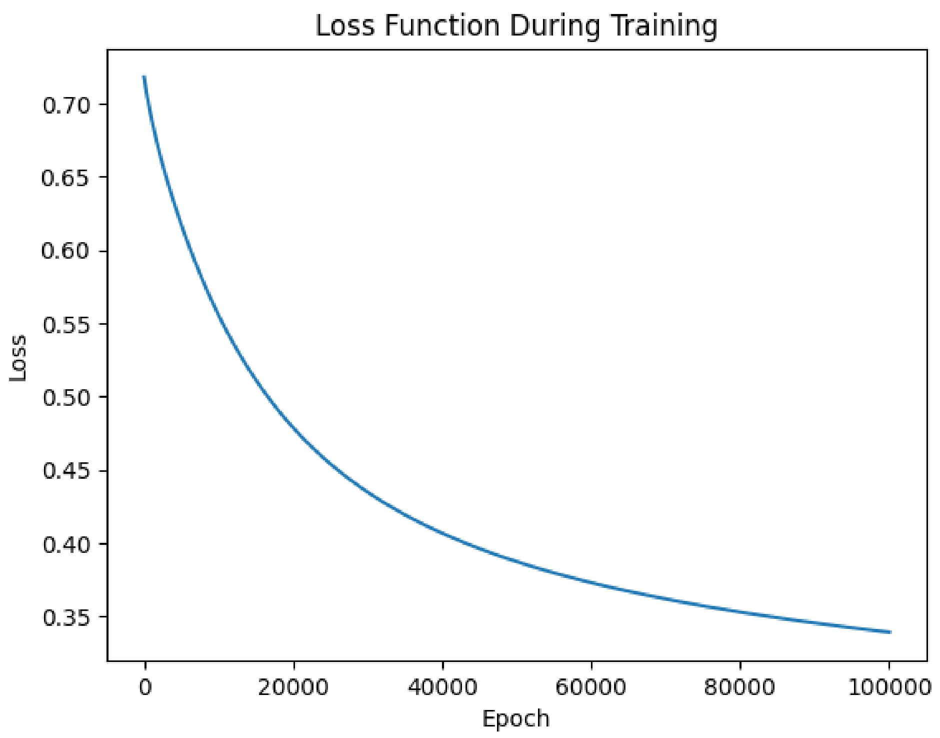


Figure 1.Cross Entropy Loss for Logistic Regression (2D point cloud data)

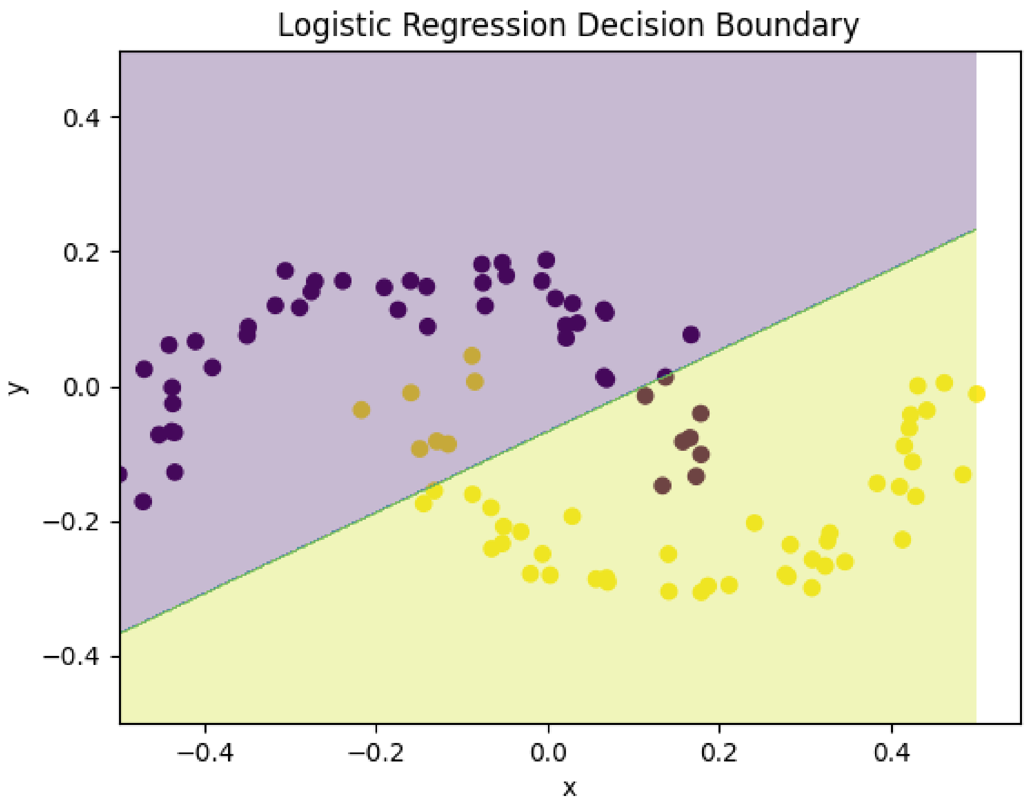


Figure 2. Decision Boundary for Logistic Regression (2D point cloud data)

2-2- MLP (c and d task)

In the MLP section, the hyper parameters are tuned and set to learning rate of 0.01, ADAM optimizer, 16 neurons in hidden layer with sigmoid activation function for hidden layer and output, and finally the code has been run with 100000 epochs.

Fig 3. Show the loss during iterations, with thanks to ADAM optimizer the loss with a momentum is converged to about zero. Fig 4. Indicates the bended boundary decision which has great accuracy in separating the data.

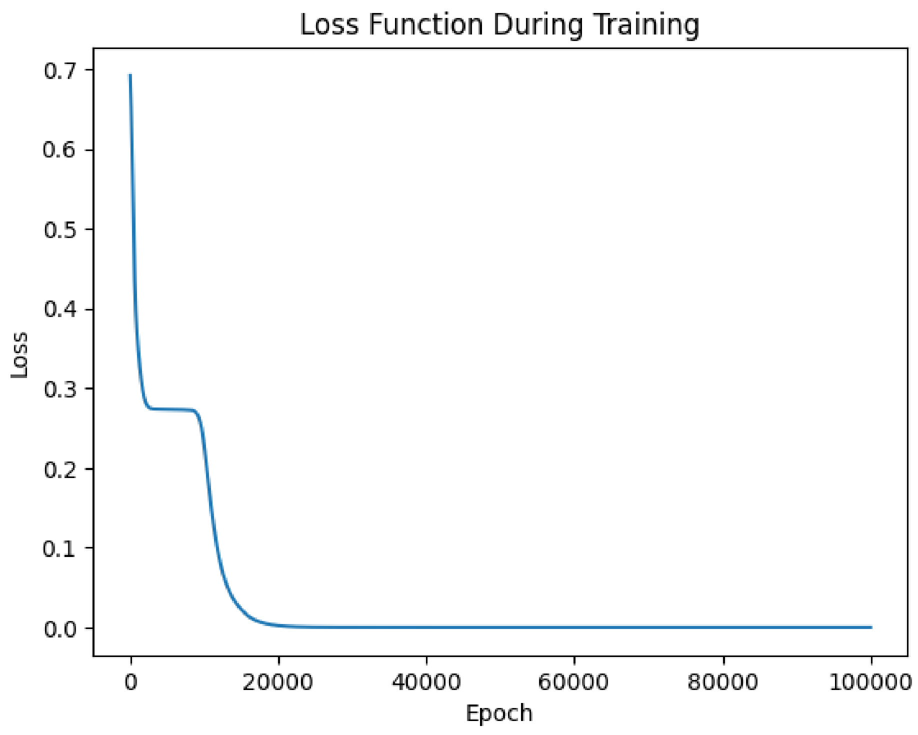


Figure 3. BCE Loss for MLP (2D point cloud data)

