Bharath Kollanur

CSC 7700 – Project 1 Multi-Layer Perceptron

James Ghawaly

26 February 2025

Abstract

This project implements a Multi-Layer Perceptron (MLP) engine from scratch using NumPy and applies it to two tasks: handwritten digit classification using the MNIST dataset and vehicle fuel efficiency prediction using the Auto MPG dataset. The MLP engine supports multiple activation functions, dropout regularization, and RMSProp optimization. For MNIST classification, an 8-layer deep MLP with Relu activation and SoftMax output was trained using Cross Entropy Loss function and achieved an accuracy of **96.83%**, effectively classifying digits. For vehicle MPG prediction, a regression MLP with a linear output layer was trained using Mean Squared Error (MSE) loss and achieved a total loss of **6.7961**. Results demonstrate that the MLP engine successfully generalizes to both classification and regression problems while highlighting the importance of hyperparameter tuning and architectural choices.

Methodology

MLP Engine Implementation

The MLP engine was implemented using an object-oriented approach, ensuring modularity and flexibility. The core components include classes for layers, activation functions, loss functions, and a batch generator to support mini-batch training.

Architecture and Components

1. **Batch Generator:** A generator function that randomly shuffles and partitions the dataset into mini-batches, improving convergence stability.

2. **Activation Functions:** Implemented as an abstract base class (ActivationFunction) with multiple derived functions such as ReLU, Sigmoid, Tanh, SoftMax, Mish, and Softplus. Each activation function includes both forward propagation and derivative calculations for backpropagation.

3. **Loss Functions:** The engine supports SquaredError for regression and CrossEntropy for classification, both implemented as subclasses of LossFunction. These functions compute the loss and its gradient for optimization.

4. **Layer Class:** Defines a fully connected (dense) layer, initializing weights using Glorot Uniform Initialization and biases as zeros. It performs forward propagation using matrix multiplications and activation functions and supports dropout regularization during training.

5. **Backpropagation:** The layer computes gradients of weights and biases using chain rule differentiation, allowing for efficient parameter updates.

6. **Multilayer Perceptron (MLP) Class:** Encapsulates the neural network, handling both forward and backward passes. It supports various depth configurations and different learning strategies.

Training and Optimization

• The training function implements **stochastic gradient descent (SGD)** with optional **RMSProp** for adaptive learning rate adjustments.

• **Mini-batch training** is used to balance computational efficiency and convergence speed.

• Hyperparameters such as learning rate, batch size, dropout rate and number of epochs are adjustable, allowing for fine-tuning of performance.

• Training and validation losses are tracked per epoch for performance monitoring.

This structured implementation allows for scalable neural network training, providing support for both classification and regression tasks while enabling further customization.

MLP Design and Training Challenges

MNIST MLP Design

- Input layer: 784 neurons (flattened image pixels)
- Four hidden layers with dropout rate of 0.01:
  - 256 neurons with ReLU activation.
  - 128 neurons with ReLU activation.
  - 64 neurons with ReLU activation.
  - 32 neurons with ReLU activation.
- Output layer with 10 neurons (one for each digit class) using SoftMax activation.

MPG MLP Design

- Input layer corresponding to the number of features after preprocessing.
- Eight hidden layers with 128 neurons each using ReLU activation and a dropout rate of 0.03.
- Output layer with a single neuron using a Linear activation function for continuous value prediction.

For MNIST classification, an 8-layer MLP was designed with 64 hidden units per layer, ReLU activation, and a SoftMax output layer. The model was trained using cross-entropy loss with mini-batches of size 64 and RMSProp optimization. One challenge was overfitting due to the depth of the network, which was mitigated using dropout regularization at a rate of 0.01 and learning rate tuning. The final model achieved strong generalization on the test set.

For vehicle MPG prediction, a regression MLP with 8 hidden layers was used, where the output layer had a single neuron with a linear activation function. The model was trained using the Mean Squared Error (MSE) loss function, and hyperparameter tuning was performed to optimize learning rate and batch size. A challenge was handling missing values in the dataset, which was resolved using data substitution techniques.

