CSE641 - Deep Learning

Assignment 1

- Abhimanyu Gupta (2019226)
- Meenal Gurbaxani (2019434)

Contribution

Abhimanyu Gupta - Part 2
Meenal Gurbaxani - Part 1

Part 1

1)

Methodology:

x - given sample
y - true classes for x
y' - predicted classes for given x
The perceptron training algorithm is implemented using NumPy to update weights wherever misclassification (y' ≠ y) is made.
Weight update rule used -

$$w = w + y * x$$

To plot the decision boundary, its intercepts on x and y axis are calculated by individually putting x_1 and x_2 to be zero in the equation $w_1 * x_1 + w_2 * x_2 + w_3 * 1 = 0$ ($w^T * x = 0$).

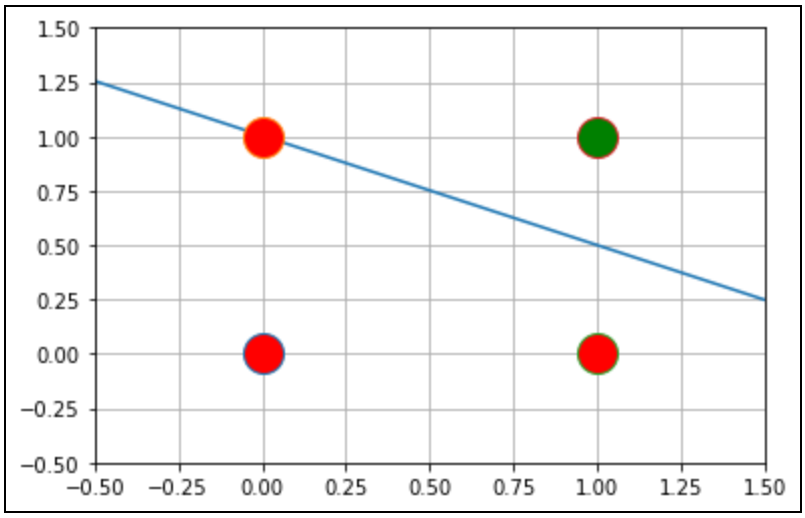
To demonstrate that PTA wouldn't converge we need PTA to run for more than 1000 (loose bound) as among 'and', 'or' and 'not', maximum of 10 steps were taken thus, 10^2 would be quite safe to predict if PTA would converge on given data

Observations:

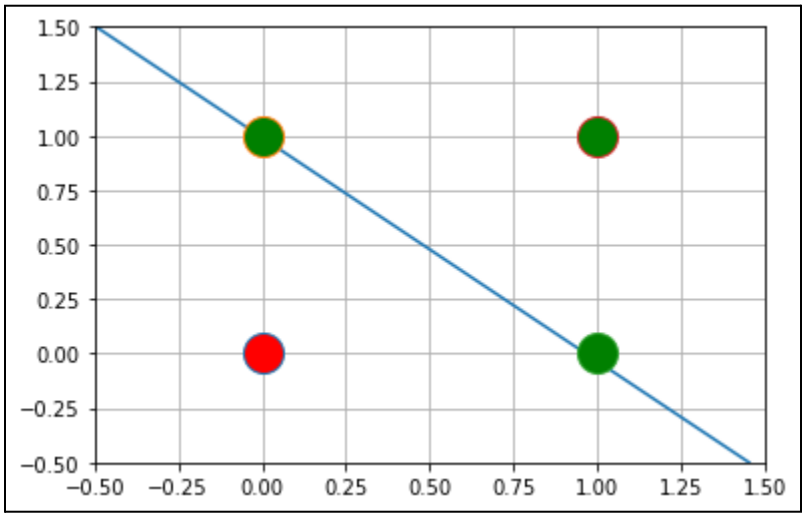
➔ PTA can only converge on linearly separable function and so thus for And, Or and Not; but doesn't for non-linearly separable function XOR.

Outputs:

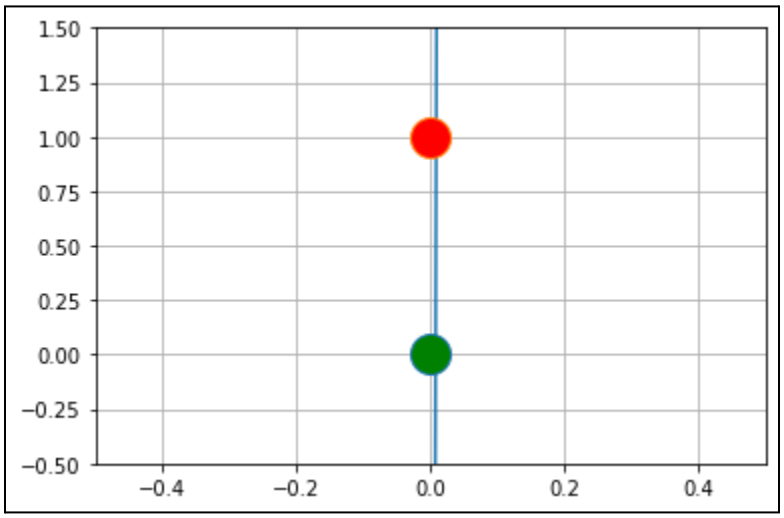
- And(x1, x2, y): It took about 10 steps to converge for model



- Or(x1, x2, y): It took about 5 iterations to converge for model



- Not(x, y): It took about 2 iterations to converge for model



2)

Methodology:

- ➔ Class Madeleine was implemented for purpose of computing the following function.
- ➔ To handle the given shape the squares of size 2 by 2 (whether coloured or not) were replaced by a single point present at the centre of those squares, and then trained through Madeleine
- ➔ For instance, the square defined by vertexes (4, 0), (6, 0), (4, 2) and (6, 2) was replaced by a single point at (5, 1).

Observations:

a) At least 4 neurons are required to train the data.

b) Since the data is not linearly separable so a 2 neuron based Model will fail to model the problem.

Additionally, the Madeleine Learning algorithm fails to train with 2 neurons as can be seen. The accuracy is consistent and unchanging which means that it is the default value of the weights itself and not update

As a result, we would need at least 4 neurons

Part 2

Methodology:

- **Preprocessing**
  - The input images were already in the form of a 1d list so no image flattening was needed.
  - But, the pixel values were in the range of 0 to 255, hence StandardScaler from sklearn was used to Standardardizing the input for Gaussian Distribution.
- **Learning**
  - Class MultiLayerPerceptron was implemented for purpose of training and predictions.
  - Additionally, the validation set was also created to have more insights into the training phase.
  - From the loss plots of training and validation data, it was evident that the learning keeps on getting better with more epochs in the case of simple gradient descent.
  - Moreover, from these plots, it could be established that 200 epochs are a good time for the model to learn.
  - Further, based on the loss plots of training and validation data in the case of optimizers it was quite clear that the model was overfitting on the training data after mere 5-7 epochs. Thus, in this case comparatively less number of epochs would suffice.
- **Saving Results**
  - To store the learnt parameters pickle was used.
  - The complete instance of MultiLayerPerceptron was saved with a name containing all the information about model configurations.

Observation:

Running the model on several configurations, the following things were observed:

- *The model works better with less number of hidden layers*, it was evident from the fact that the average testing accuracy for the model with 3 layers showed better results (about 30% increase) than the model with 4 layers.
- *The model works better with a higher learning rate*, it was evident from the fact the average testing accuracy for the model with a learning rate of 0.01 showed better results (about 13%-20% increase) than the model with a learning rate of 0.001.
- For simple gradient descent, the model keeps on improving its learning, with more epochs. (65% testing accuracy for 200 epochs and 70% testing accuracy for 300 epochs, with other hyperparameters being same)
- Out of tanh, relu and sigmoid activation functions, tanh showed the best result followed by relu and sigmoid (~70%, ~65% and ~60% respectively).

➔ For different optimizers, the model was able to learn the dataset very quickly and easily reached a testing accuracy of around 90% for most of the optimizers, except rmsprop for which it was still 85%.

➔ But, for these optimizers, the model started overfitting the training dataset way quickly taking only 5-7 epochs.