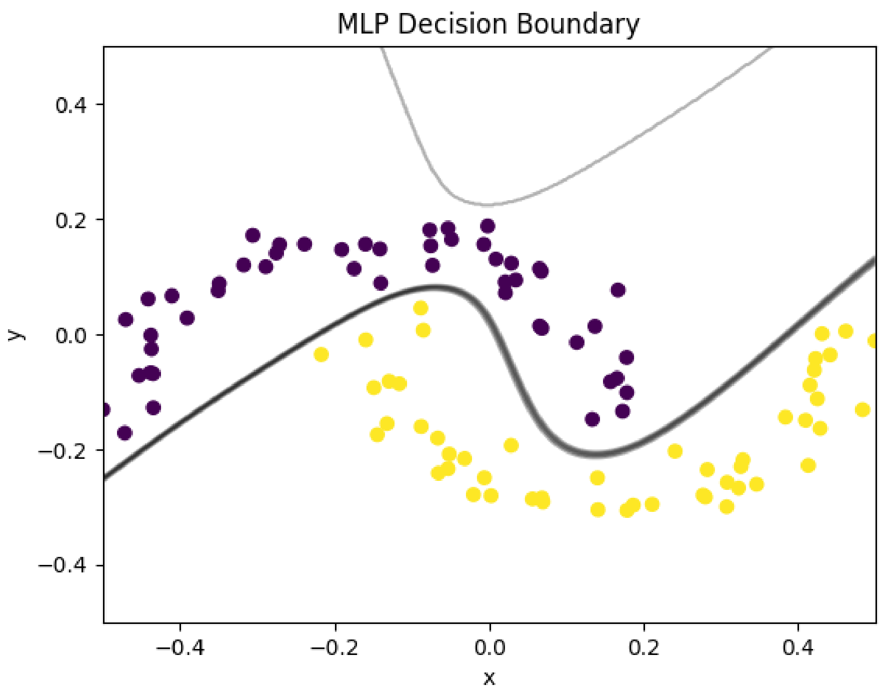
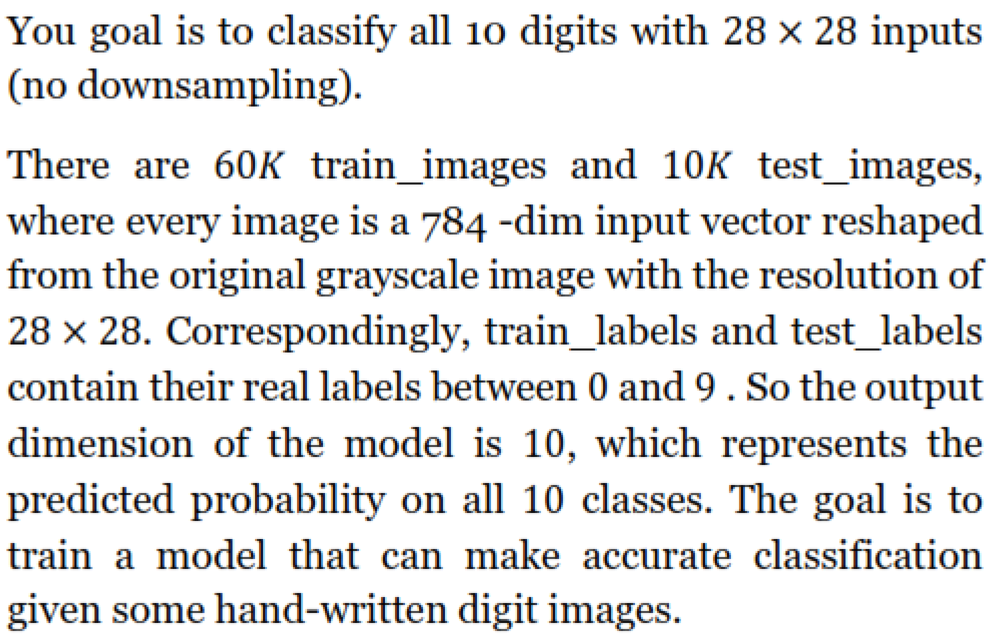


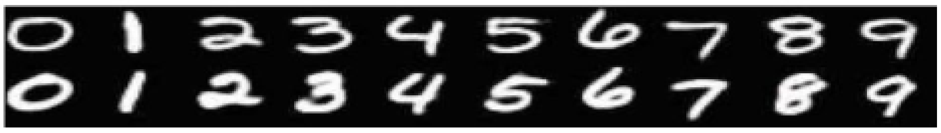
Figure 4. Decision Boundary for MLP (2D point cloud data)

Github Adress for he codes: https://github.com/Mbigdeli2003/Deep-Learning/blob/main/DNN_HW2_Q3_Morteza_Bigdeli.ipynb

3. Image Classification on MNIST

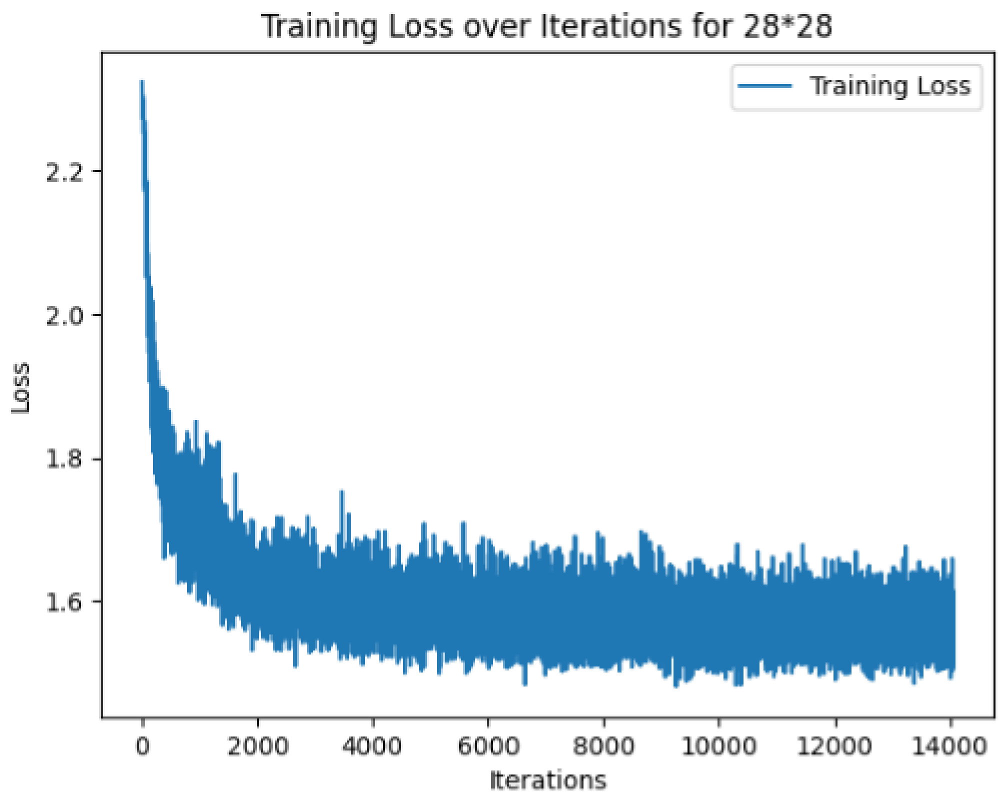

You goal is to classify all 10 digits with 28×28 inputs (no downsampling).

