CSE 151B Project Milestone Report

Alex Zhang
acz001@ucsd.edu

Vivian Liu
v6liu@ucsd.edu

1 Task Description and Exploratory Analysis

1.1 Problem A

Knowing information about how long taxi rides are is crucial to improving traffic, since it will allow us to take steps to address traffic issues (i.e. service reliability, resource allocation, traffic management, efficient scheduling, and more). Therefore for this project, our task is to build a deep learning model that predicts trip time of taxi rides based on their (initial) partial trajectories with the lowest Root Mean Square Error (RMSE). This loss measures the average deviation between the predicted and actual values; it squares the differences, averages them, then takes the square root.

The dataset we used includes information about call type, origin call, origin stand, taxi id, timestamp, day type, and polyline trajectory of taxi rides in Porto, Portugal. With a model that can generalize to taxi rides and predict ride times, dispatch systems will better be able to assign drivers more efficiently to each pickup request. Our input of the model includes the aforementioned dataset on Porto taxi rides, and our output is the travel time of a given taxi trip.

1.2 Problem B

Our testing dataset has a size of (1710670, 9), so we have information on 1710670 taxi rides total. However, when combing through the data, we realized that there were 10 of them that were missing information. We decided to remove these rows since we didn't want to feed incomplete data to our model, which left us with 1710660 rides total to train on. We used the same dataset for training and validation but reserved some examples to save for validation (approximately 20% the size of the training set). Our testing dataset has a size of (327, 8), so 327 rides total.

While exploring our data, we used the figures below to help us determine how much each feature impacted the final result. Looking at each of these features, it appears that month, day, hour, and week all play a part in determining taxi time. For example, when analyzing the months graph we say that May has a higher frequency of longer trips since colors are shifted up a bit compared to the other months. Looking at the hour graph, it seems that taxi durations are shorter later in the day, and looking at the week, it seems like day 4 is relatively busy and day 6 tends to have shorter taxi durations. Looking at origin stand is also helpful, since durations for some are clearly longer than others.

We decided to use as much of the given data as possible for our input and take into account every factor that we can. This includes the latitude, longitude, time (we broke this down into year, month, day, hour, and weekday), call type, and day type. Because of all these features we were using, our input dimension was 9. For our output, we just write the estimated length of the trip. Our submission has two columns: trip ID and travel time, and it has 327 trips to estimate.

Preprint. Under review.

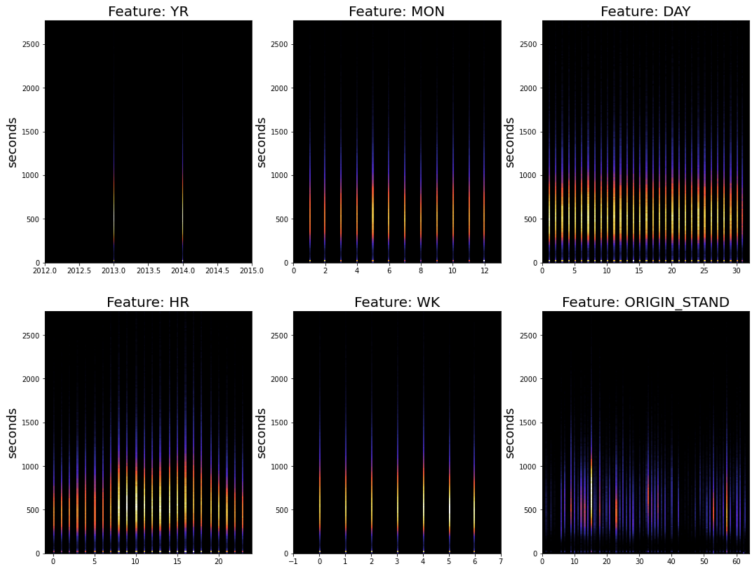

Figure 1: Histograms of different features.

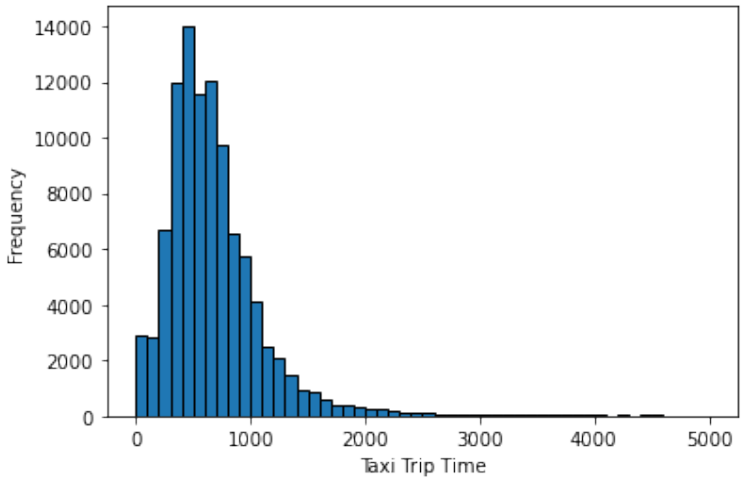

Figure 2: Distribution of travel time for all trips

2 Deep Learning Model and Experiment Design

2.1 Problem A

We used DataHub to write and edit our code and GPU (CUDA) to run it. For our optimizer, we used Adam since it is an adaptive optimization algorithm that dynamically adjusts learning rates based on gradient estimates, which leads to faster convergence during training. For our loss function, we used L1Loss, or Mean Absolute Error instead of Root Mean Squared Error since we noticed that there were a good amount of outliers in the data, and MAE penalizes outliers less.

