MLP for MNIST data prediction using basic Python and Numpy

Manpreet Singh
Master of Applied Computer Science
B00853930
mn407919@dal.ca

Abstract

A multilayer perceptron (MLP) is a class of feedforward artificial neural network. This paper covers the implementation of an MLP for MNIST data prediction with basic python and Numpy without the help of a neural network package such as Keras or PyTorch.In this experiment, I have done hyperparameter tuning for three hyperparameters: learning rate, number of neurons, and number of hidden layers. Models were created with 1 hidden layer, 2 hidden layers, and 5 hidden layers and then trained on training data. It was found that the accuracy of MLP using 1 hidden layer performed best on the MNIST dataset with an accuracy of 95.89 percent and an absolute error of 94.39. I also trained the models using different numbers of neurons and different learning rates. The model with 100 neurons and 0.01 learning rate has the highest accuracy and outperformed the other models. Finally, the model with 1 hidden layer, 0.01 learning rate, and 100 neurons was tested on test data and also introduced noise in the test set and the accuracy after adding noise dropped from 32.59 percent to 15.13 percent.

1. Introduction

Neural networks are a set of algorithms that were inspired by the neural networks in the human brain. It is one of many different tools used in machine learning algorithms to process complex data. A multilayer perceptron is a neural network connecting multiple layers in a directed graph, MLP is widely used for solving problems as well as used for research into computational neuroscience, parallel distributed processing, applications like speech recognition, image recognition, and machine translation. The aim of this project is to implement an MLP from scratch. This gives us a deep understanding of underlying concepts like backpropagation, gradient descent, and regularization. The neural network is trained to classify handwritten digit images in the MNIST dataset. It translates the images into their ASCII representation. In this paper, details about hyperparameter tuning of the learning rate, number of layers, number of neurons is discussed. Further, regularization techniques like Ridge regression and LASSO are applied to the neural network and their comparative analysis is discussed. After training the model, its robustness is tested by adding noise to test data. Learning curves are generated for accuracy and error on training and test data for various experiments. Finally, the model was tested on the test set, and then to check the robustness of the model noise was introduced in the test set.

2. Dataset

The MNIST is a well-known dataset that is commonly used for training various image processing systems. It contains images with 28 × 28 pixels in gray scale, each coped with a label from 0 to 9, indicating the corresponding digit. Dataset contains 70,000 labelled images which is divided into training set of 60,000 images and 10,000 test images. I have picked 1024 images for training and 501 for testing. For this experiment, I needed to program number recognition system that translates the MNIST image into its ASCII representation. So, I converted output labels of MNIST dataset first into their ASCII and then into their binary representation.

3. Initial Architecture

First, I loaded the MNIST dataset. Then as the images were greyscale so the values of pixels were in the range of 0 - 255, so values of pixels were normalized and converted to the range of 0 to 1. Then input values were reshaped and converted to vector of 784 values. Once data was preprocessed, then initial architecture was built. The initial architecture used for the model consists of an input layer, one hidden layer and an output layer. The input layer has 785 neurons, hidden layer has 100 neurons and the output layer has 8 neurons. In this experiment, sigmoid activation function is used for hidden layers and output layer neurons. The number of trials for training the model were fixed to 100.

4. Hyperparameter Tuning

After creating the basic architecture of model, the next step was to improve the performance of the model on the training set. To improve the performance of the model I did hyperparameter tuning. The three hyperparameters which were tuned are: Number of hidden layers, Number of neurons, and learning rate. Experiments were performed using different values and combinations of these hyperparameters.

4.1. Number of Hidden Layers

I performed this experiment using a different number of hidden layers and by applying the sigmoid activation function on the neurons. All the other hyperparameters were fixed, the learning rate was 0.01, momentum was 0.9. Weights for the layers of the model were initialized using NumPy's random function. I have performed three experiments with a different number of layers. Based on the results, one hidden layer model has the best performance among the three models with an accuracy of 95.89 percent and an absolute error of 94.39. So, for the final MLP model, one hidden layer architecture was used.