There are $60K$ train_images and $10K$ test_images, where every image is a 784 -dim input vector reshaped from the original grayscale image with the resolution of 28×28. Correspondingly, train_labels and test_labels contain their real labels between 0 and 9 . So the output dimension of the model is 10, which represents the predicted probability on all 10 classes. The goal is to train a model that can make accurate classification given some hand-written digit images.



a) Logistic Regression

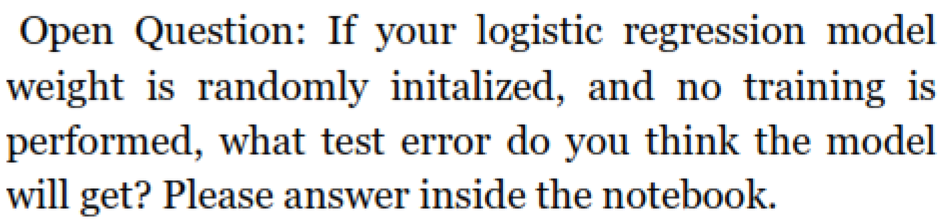
With considering this method with hyper parameters as learning rate of 0.01 and 15 epochs the loss during iterations with crossentropyloss is converged to 1.6 and the test error is obtained 8.1% with model evaluation using the test data of mnist (Fig 5.)



```
Test Error for 28*28: %8.07
```

Figure 5. Training Loss for Logistic Regression During Iterations

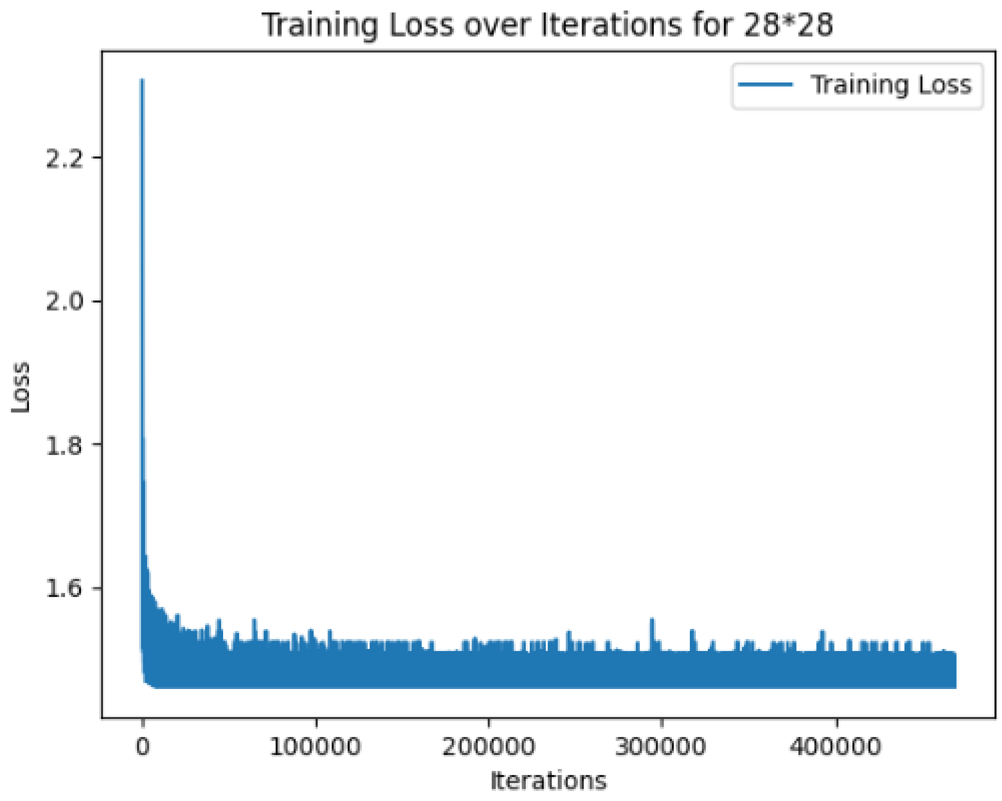
b) Open Question


Open Question: If your logistic regression model weight is randomly initalized, and no training is performed, what test error do you think the model will get? Please answer inside the notebook.

if a logistic regression model's weights are randomly initialized and no training is performed, the model will not be able to make accurate predictions. Without training, the model will essentially be making random guesses, resulting in high test error. The test error in this scenario would likely be close to 0.5, as the model would be no better than random chance at predicting the outcome. It is essential to train the model on labeled data to optimize the weights and improve its predictive performance.

c) Two-layer MLP with 64 Sigmoid neurons in the hidden layer and Softmax for the output.

Considering the defined model in this part, the hyper parameter are set with the learning rate 0f 0.001, one hidden layer with sigmoid activation function, ADAM optimizer is used and the trainig is considered 500 epochs. The Fig 6. Show crossentropyloss during iterations which is converged to less that 1.4 at last. The test error is obtained 3.26% in model evaluation which is acceptable enough to this data.