Outputs and Plots:

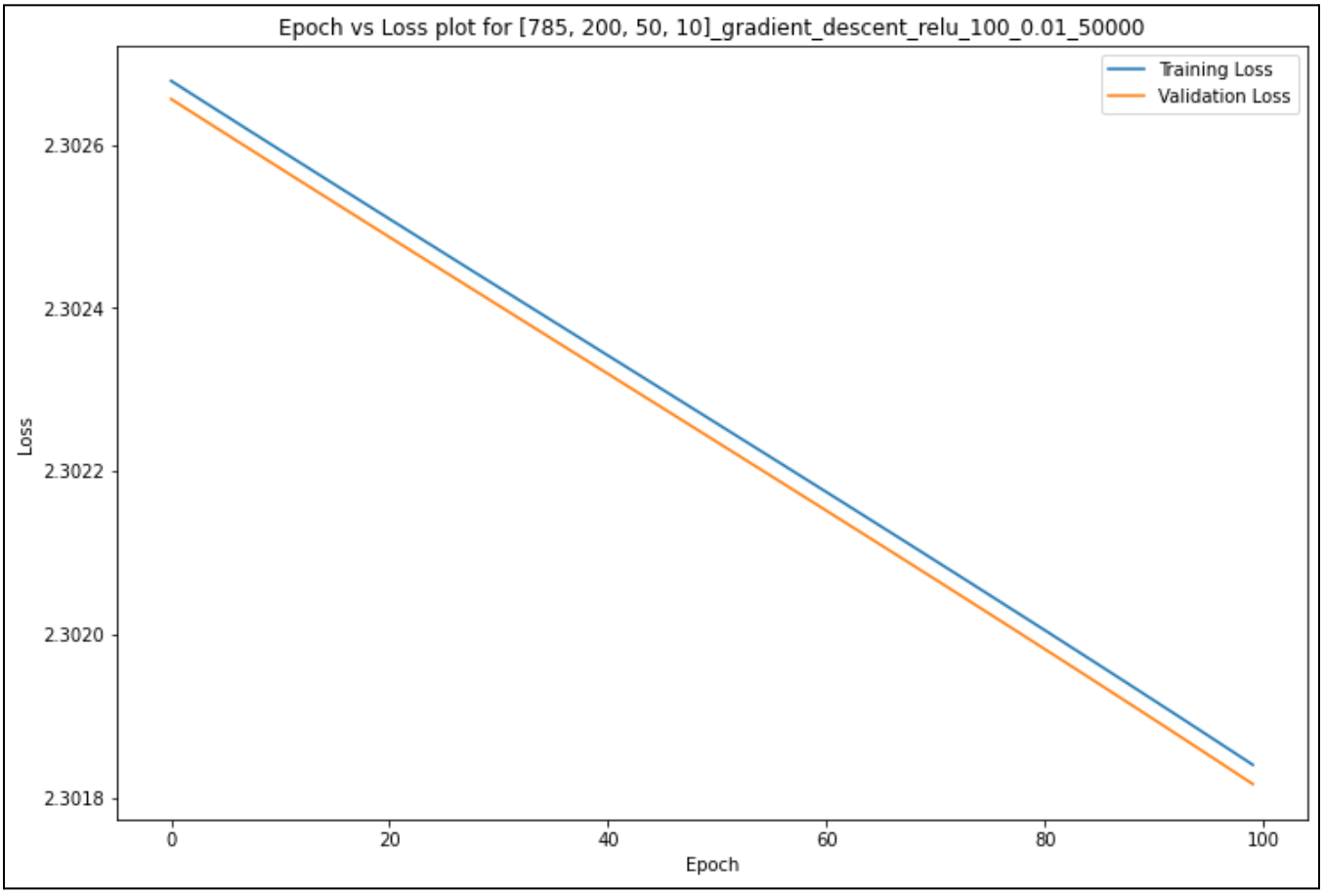
1.

i)

Configuration-

MulitLayerPerceptron(layers = [**784, 200, 50, 10**], num_epochs = 100, learning_rate = **1e-2**, activation_function = 'relu', batch_size = None, optimizer = 'gradient_descent')

Loss Plot-



Output-

Training Accuracy: 22.594 %

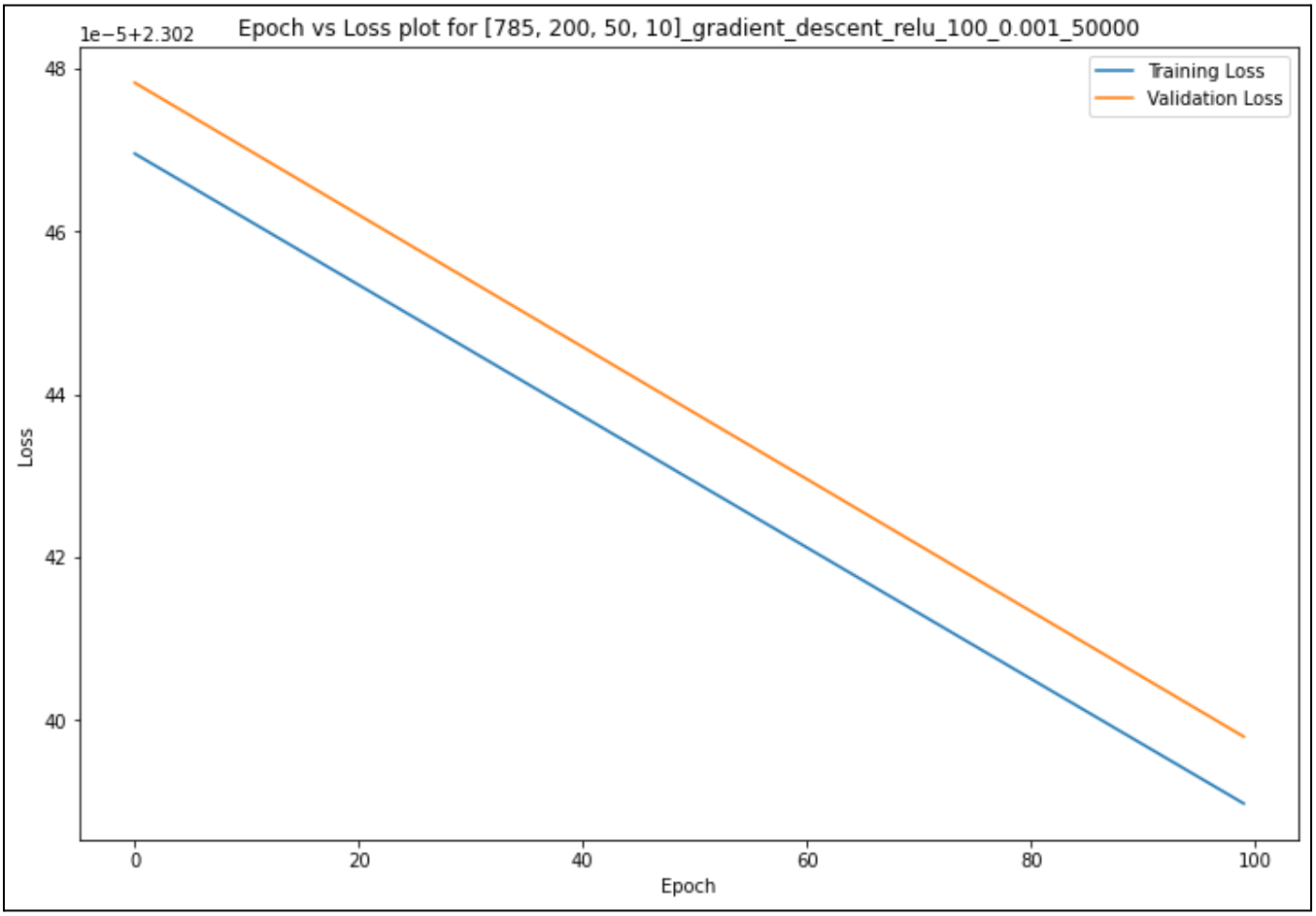
Testing Accuracy: 23.34 %

ii)

Configuration-

MulitLayerPerceptron(layers = **[784, 200, 50, 10]**, num_epochs = 100, learning_rate = **1e-3**, activation_function = 'relu', batch_size = None, optimizer = 'gradient_descent')