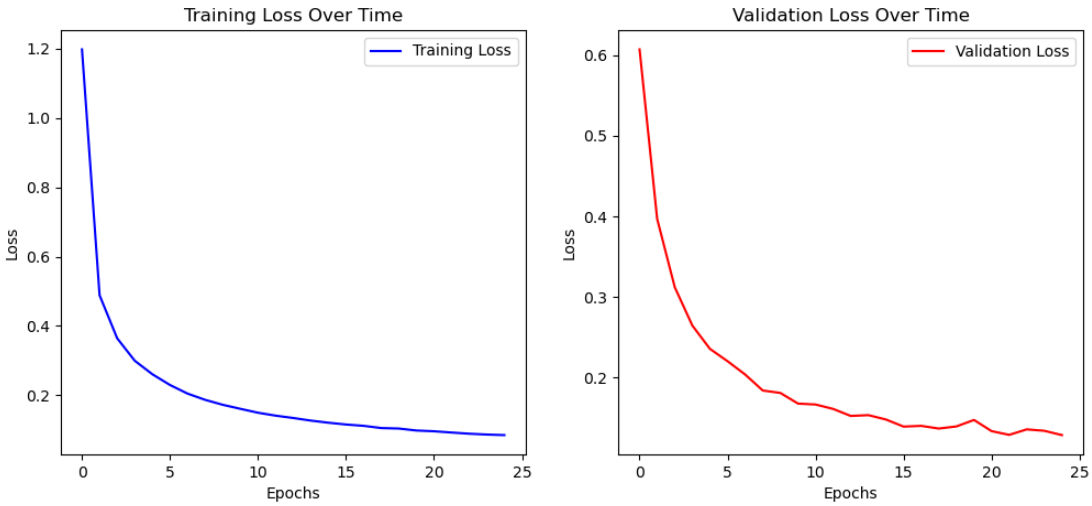
The project demonstrates that a well-structured MLP engine can effectively handle both classification and regression tasks, with results dependent on network architecture, hyperparameter selection, and data preprocessing techniques.

Results

For MNIST Classification dataset:

Loss curves (both training and validation)

Fig 1. Training and Loss validation curves of training/validation MNIST dataset over no. of epochs



Test accuracy and confusion matrix.

Fig 2. Confusion matrix and Accuracy of Test MNIST dataset for MLP classification model