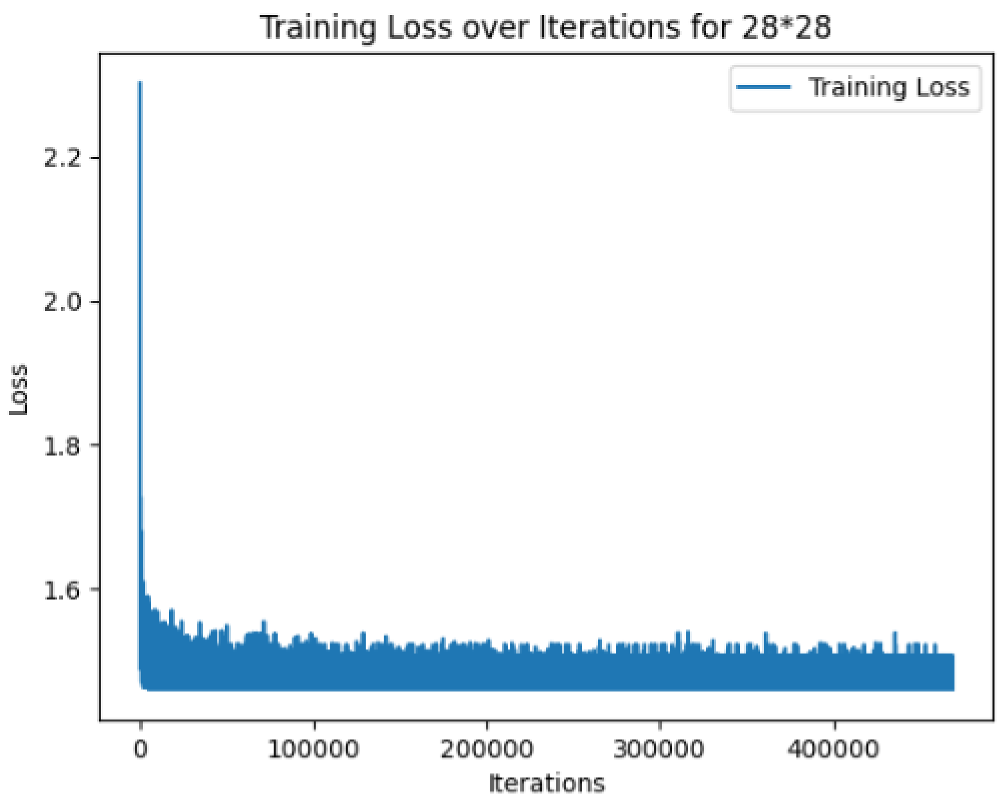
```
Test Error for 28*28: %3.2699999999999995
```

Figure 6. Training Loss for MLP During Iterations with 64 neurons (Sigmoid)

Two-layer MLP with 64 neurons and different activation function in hidden layer.

i. Tanh

In this part the hyper parameter are considered similar to c section. Fig 7. shows the loss during iterations which is converged to less than 1.4 and test error is obtained 3.14% which is a little bit better than sigmoid function. Due to similarity of sigmoid and Tanh the result approve the expected outcomes.

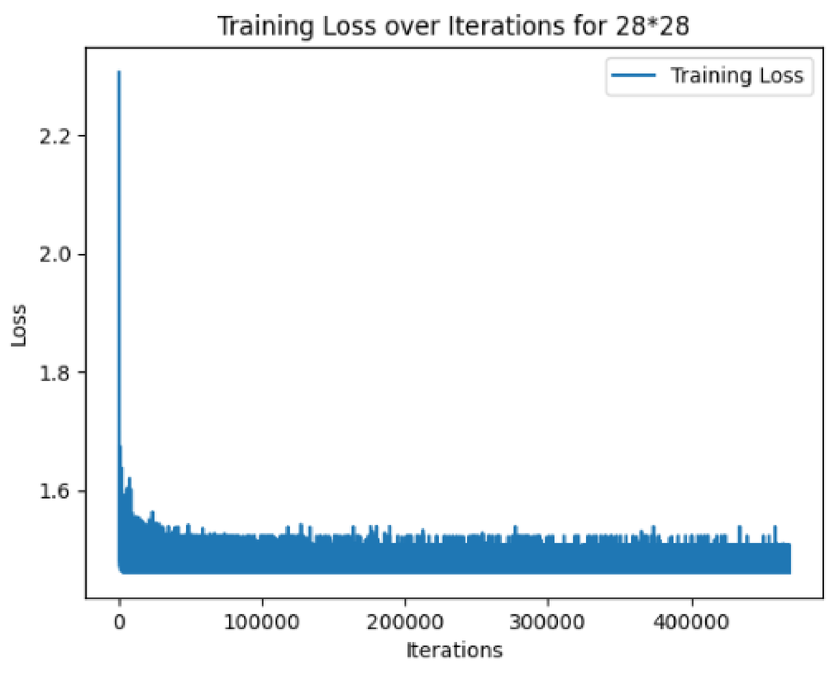


```
Test Error for 28*28: %3.1499999999999972
```

Figure 7. Training Loss for MLP During Iterations with 64 neurons (Tanh)

ii. ReLU

In this part ReLU activation function is considered for the hidden layer. The hyper parameters are similar to last part, ADAM optimizer has been applied to the model. Fig 8. shows the training loss during iterations which has been obtained less than 1.2 and the test error is 2.7% which is better than Tanh and Sigmoid activation function for the hidden layer.

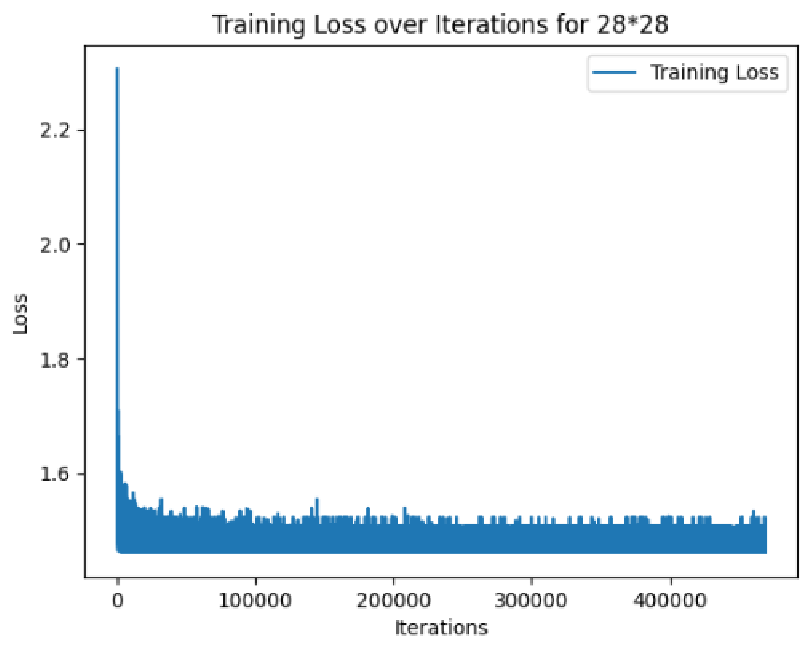


```
Test Error for 28*28: %2.7000000000000024
```

Figure 8. Training Loss for MLP During Iterations with 64 neurons (ReLU)

iii. Leaky ReLU with c=0.01

Fig 9. shows the output of the loss during iterations which is about the same with last part using ReLU. Hyper parameter has been considered the same as before and the test error has been obtained 2.23% which is better than last parts.



```
Test Error for 28*28: %2.2399999999999975
```

Figure 9. Training Loss for MLP During Iterations with 64 neurons (Leaky ReLU)