Loss Plot-



Output-

Training Accuracy: 9.158 %

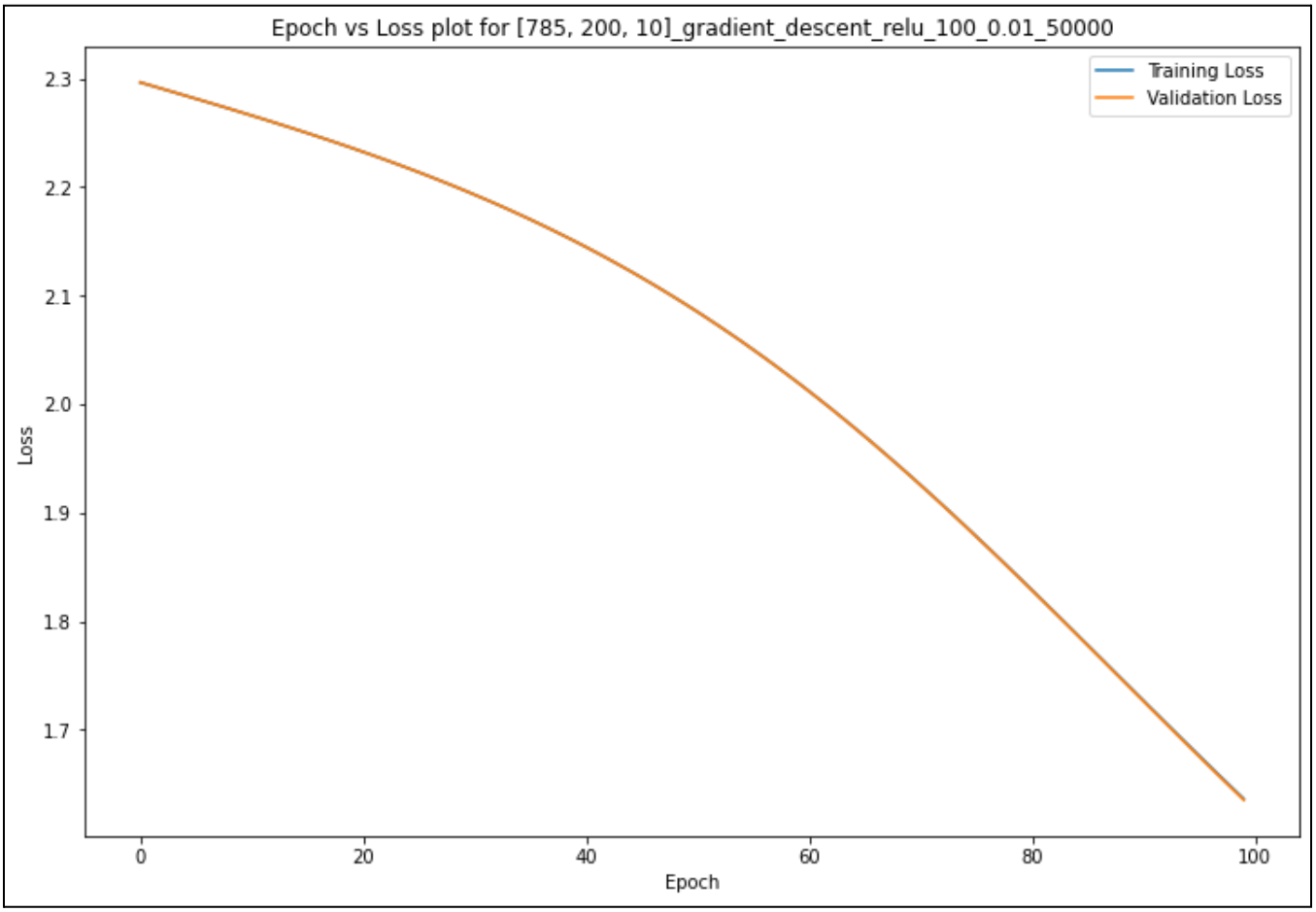
Testing Accuracy: 9.3 %

iii)

Configuration-

MulitLayerPerceptron(layers = **[784, 200, 10]**, num_epochs = 100, learning_rate = **1e-2**, activation_function = 'relu', batch_size = None, optimizer = 'gradient_descent')

Loss Plot-



Output-

Training Accuracy: 58.104 %

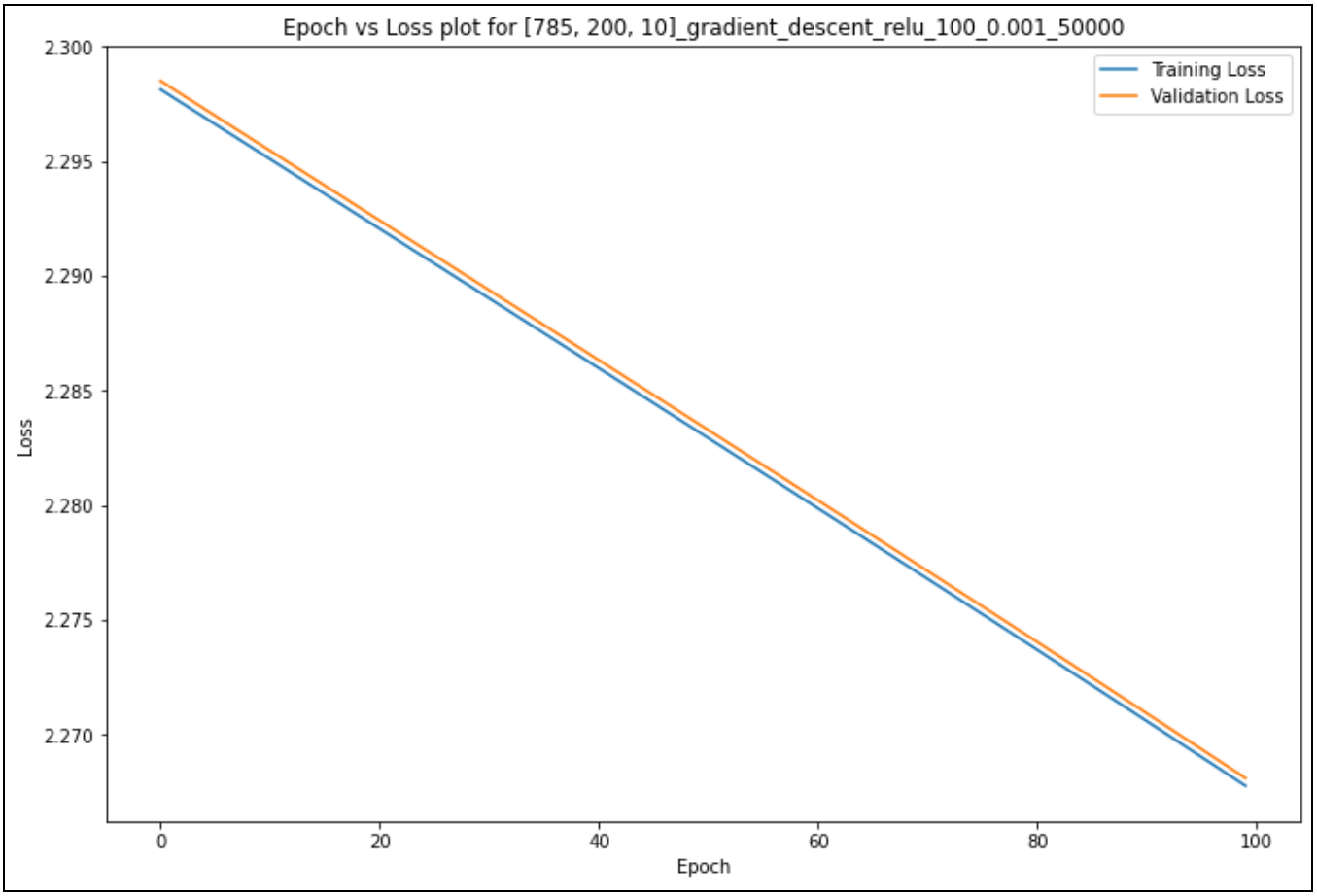
Testing Accuracy: 57.96 %

iv)

Configuration-

MulitLayerPerceptron(layers = **[784, 200, 10]**, num_epochs = 100, learning_rate = **1e-3**, activation_function = 'relu', batch_size = None, optimizer = 'gradient_descent')

