

# Bike Sharing Demand Prediction using PyTorch-based MLP

Student Name  
Machine Learning A

November 25, 2025

## 1 Introduction

Bike sharing systems are a means of renting bicycles where the process of obtaining membership, rental, and return is automated via a network of kiosk locations throughout a city. Using these systems, people are able rent a bike from a one location and return it to a different place on an as-needed basis. Currently, there are over 500 bike-sharing programs around the world.

The goal of this report is to predict the total count of bikes rented during each hour covered by the test set, using data provided by the Kaggle competition "Bike Sharing Demand". Accurate prediction of bike demand is crucial for efficient fleet management, ensuring that bikes are available when and where users need them, thereby improving user satisfaction and operational efficiency. This can also contribute to reducing traffic congestion and promoting eco-friendly transportation.

## 2 Method

### 2.1 Data Collection

The dataset is obtained from the Kaggle competition "Bike Sharing Demand". It contains historical usage patterns with weather data. The training set is comprised of the first 19 days of each month, while the test set is the 20th to the end of the month.

### 2.2 Preprocessing

To prepare the data for the neural network model, several preprocessing steps were applied:

- **Feature Engineering:** The `datetime` column was decomposed into year, month, day, weekday, and hour to capture temporal patterns such as daily cycles and seasonal trends.
- **Categorical Encoding:** Categorical variables such as `season` and `weather` were One-Hot Encoded to allow the model to interpret them correctly without assuming an ordinal relationship.
- **Target Transformation:** The target variable `count` was transformed using  $\log(1 + x)$  to handle the skewed distribution and to align with the evaluation metric (RMSLE).
- **Standardization:** All input features were standardized using StandardScaler (zero mean and unit variance). This is critical for Multi-Layer Perceptrons (MLP) because neural networks converge faster and perform better when inputs are on a similar scale, preventing gradients from vanishing or exploding.
- **Feature Selection:** Columns `casual` and `registered` were removed as they are not present in the test set and sum up to the target `count`.

## 2.3 Models

### 2.3.1 Linear Regression (Baseline)

Linear Regression models the relationship between the dependent variable  $y$  and independent variables  $\mathbf{x}$  as a linear combination:

$$y = \mathbf{w}^T \mathbf{x} + b \quad (1)$$

This model serves as a baseline to evaluate the effectiveness of more complex models. We used the implementation from `scikit-learn`.

### 2.3.2 Multi-Layer Perceptron (MLP) with PyTorch

We implemented a Multi-Layer Perceptron using the **PyTorch** deep learning framework. The network architecture consists of:

- Input layer matching the number of features.
- Hidden layers with configurable sizes and activation functions (ReLU or Tanh).
- Output layer with a single neuron (regression).

The forward pass for a layer  $l$  is given by:

$$a^{(l)} = \sigma \left( W^{(l)} a^{(l-1)} + b^{(l)} \right) \quad (2)$$

where  $W^{(l)}$  and  $b^{(l)}$  are the weights and biases, and  $\sigma$  is the activation function.

We used the **Adam** optimizer and **Mean Squared Error (MSE)** loss function. Since the target is log-transformed, minimizing MSE is equivalent to minimizing RMSLE on the original scale.

## 3 Experimental Settings

The training data was split into a training set (80%) and a validation set (20%). We performed a grid search to find the best hyperparameters for the MLP. The search space was:

- Hidden Layer Sizes: (50,), (100,), (50, 50)
- Activation Functions: ReLU, Tanh
- Alpha (Weight Decay): 0.0001, 0.01
- Initial Learning Rate: 0.001, 0.01

The maximum number of epochs was set to 500, with early stopping based on validation loss.

## 4 Results

### 4.1 Performance Comparison

The evaluation metric is the Root Mean Squared Logarithmic Error (RMSLE). The results on the validation set are shown in Table ??.

The PyTorch-based MLP Regressor significantly outperformed the Linear Regression baseline, achieving an RMSLE of 0.2956 compared to 1.0161. This demonstrates the effectiveness of deep learning in capturing non-linear relationships in the data.