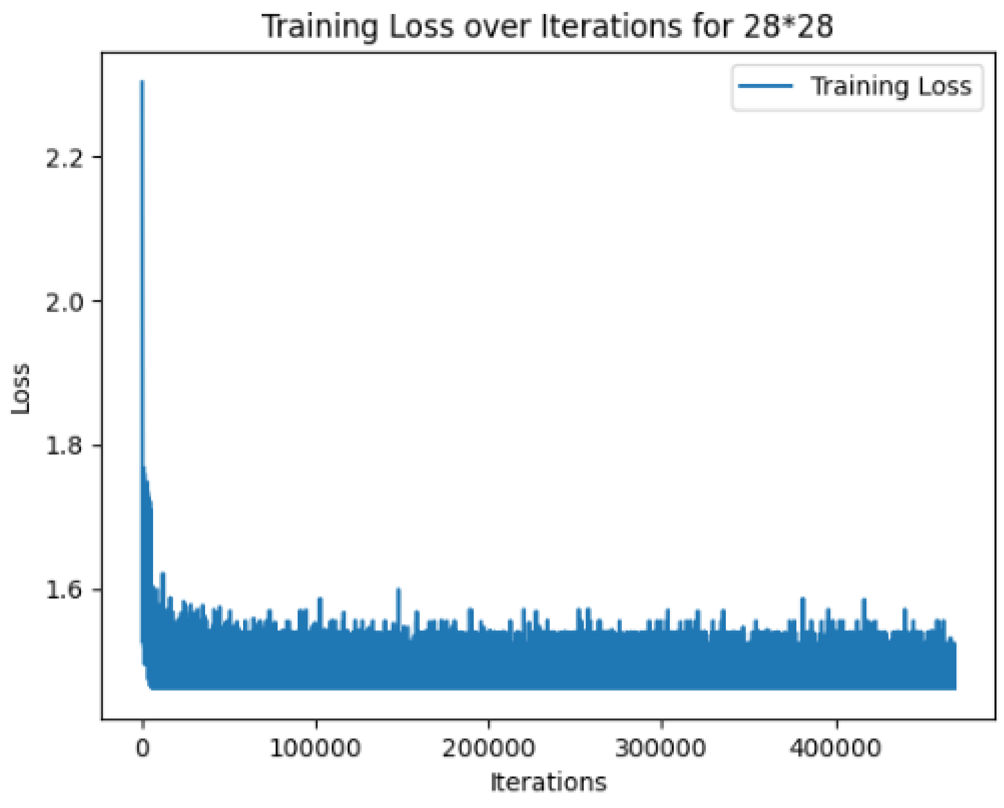
d) Best Learning Rate for the ReLU activation function

In part c-ii the several learning rates were tested and the best one which is 0.001 has been applied to the model. The loss and the test error in that part are the best ones which could be achieved by training with those hyper parameters.

e) Three Layer MLP with 32 neurons in second layer and 24 in third layer

i. ReLU

The hyper parameters are considered the same as before, Fig 10. Indicates that the loss during iterations which is converged to less than 1.5 and the test error is 3.24%.

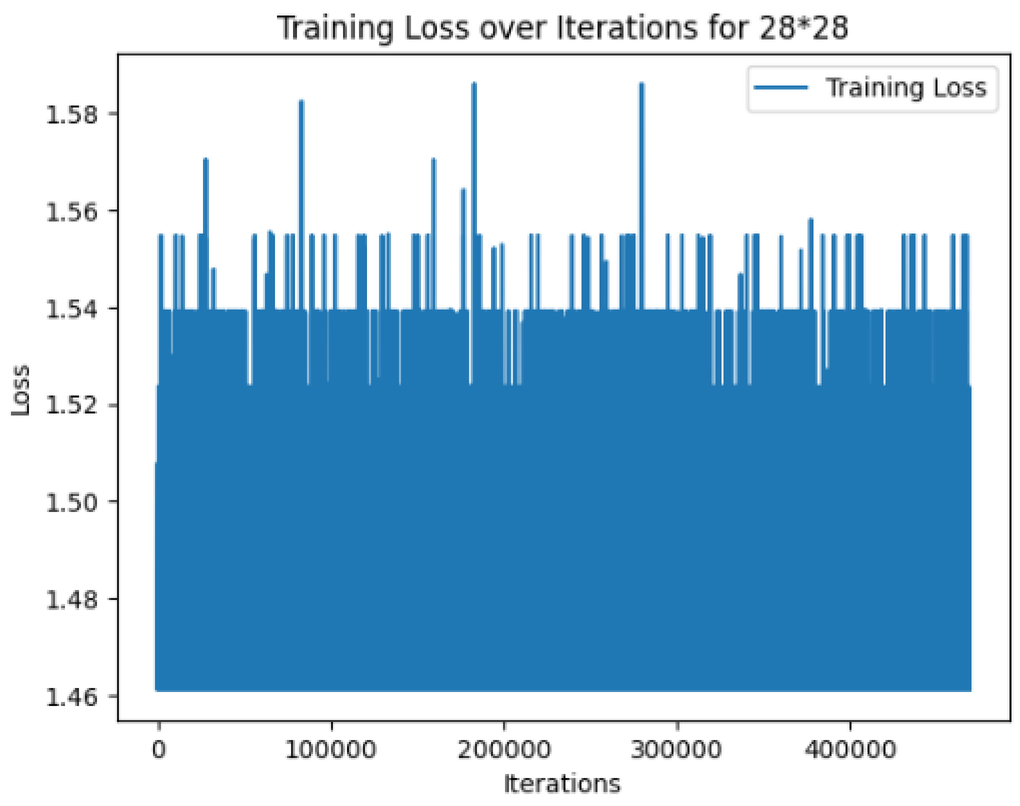


```
Test Error for 28*28: %3.2499999999999973
```

Figure 10. Training Loss for MLP During Iterations with 32 and 24 neurons (ReLU)

ii. Sigmoid

Fig 11. shows the loss during iterations. Due to the sigmoid activation function the loss is fluctuating between 1.58 to 1.4 in all iterations, however, the test error is about 3.01% which is acceptable for this training.



```
Test Error for 28*28: %3.0100000000000016
```

Figure 11. Training Loss for MLP During Iterations with 32 and 24 neurons (Sigmoid)

Comparing these results with last section using one hidden layer, it can concluded that adding hidden layers might not reduce the test error in some data training, moreover, it is just adding the calculation time and training. It can also be seen even at some level, adding one hidden layer may lead to worse test error and training (considering ReLU in last section result with the three layer MLP in this part with same activation function.)