Loss Plot-



Output-

Training Accuracy: 38.086 %

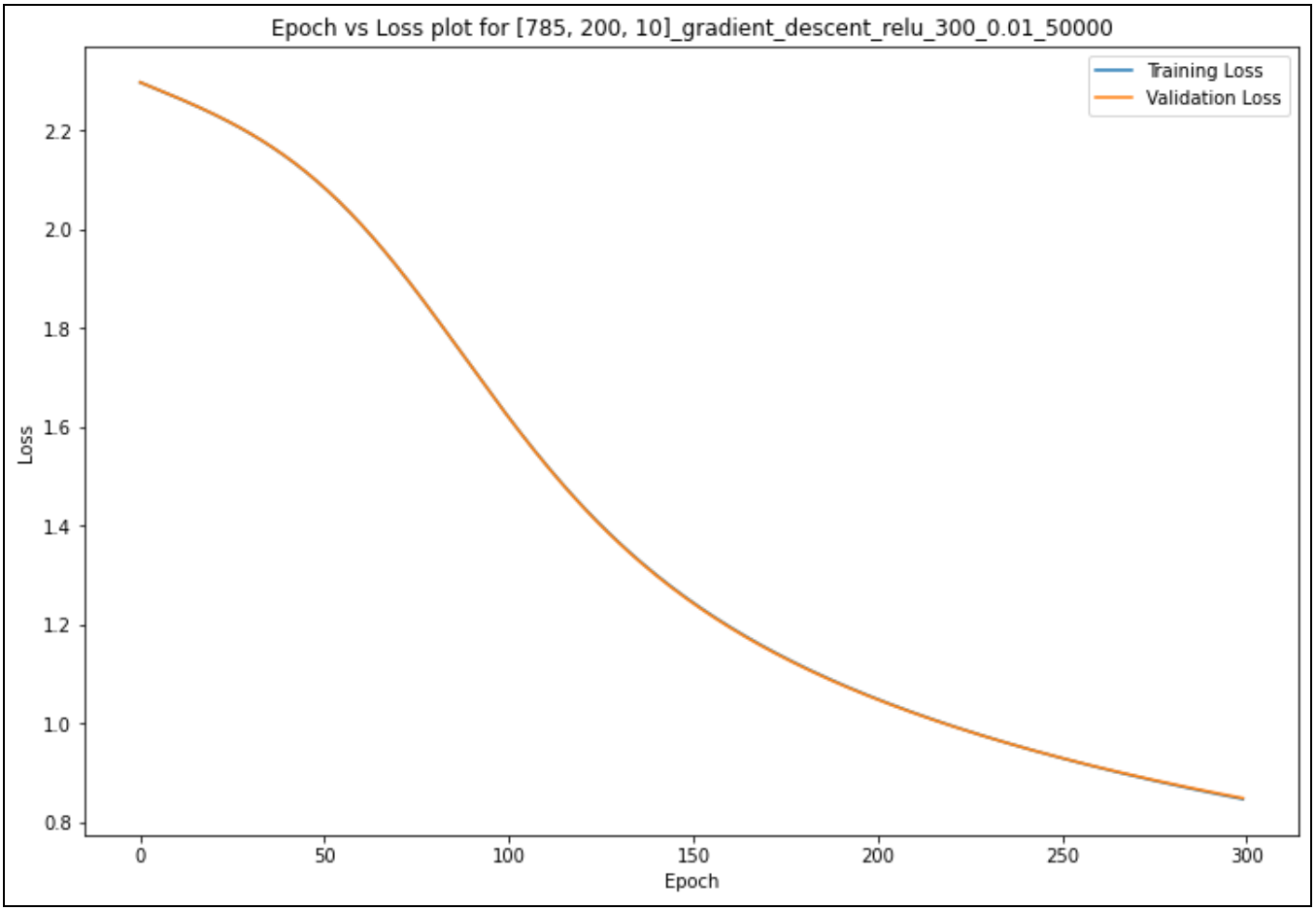
Testing Accuracy: 38.29 %

v)

Configuration-

MulitLayerPerceptron(layers = [784, 200, 10], num_epochs = **300**, learning_rate = 1e-2, activation_function = 'relu', batch_size = None, optimizer = 'gradient_descent')

Loss Plot-



Output-

Training Accuracy: 72.304 %

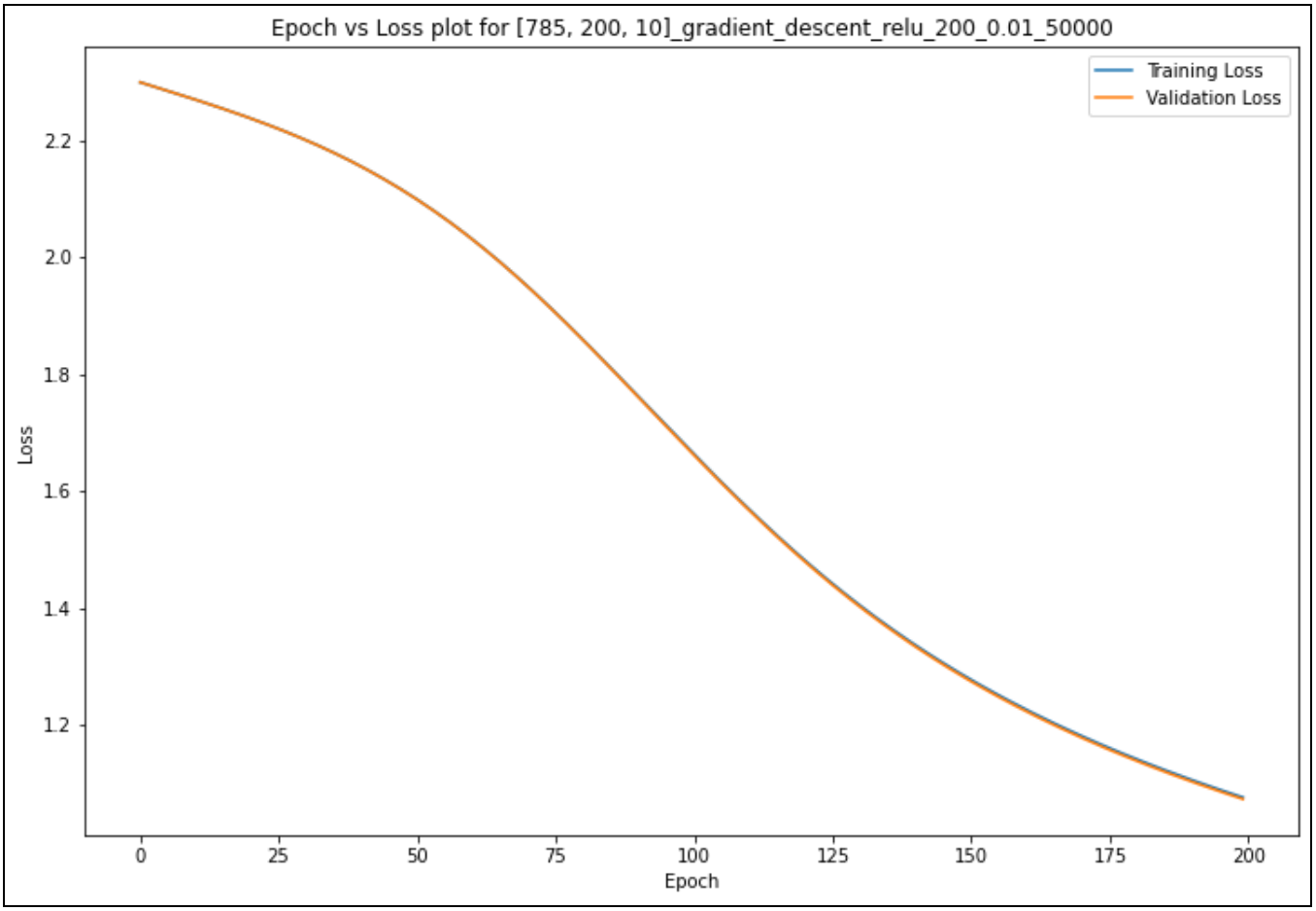
Testing Accuracy: 72.0 %

v)

Configuration-

MulitLayerPerceptron(layers = [784, 200, 10], num_epochs = **200**, learning_rate = 1e-2, activation_function = **'relu'**, batch_size = None, optimizer = 'gradient_descent')