Table 1 ACCURACY AND ABSOLUTE ERROR FOR DIFFERENT NUMBER OF LAYERS

Number of Hidden Layers	1 Layer	2 layers	5 layers
Number of Neuron per Layer	100	100	128
Accuracy	95.89	23.046	9.863
Absolute Error	94.39	1223.269	1749.00001

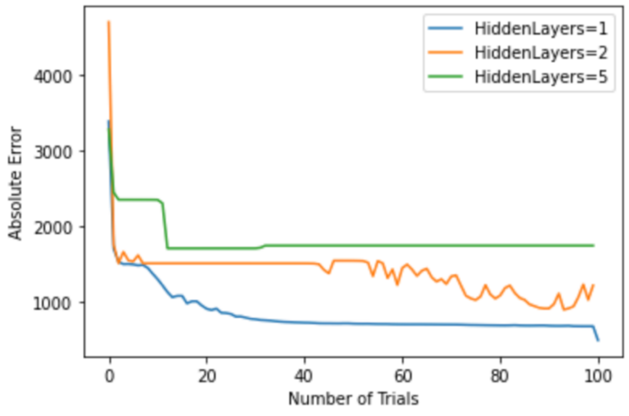

Figure 1 Absolute Error curve for different number of layers

4.2. Number of Neurons

I performed this experiment using a different number of neurons. All the other hyperparameters were fixed, the learning rate was 0.01, momentum was 0.9 and one hidden layer was used. In the first model, 100 neurons were used in the hidden layer, in the 2nd model 10 neurons were used, and in the 3rd model 1000 neurons were used. The model with 100 neurons has the highest accuracy of 93.95 percent with the least absolute error of 132.62.

Table 2 ACCURACY AND ABSOLUTE ERROR FOR DIFFERENT NUMBER OF NEURONS

Number of Neurons	100 Neurons	10 Neuron	1000 Neurons
Accuracy	93.95	68.84	34.57
Absolute Error	132.62	564.93	741.080

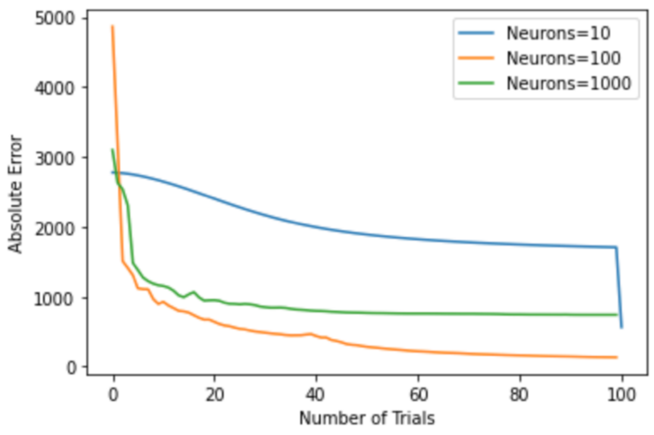

Figure 2 Absolute Error curve for different number of neurons

4.3. Learning rate

I performed this experiment with different learning rates. All the other hyperparameters were fixed, the learning rate was 0.01, momentum was 0.9 and one hidden layer with 100 neurons were used. In the first model, the learning rate was 0.001, in 2nd learning rate was 0.01 and in 3rd learning, the rate was 0.000001. The number of neurons in the hidden layer was 100. The model with a learning rate of 0.01 has the highest accuracy of 78.027 percent with the least absolute error of 274.96.

Table 3 ACCURACY AND ABSOLUTE ERROR FOR DIFFERENT LEARNING RATES

Learning rate	0.001	0.01	0.000001
Accuracy	68.066	78.027	7.324
Absolute Error	537.15	274.96	1707.49

