Taxi Fare prediction using Neural Network

N Surya Prakash Reddy(20CS10038)

1 Introduction

The objective of this project is to predict taxi fares from previous data using a Multi Layer Perceptron Regressor.
SkLearn's MLPRegressor tool has been used as a benchmark.

2 Implementation

2.1 Pre Processing

The Dataset has been checked for invalid values and cleaned. Instead of using all the data given as is, there were some changes to get better inputs.
Instead of using the latitudes and longitudes directly as input to the neural network, we first convert them to travel distance since latitudes and longitudes are not really linearly related to fares. Fares with zero displacement are also removed since the dependence is only on date which doesn't make sense for prediction.

The date is split into two parts, time of the day and day of the week since a direct conversion to unix timestamp gives a large float.Time is normalized so that it doesn't have complete influence over the Z values(A.W).

2.2 Train Test Split

The splitting is done using numpy's seeded permutation utility to get a 80:20 split for train and test data.

2.3 Data Loader

Since the train test split already saves the train and test data, the mini batch loader returns just the indices of the mini batches to save memory.

2.4 Weight Initializer

We initialize the weights and biases layer by layer using the data and hidden layer sizes given while initializing.

The weights are a 2D array of shape (inp_layer_size, out_layer_size) to the next layer from the previous layer.

2.5 Forward Propagation

The activations are initialized with the given X value. Then for each layer, activations are calculated as activation_fn(A.W + B).
The prediction part is also a forward propagation step but instead of storing each layers activation data, we just use a single variable which finally houses the output.

2.6 Back Propagation

We use chain rule starting from the output layer to calculate the updated weights for each layer. We use L2 regularisation to reduce the weight increase so that no single weight holds too much influence over the results.
We also calculate the loss at every 10th epoch and plot both train and test losses.

2.7 Training

We use a mini batch SGD loop for training. In every epoch, we take a batch from the data loader and update the weights after a forward and back propagation step.

3 Results

3.1 Specification 1

For a model with one hidden layer of 32 neurons and logistic activation function, the r2 scores are:

3.1.1 Custom Implementation

- Train score = 0.571
- Test score = 0.548

The loss graph over 200 epochs is as follows

Figure 1: Loss Graph for case 1

3.1.2 SkLearn implementation

- Train score = 0.578
- Test score = 0.557

3.2 Specification 2

For a model with two hidden layers of 64, 32 neurons and rectified linear unit activation function, the r2 scores are:

3.2.1 Custom Implementation

- Train score = 0.536
- Test score = 0.514

The loss graph over 200 epochs is as follows

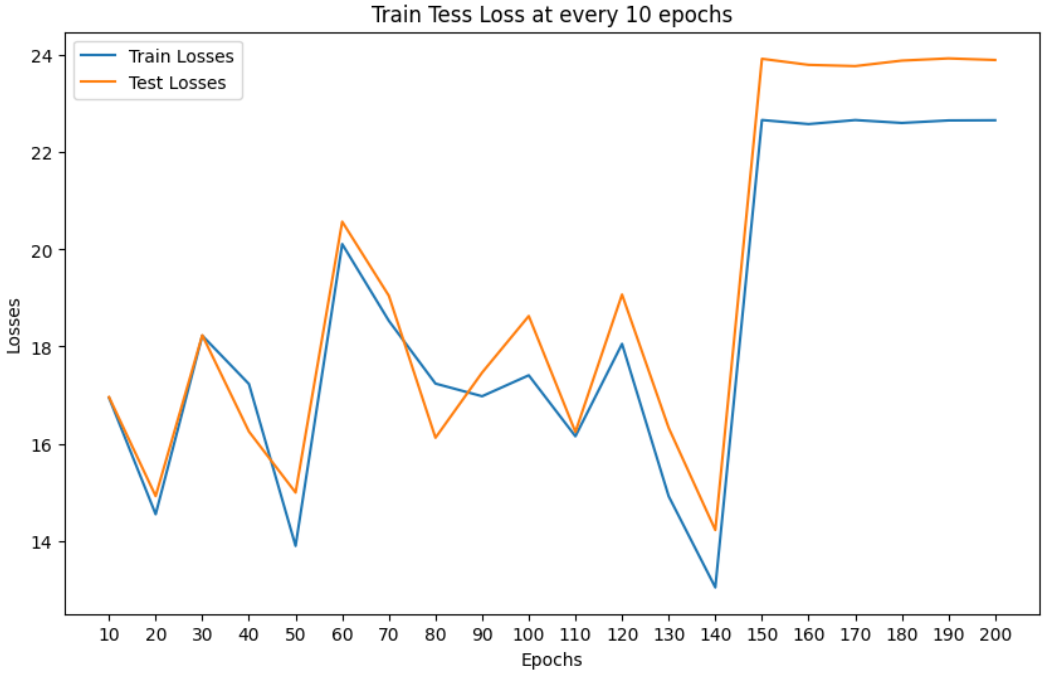

Figure 2: Loss Graph for case 2

3.2.2 SkLearn implementation

- Train score = -0.00027
- Test score = -0.00042