Loss Plot-



Output-

Training Accuracy: 65.83200000000001 %

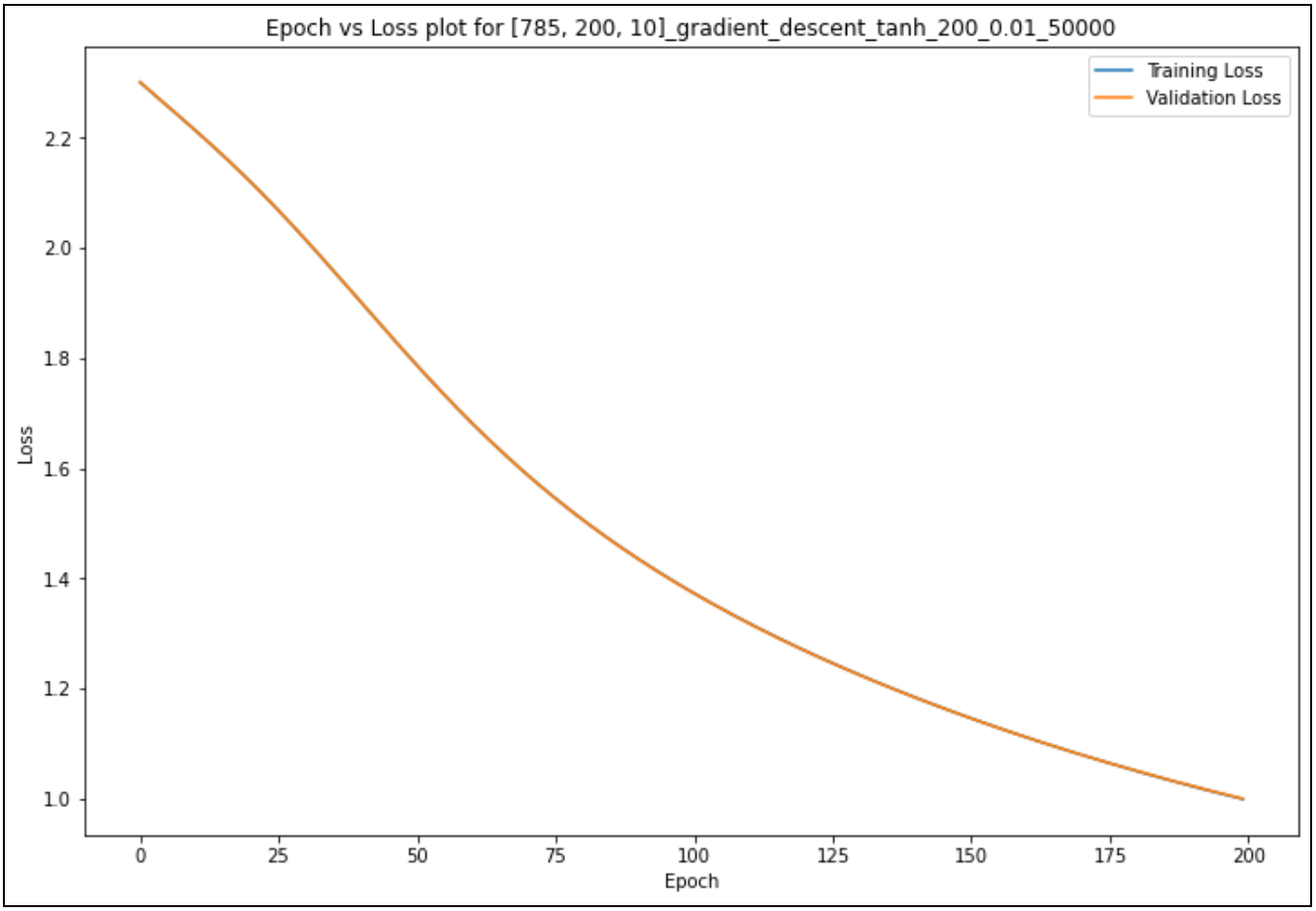
Testing Accuracy: 65.75999999999999 %

vi)

Configuration-

MulitLayerPerceptron(layers = [784, 200, 10], num_epochs = 200, learning_rate = 1e-2, activation_function = '**tanh**', batch_size = None, optimizer = 'gradient_descent')

Loss Plot-



Output-

Training Accuracy: 70.128 %

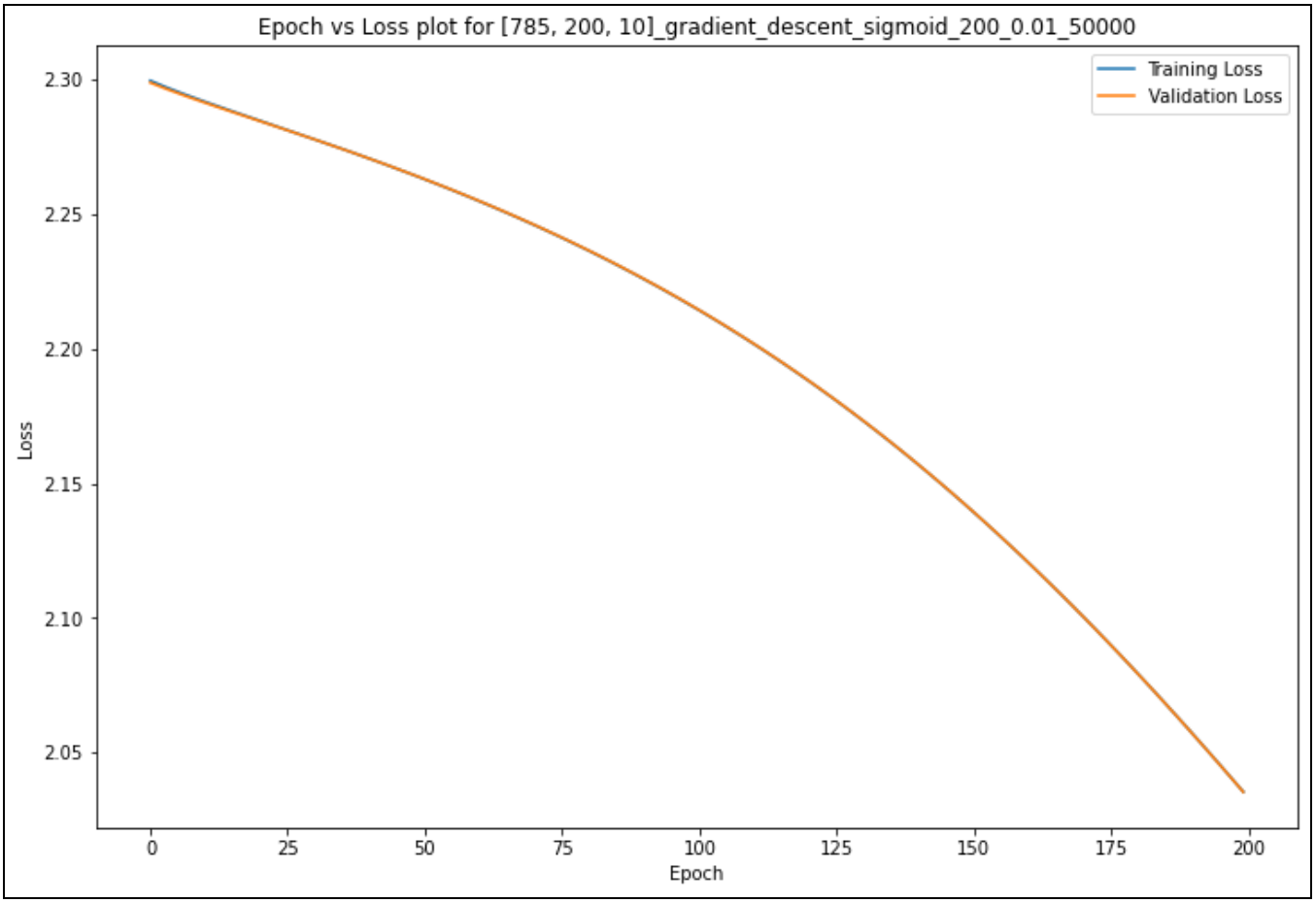
Testing Accuracy: 70.11 %

vii)

Configuration-

MulitLayerPerceptron(layers = [784, 200, 10], num_epochs = 200, learning_rate = 1e-2, activation_function = '**sigmoid**', batch_size = None, optimizer = 'gradient_descent')

Loss Plot-



Output-

Training Accuracy: 58.977999999999994 %

Testing Accuracy: 59.13 %

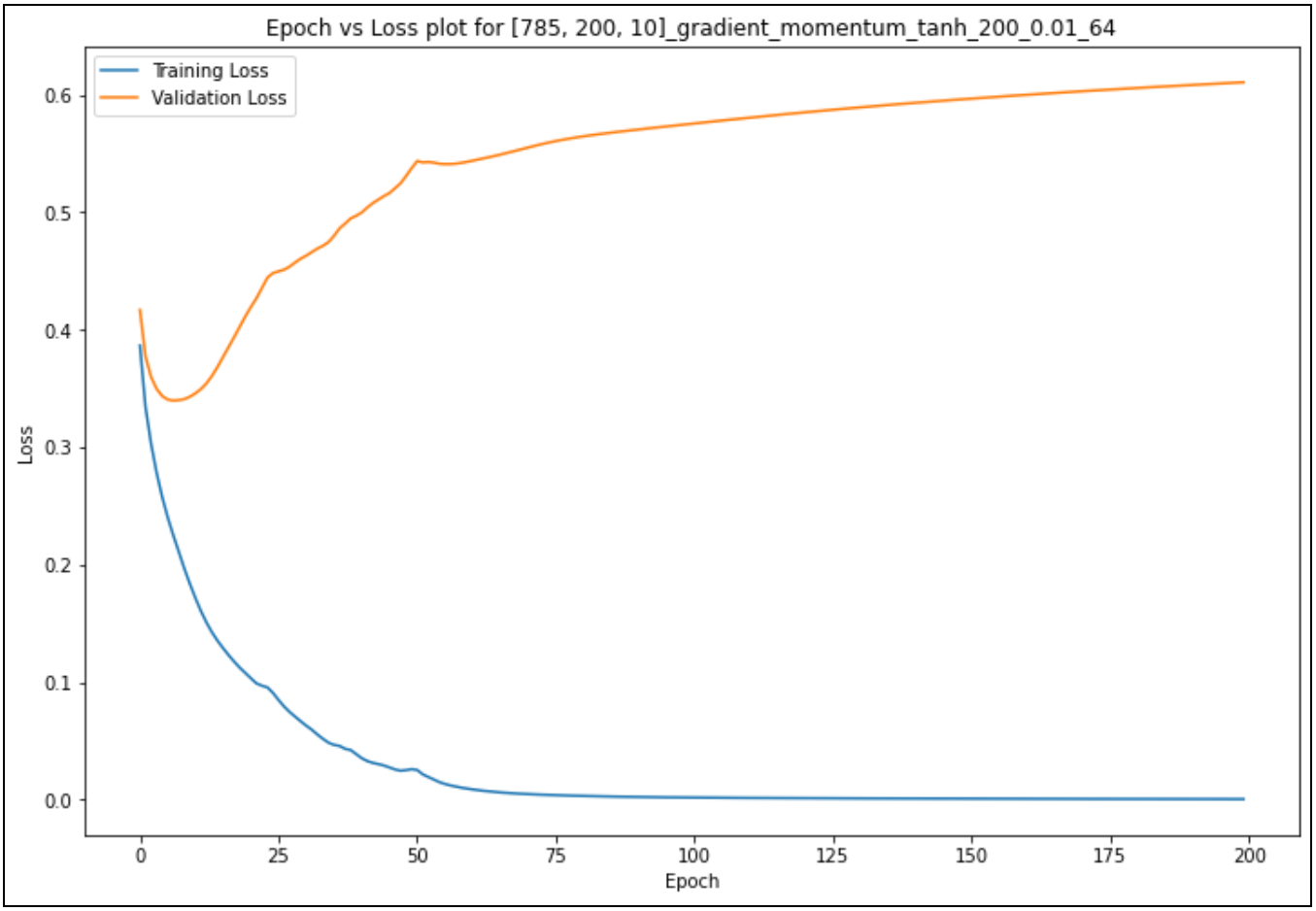
2.