To process our training data, we used a calculation on POLYLINE to find the total trip time as our training label. For the actual data, we added datetime features like month, year, day, and weekday weighted by the call type and day type of the taxi trip. For the other call types and day types, we set those to 0. Additionally, we added the latitude and longitude of the trips (if they existed). Otherwise, we set the latitude and longitude features to 0. The input we ultimately fed into the model has 44 features.

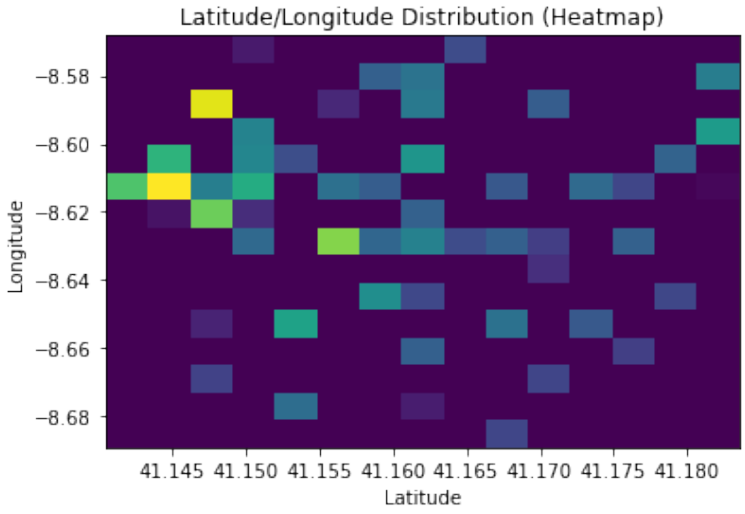

Figure 3: Distribution of trip position for all trips

For hyperparameters such as learning rate and learning rate decay, we tested different values until we found the ones that gave us the lowest loss on our validation set. We had a total of 50 epochs because that was enough for our model to converge, and we chose a batch size of 128 since the dataset is large. Additionally, since some of the data in the training set was missing, we just decided to ignore the missing ones altogether. Using these methods and parameters, it barely took any time to train all 50 epochs of our model, just a few seconds max.

2.2 Problem B

We started with a simple MLP model, then tried adding more linear layers to see if it would improve accuracy. Our final model included 5 linear layers with ReLU activation functions. We have one layer going from our input size to a size of 256, 3 internal layers with a size of 256, and a final layer with a size of 1. During this process, we experimented with things like feature engineering (i.e. extracting time data from TIMESTAMP and normalizing the latitude and longitude) and trying different losses. We have tried loss functions such as RMSE, cross-entropy, and L1 loss during training, and we found that trying to minimize L1 loss gave us the best results. We also tried training the model in different batches and experimenting with early stopping in case there's overfitting.

3 Experiment Results and Future Work

3.1 Problem A

Here are visualizations for our training and validation losses:

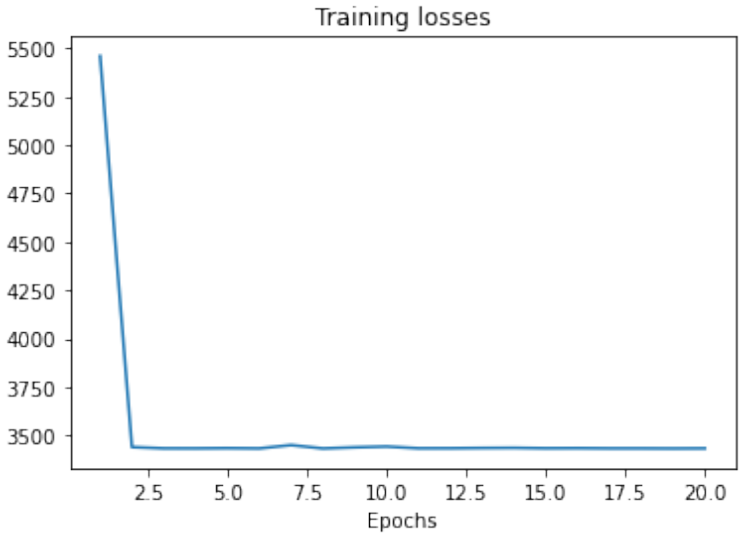

Figure 4: RMSE losses over training epochs.