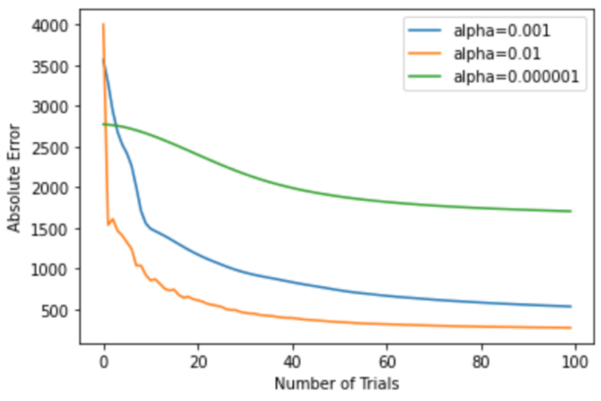

Figure 3 Absolute Error curve for different Learning rates

5. Regularization

Regularization is a technique to reduce the overfitting of data in a model by penalizing the magnitude of coefficients of features. In this experiment, LASSO and ridge regression was applied on training data to avoid overfitting of data and so that model can work well on unseen data. In LASSO penalty is equivalent to the absolute value of the magnitude of coefficients is added to the error term whereas in Ridge regression penalty is equivalent to the square of the magnitude of coefficients is added to the error term. The key difference between ridge and lasso regression is that LASSO tends to make coefficients to absolute zero as compared to Ridge which never sets the value of coefficient to absolute zero. From the experiment, it was clear that LASSO regularization has more accuracy (94.01 percent) on training data and it reduces overfitting to a more extent.

Table 4 ACCURACY AND ABSOLUTE ERROR FOR LASSO AND RIDGE REGULARIZATION

Regularization	Ridge regression	LASSO
Train Accuracy	73.04	94.01
Train Error	21796.6	140.48
Test Accuracy	24.90	28.71
Test Error	365.422	300.309

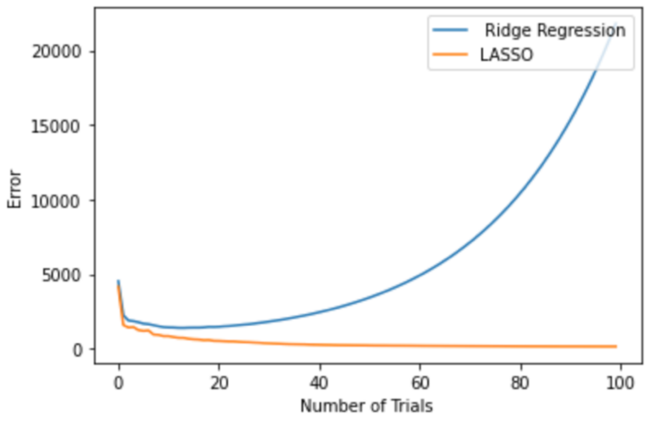

Figure 4 Absolute error curve for Ridge and LASSO for train data

As can be seen from above table LASSO regularization outperforms ridge regression by giving train accuracy 94.01 percent and test accuracy of 28.71 percent. So, in final model LASSO regularization was used but there is overfitting on train data as the variance between train accuracy and test accuracy is 66 percent.

6. Testing

After training the models on train data, I tested the model's accuracy on test data. The model with 1 hidden layer, 785 neurons in the input layer,100 neurons in the hidden layer, 8 neurons in the output layer, and learning rate 0.01 was tested on test data. The accuracy was 33.300 percent with an absolute error of 244.036. After this noise was added to test data which reduced the accuracy of the model on the data to 13.37 percent and absolute error was increased to 646.29.

Table 5 ACCURACY AND ABSOLUTE ERROR FOR TEST DATA

Data	Test data	Data after adding noise
Accuracy	33.300	13.37
Absolute Error	244.036	646.29

7. Conclusion

From the above experiments, it can be concluded that the model with one hidden layer performs better than models with more hidden layers as it giver accuracy of 94.39 percent whereas model with 2 layers gives accuracy of 23.046. In terms of learning rate, it can be concluded that using 0.01 learning rate as compared to 0.001 and 0.000001 gives highest accuracy of 78.027 percent for the model. LASSO regularizations reduce more overfitting than ridge regression and accuracy of model using LASSO is 94.01 on train set and 28.71% on test set.
It can be concluded that model is overfitting on train data as the variance between train and test accuracy is 60%. Furthermore, it can be seen that model is not robust to the noise as the accuracy of test set drops from 33% to 13% after introducing noise. So, there is lot of scope for improvement in the model and further experiments can be done to improve regularization as well as robustness of the model.

8. References

[1] Ciresan, D.C., Meier, U., Gambardella, L.M., Schmidhuber, J.: Deep, big, simple neural nets for handwritten digit recognition. Neural Computation 22(12) (2010) 3207–3220

[2] LeCun, Y., Cortes, C.: THE MNIST DATABASE of handwritten digits, http://yann.lecun.com/exdb/mnist/

[3] T. P. Trappenberg, Fundamentals of machine learning. Oxford, United Kingdom: Oxford University Press, 2020.