viii)

Configuration-

MulitLayerPerceptron(layers = [784, 200, 10], num_epochs = 200, learning_rate = 1e-2, activation_function = '**tanh**', batch_size = 64, optimizer = '**gradient_momentum**')

Loss Plot-



Output-

Training Accuracy: 100.0 %

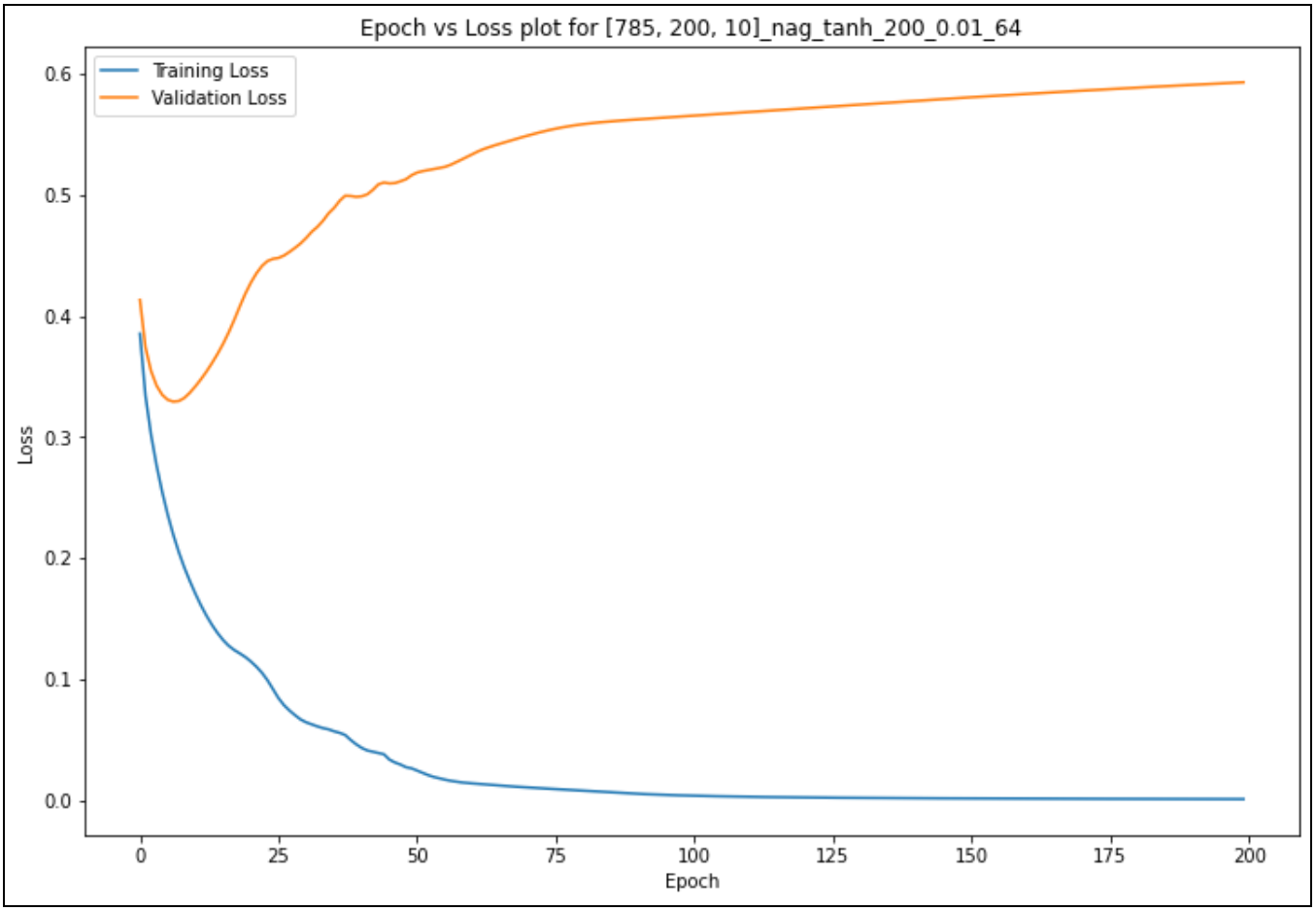
Testing Accuracy: 89.01 %

ix)

Configuration-

MulitLayerPerceptron(layers = [784, 200, 10], num_epochs = 200, learning_rate = 1e-2, activation_function = '**tanh**', batch_size = 64, optimizer = '**nag**')

Loss Plot-



Output-

Training Accuracy: 100.0 %

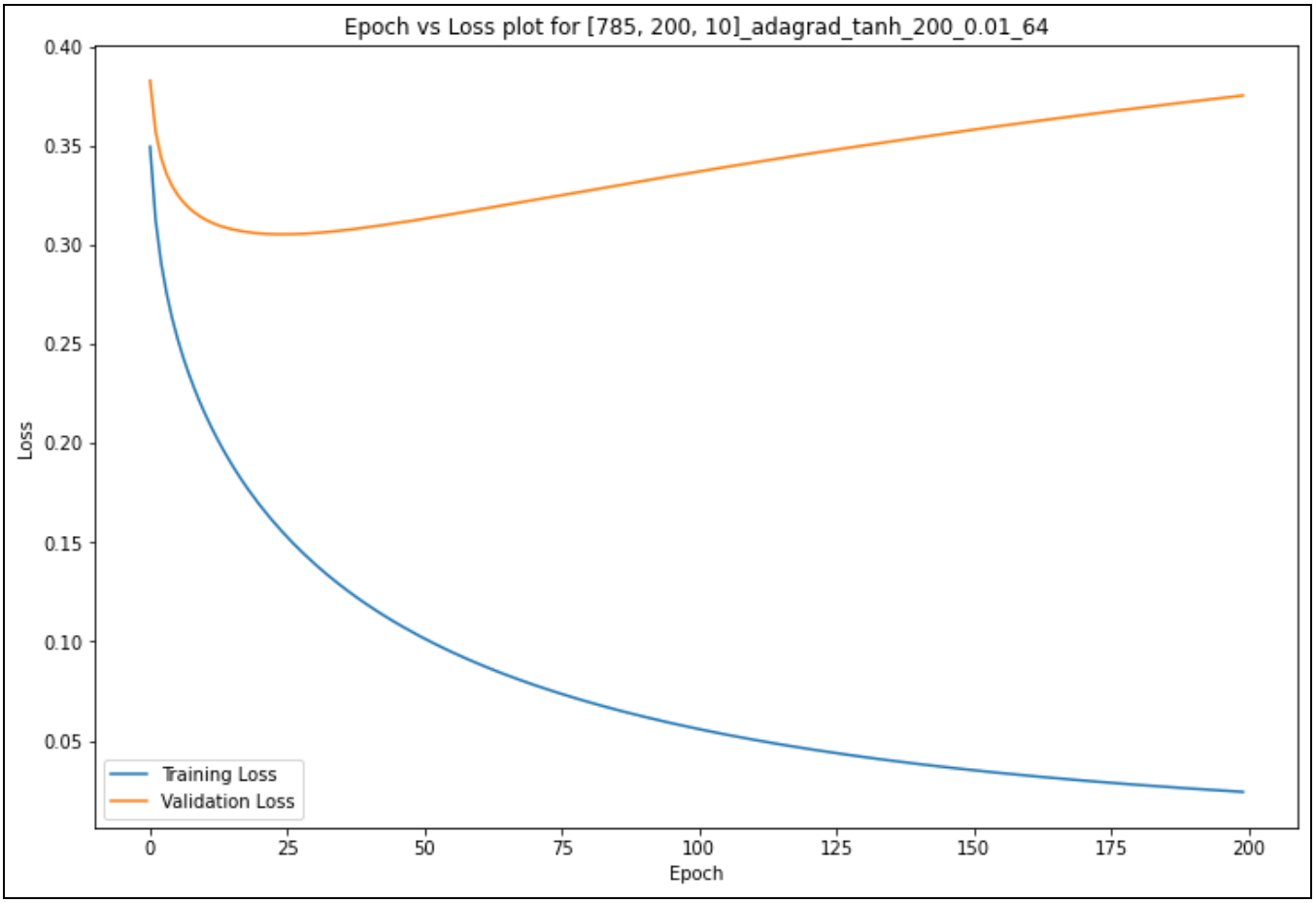
Testing Accuracy: 89.08 %

x)