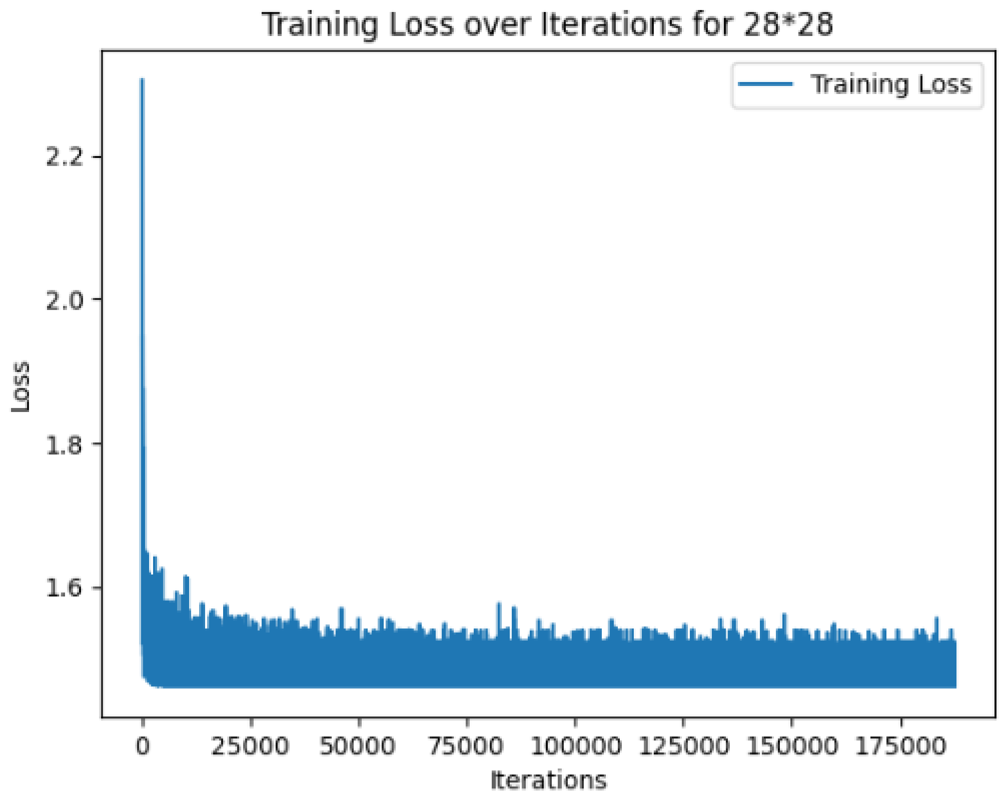
f) One hidden layer 32, 64, and 128 neurons

In this section 32, 64, and 128 neurons with ReLU and sigmoid activation function has been considered for training, the learning rate is set 0.001 and

ADAM optimizer used for training. The test errors is compared after representing all training parts.

i. 32 neurons with ReLU

Fig 12. shows the training loss which is converged to less than 1.4 and the test is 3.32% for this training.

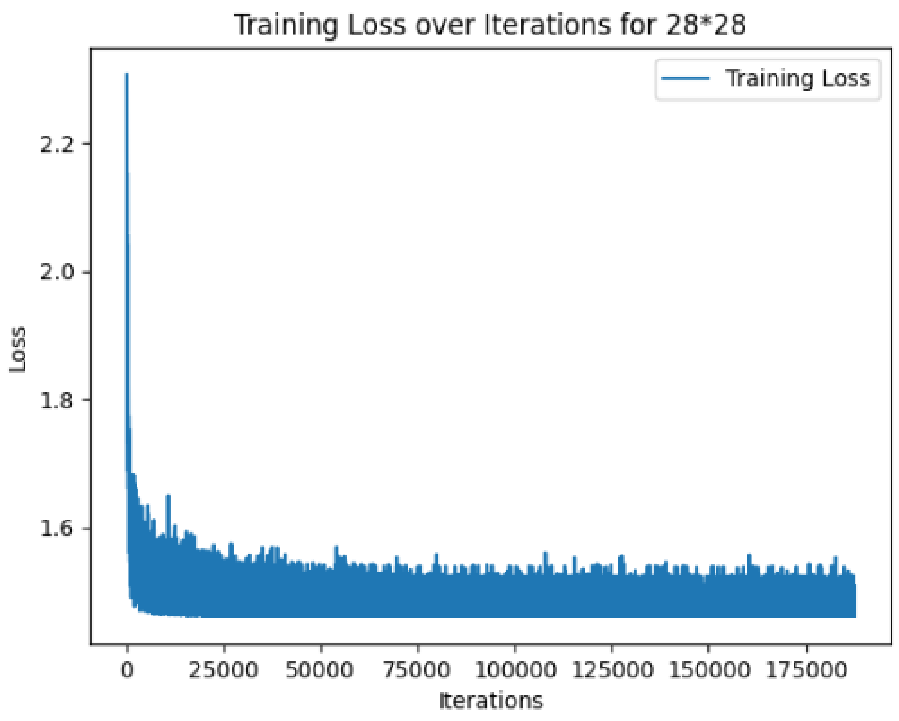


```
Test Error for 28*28: %3.3299999999999996
```

Figure 12. Training Loss for MLP During Iterations with 32 neurons (ReLU)

ii. 32 neurons with Sigmoid

Fig 13. shows the training loss during iterations which is not very different than last part, however, using Sigmoid activation function in hidden layer led the test error to 4.44%.

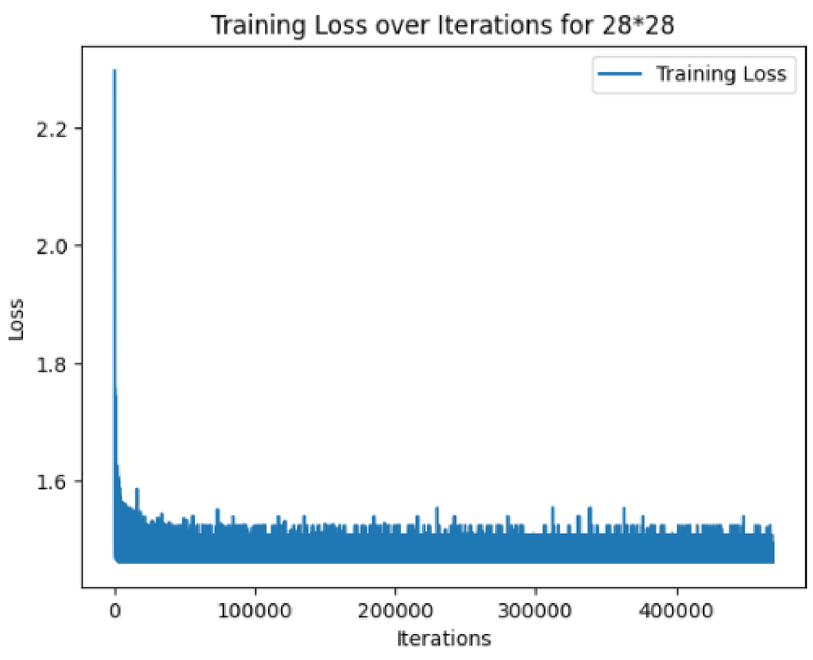


```
Test Error for 28*28: %4.449999999999998
```

Figure 13. Training Loss for MLP During Iterations with 32 neurons (Sigmoid)

iii. 64 neurons with ReLU

Using ReLU activation function and adding neurons to hidden layer result better performance in training which is shown in Fig14. Moreover, the test error is also reduced to 2.39%.