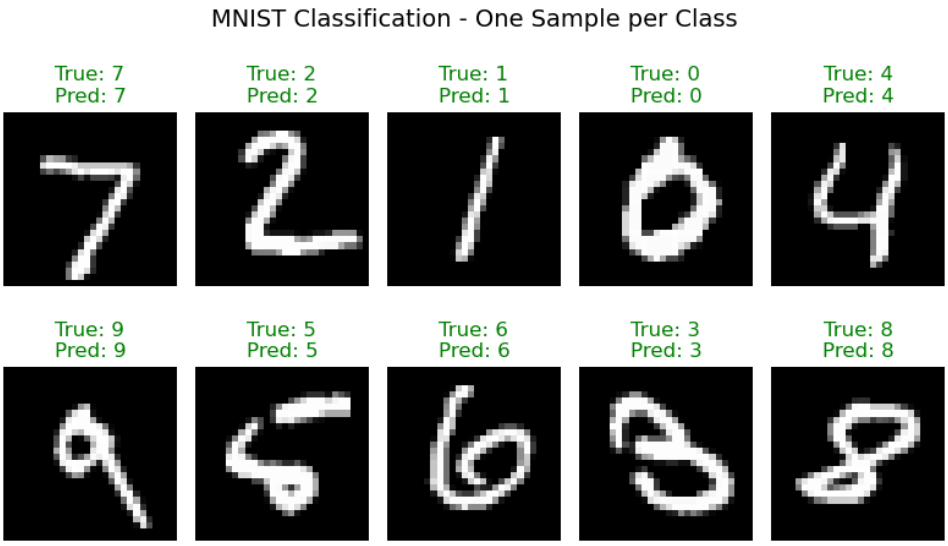
```
Test Accuracy: 0.9683
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       980
           1       0.98      0.98      0.98      1135
           2       0.98      0.95      0.96      1032
           3       0.95      0.97      0.96      1010
           4       0.97      0.97      0.97       982
           5       0.96      0.97      0.96       892
           6       0.97      0.98      0.97       958
           7       0.97      0.97      0.97      1028
           8       0.96      0.95      0.96       974
           9       0.97      0.96      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

Showcasing sample of each class (0-9) in the MNIST dataset along with predicted class for each class.

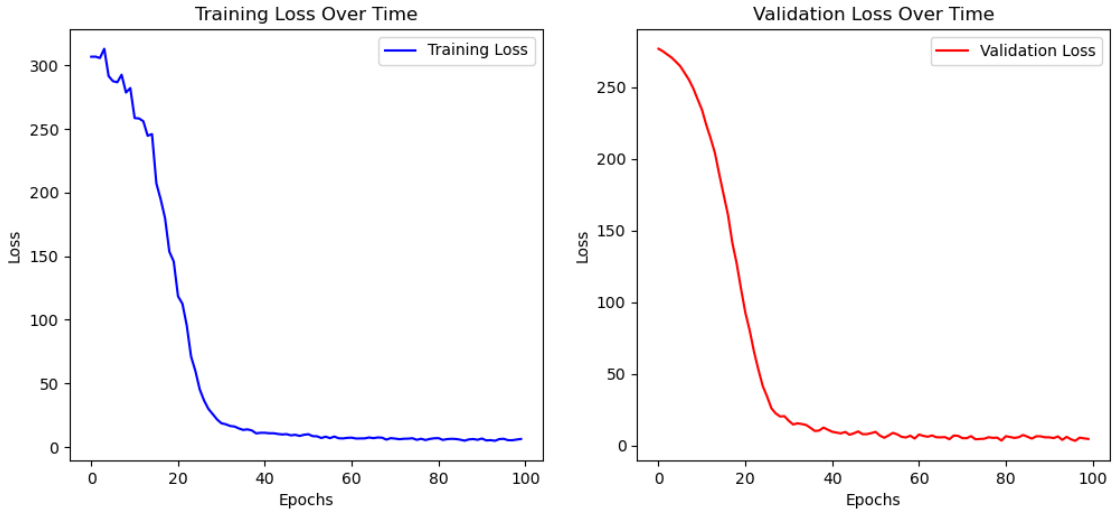
Fig 3. Test labels along precited class values of MNIST Dataset



For MPG Regression dataset:

Loss curves (both training and validation)

Fig 4. Training and Loss validation curves of training/validation MPG dataset over no. of epochs



Total testing loss over the test dataset (Mean Squared Error) : **6.7961**

R2 Score: **0.8625**

Showcasing 10 different sample from test data and predicting MPG against true MPG:

```
================================================
🔷 Predicted vs True MPG (10 Random Samples)
================================================
+-------+----------+---------------+-----------+
| Index | True MPG | Predicted MPG | Abs Error |
+-------+----------+---------------+-----------+
|  11.0 |  31.90   |     35.28     |    3.38   |
|  38.0 |  29.00   |     31.69     |    2.69   |
|  24.0 |  14.00   |     13.48     |    0.52   |
|  2.0  |  27.00   |     29.11     |    2.11   |
|  0.0  |  20.00   |     19.70     |    0.30   |
|  21.0 |  13.00   |     14.39     |    1.39   |
|  54.0 |  29.80   |     38.34     |    8.54   |
|  32.0 |  14.00   |     13.48     |    0.52   |
|  31.0 |  34.10   |     35.95     |    1.85   |
|  6.0  |  15.00   |     12.32     |    2.68   |
+-------+----------+---------------+-----------+
================================================
```

Table 1. Predicted values of 10 random test samples from test MPG dataset

Code Repo Link: https://github.com/kollanur/Project-1-CSC-7700-Spring-2025

Discussion & Conclusion

The experimental results demonstrate the effectiveness of MLP models in both classification and regression tasks. The MNIST classification model achieved high accuracy, reflecting the network's capability to learn complex patterns in image data. For the Auto MPG dataset, the deep MLP regression model effectively predicted fuel efficiency, though minor deviations in the results indicate sensitivity to hyperparameter selection. A key challenge was overfitting, particularly in the deeper network, which was mitigated through dropout regularization. Hyperparameter tuning, including selecting optimal learning rates, dropout rates, and activation functions, played a crucial role in improving model performance. The project highlights the trade-offs between network complexity and generalization, emphasizing the need for systematic tuning in deep learning applications.