Configuration-

MulitLayerPerceptron(layers = [784, 200, 10], num_epochs = 200, learning_rate = 1e-2, activation_function = '**tanh**', batch_size = 64, optimizer = '**adagrad**')

Loss Plot-



Output-

Training Accuracy: 99.86 %

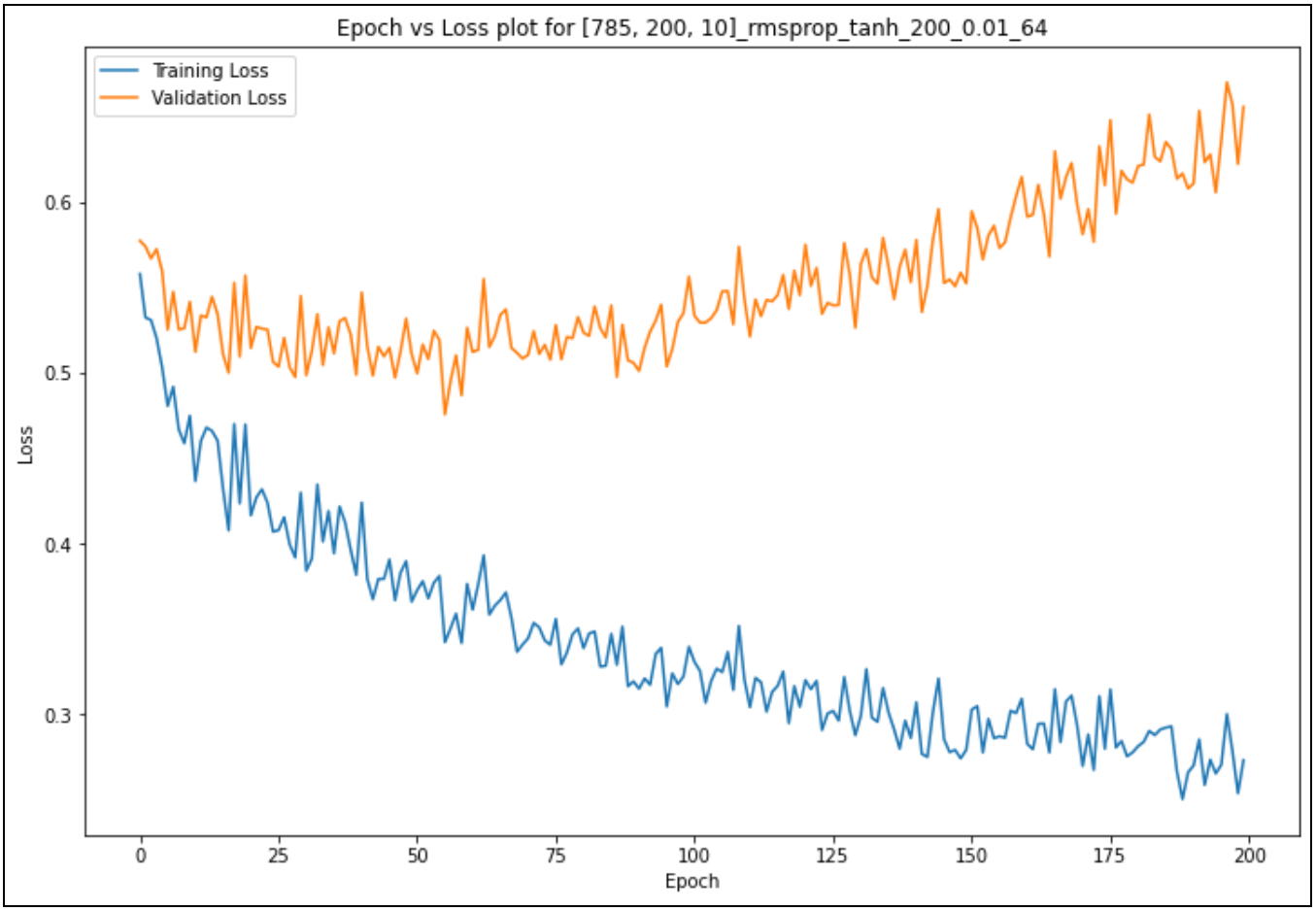
Testing Accuracy: 89.64 %

xi)

Configuration-

MulitLayerPerceptron(layers = [784, 200, 10], num_epochs = 200, learning_rate = 1e-2, activation_function = '**tanh**', batch_size = 64, optimizer = '**rmsprop**')

Loss Plot-



Output-

Training Accuracy: 91.608 %

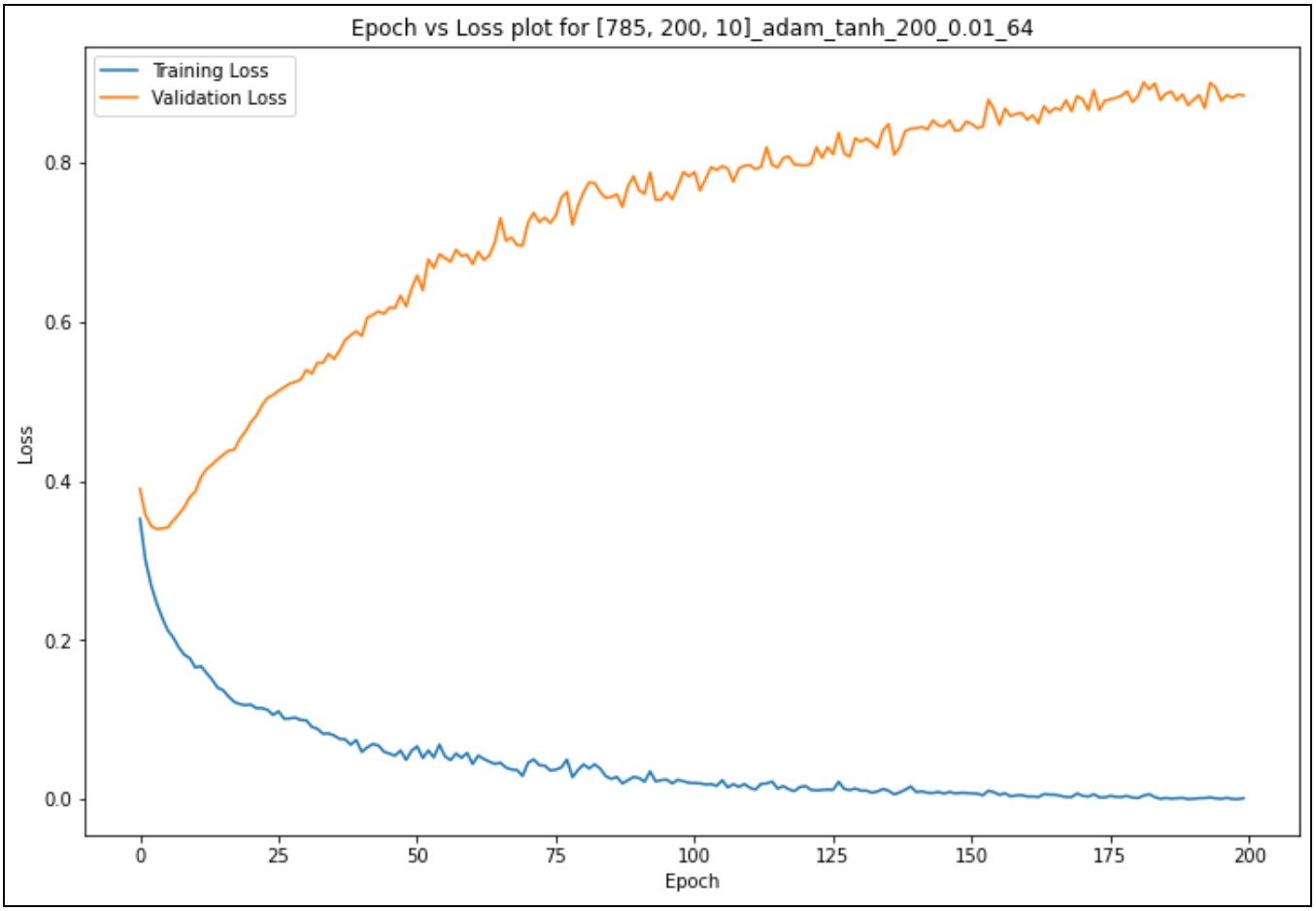
Testing Accuracy: 85.42999999999999 %

xii)