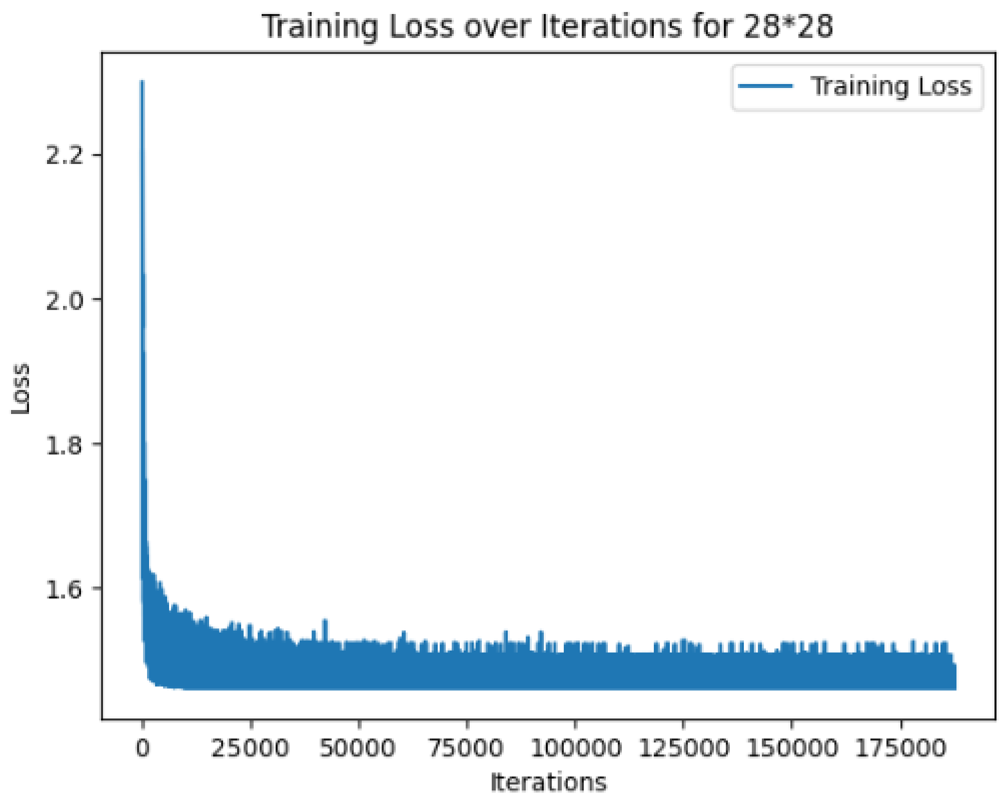


```
Test Error for 28*28: %2.3900000000000032
```

Figure 14. Training Loss for MLP During Iterations with 64 neurons (ReLU)

iv. 64 neurons with Sigmoid

Fig 15. indicates the training loss during iterations which is little bit more than iii as it was expected, it can also be seen that test error is increased to 3.26% using sigmoid activation function in hidden layer.

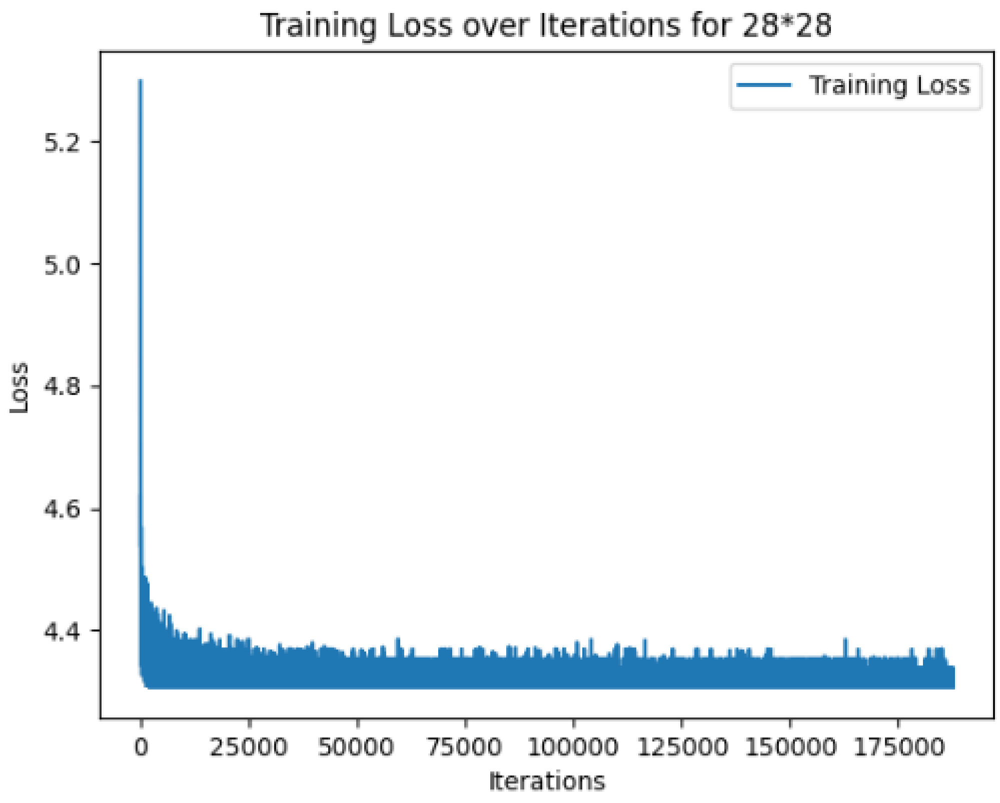


```
Test Error for 28*28: %3.269999999999995
```

Figure 15. Training Loss for MLP During Iterations with 64 neurons (Sigmoid)

v. 128 neurons with ReLU

The training loss by adding neurons has been decreased compared to i and iii, and the test error by considering 128 neurons in hidden layer and Relu activation function is obtained 2.14% which is the best value in all training parts. Fig 16. shows the result which is indicating the training loss is higher at first iterations than the 32 and 64 neurons training, but the test error is better than any other considered nets.

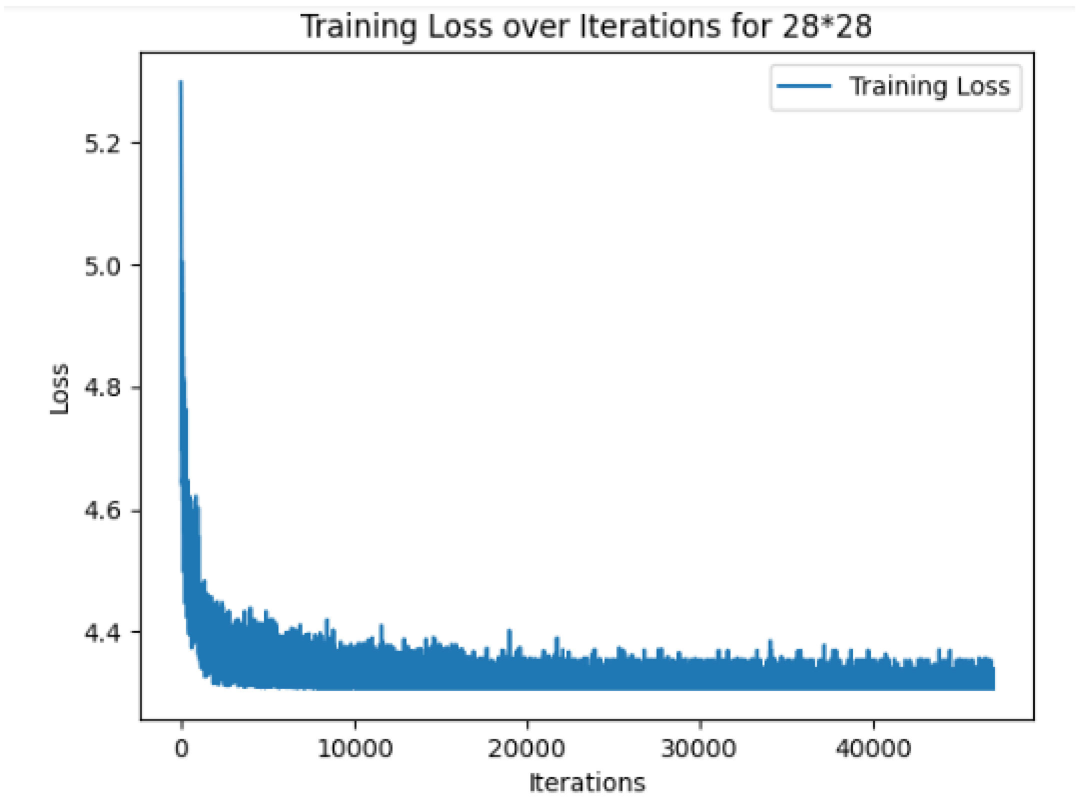


```
Test Error for 28*28: %2.1499999999999964
```