Model	RMSLE
Linear Regression	1.0161
MLP Regressor (PyTorch)	0.2956

Table 1: Validation RMSLE Comparison

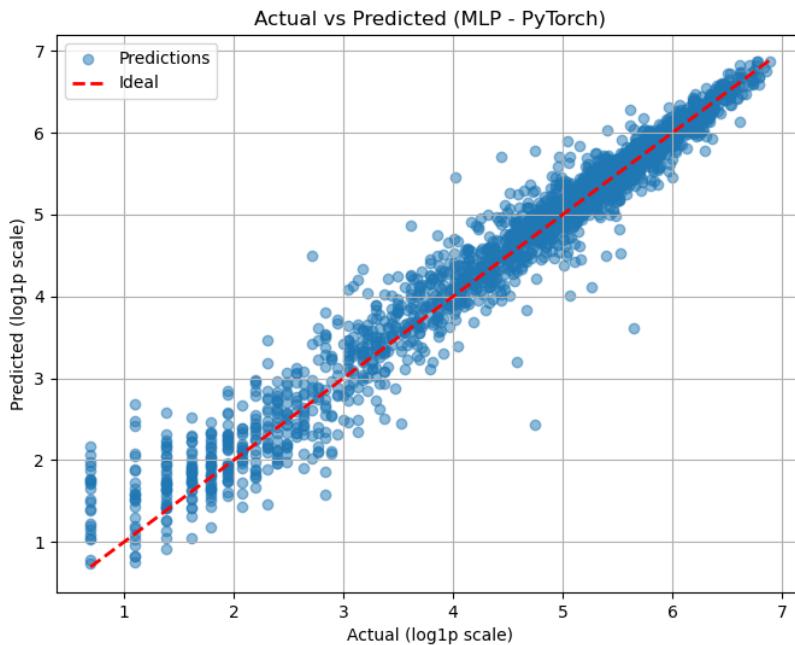


Figure 1: Actual vs Predicted Values (MLP - PyTorch)

## 4.2 Visualization

Figure ?? shows the scatter plot of actual vs. predicted values for the best MLP model. The points are tightly clustered around the ideal line, indicating high predictive accuracy.

Figure ?? shows the training loss curve of the best MLP model. The loss decreases rapidly and stabilizes, indicating successful convergence.

## 5 Conclusion

In this experiment, we implemented a Multi-Layer Perceptron using PyTorch to predict bike sharing demand. By leveraging PyTorch's flexibility, we were able to train a custom neural network and optimize its hyperparameters. The MLP achieved a significantly lower RMSLE than the linear baseline, highlighting its ability to model complex, non-linear demand patterns. The use of standardization and log-transformation of the target variable were also crucial for the model's performance.

## References

- [1] Kaggle Bike Sharing Demand Competition. <https://www.kaggle.com/competitions/bike-sharing-demand>

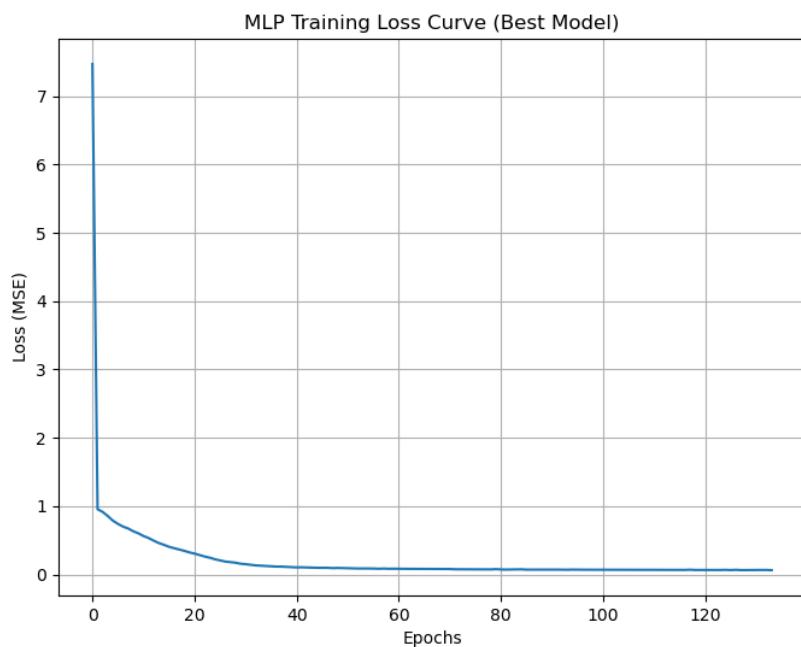


Figure 2: MLP Training Loss Curve

- [2] Paszke, A., et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." NeurIPS 2019.