Configuration-

MulitLayerPerceptron(layers = [784, 200, 10], num_epochs = 200, learning_rate = 1e-2, activation_function = '**tanh**', batch_size = 64, optimizer = '**adam**')

Loss Plot-



Output-

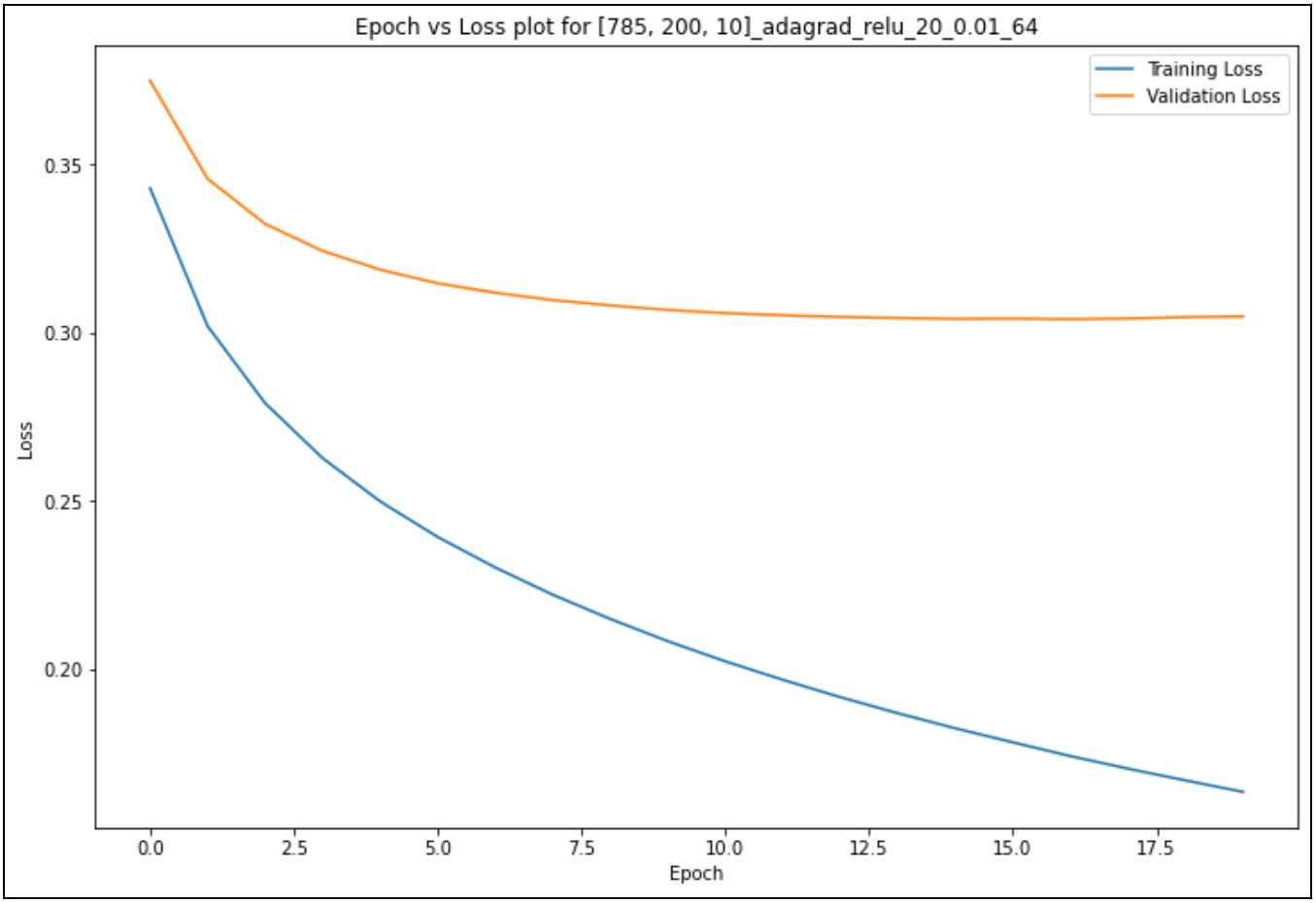
Training Accuracy: 99.958 %
Testing Accuracy: 88.06 %

xiii)

Configuration-

MulitLayerPerceptron(layers = [784, 200, 10], num_epochs = 200, learning_rate = 1e-2, activation_function = 'relu', batch_size = 64, optimizer = 'adagrad')

Loss Plot-



Output-

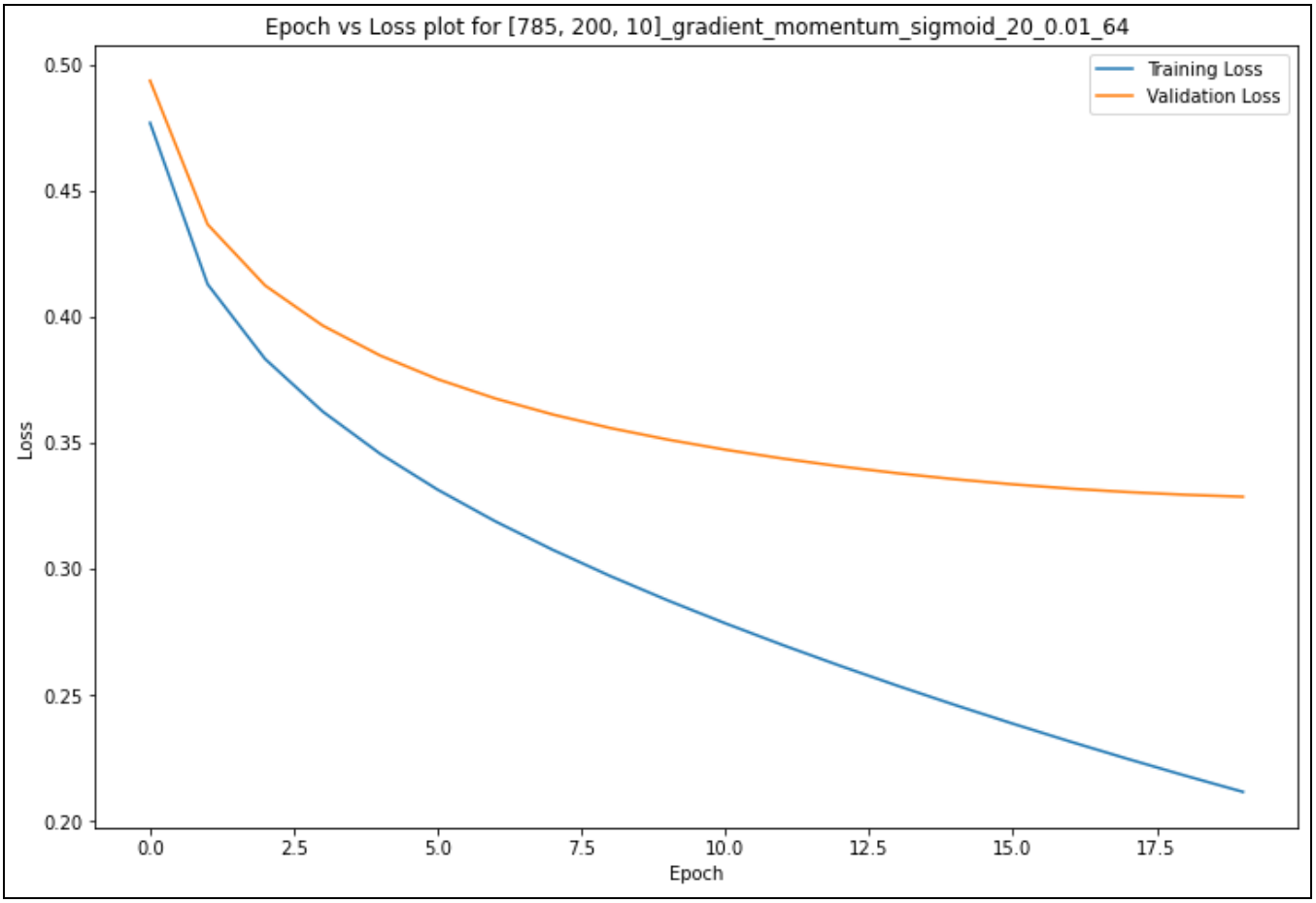
Training Accuracy: 94.552 %
Testing Accuracy: 89.82 %

xiv)

Configuration-

MulitLayerPerceptron(layers = [784, 200, 10], num_epochs = 200, learning_rate = 1e-2, activation_function = '**sigmoid**', batch_size = 64, optimizer = '**gradient_momentum**')

Loss Plot-



Output-

Training Accuracy: 92.636 %

Testing Accuracy: 88.68 %