Figure 16. Training Loss for MLP During Iterations with 128 neurons (ReLU)

vi. 128 neurons with Sigmoid

The last but not the least in this training section is 128 neurons which is using sigmoid activation function in the training. Considering Fig 17. the training loss is again higher than 32 and 64 neurons at first, but it finally converged to last section values, it is obvious that the test error is less than last section and the value is 2.7%.



```
Test Error for 28*28: %2.7000000000000024
```

Figure 17. Training Loss for MLP During Iterations with 128 neurons (Sigmoid)

Comparing the f part results

Fig 18. Indicates the test errors using 32, 64, 128 neurons with considering ReLU as activation function in hidden layer. Obviously the test error is less in MLP using 128 neurons. Fig 19. is also showing the test errors considering sigmoid activation function which shows the test errors reduced by adding neurons to hidden layer. It should also be noted that using ReLU activation function obtained less test error than using Sigmoid in hidden layer.

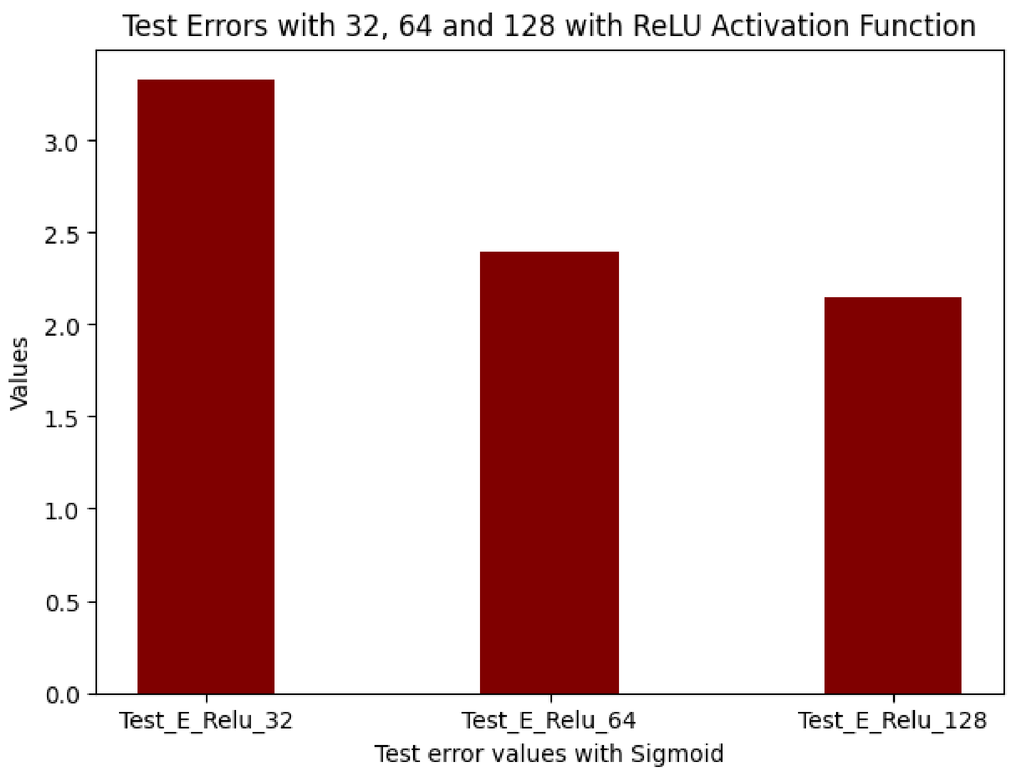


Figure 18. test errors using 32, 64, 128 neurons with considering ReLU as activation function

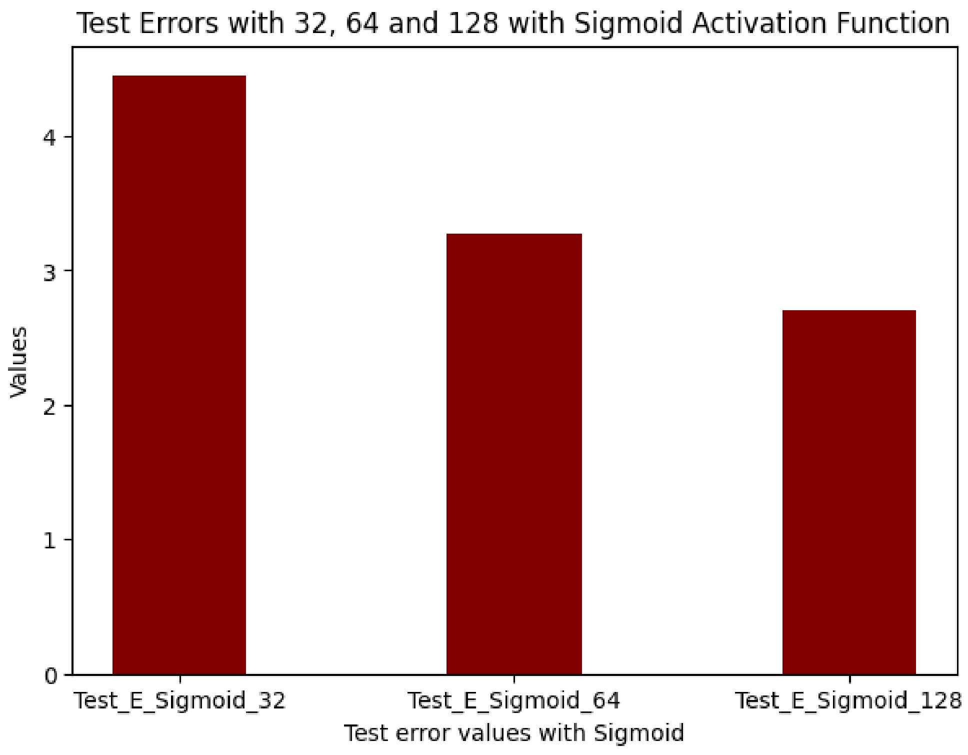


Figure 19. test errors using 32, 64, 128 neurons with considering Sigmoid as activation function