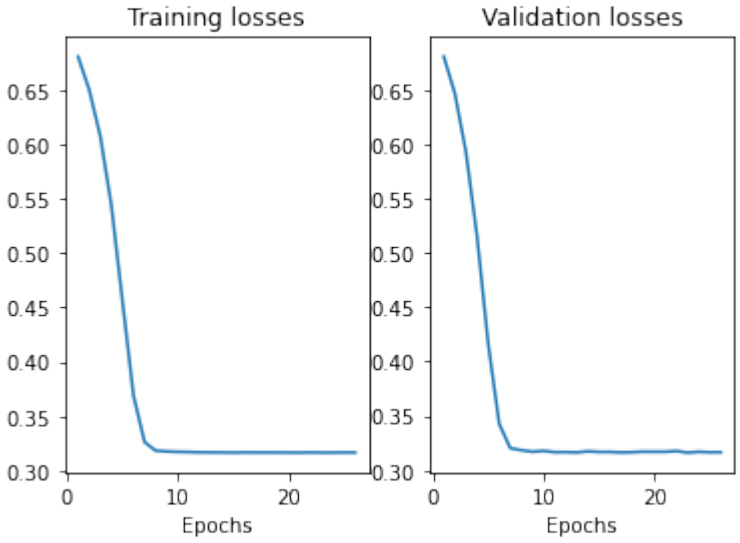

Figure 5: L1 (MAE) Losses over training epochs.

Our current rank on Kaggle is 44 with a test RMSE of 782.21556.

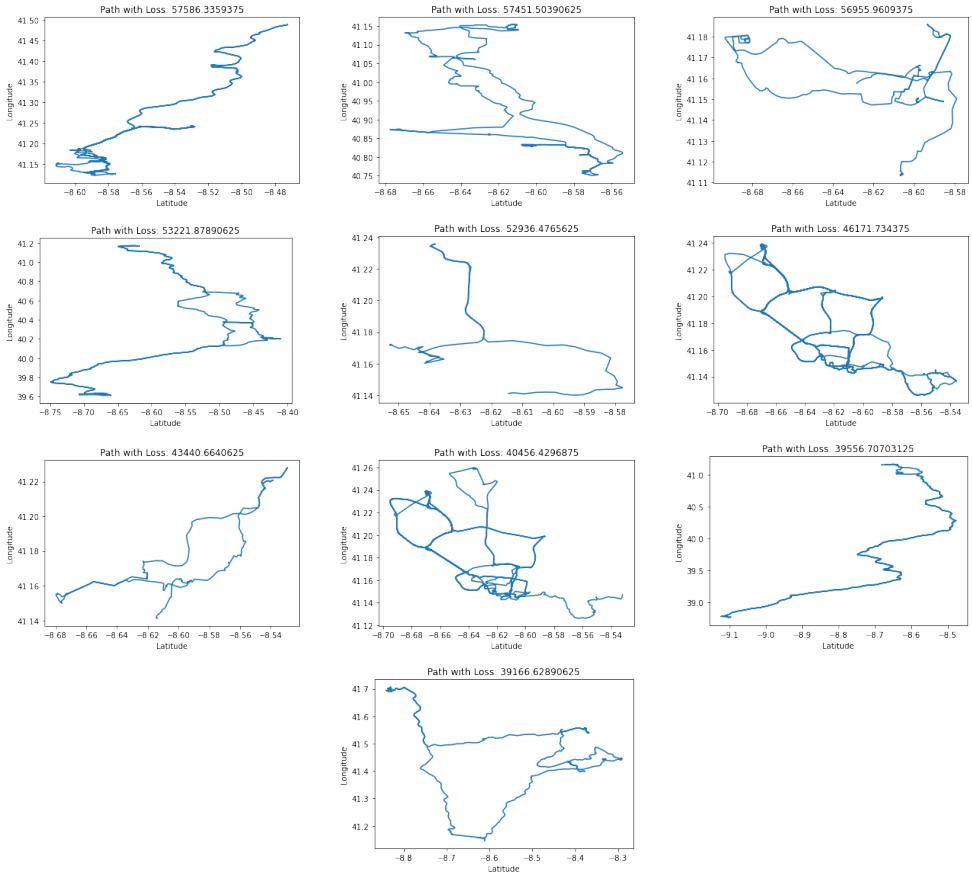

Figure 6: Our 10 training samples with the highest training loss.

It seems that our model is underfitting based on the data. In order to improve our performance, we will keep experimenting with different ways to feature-engineer our data, different model architectures, and different activation functions in the upcoming weeks. We will also do more testing to fine-tune our parameters and do more research such as looking into encoding categorical variables. Because the current way we add day types and call types results in a lot of zeroes in out input, we want to try a different way of representing our data that is less sparse.

Our GitHub Repo

https://github.com/Stuart6557/CSE151B-SP23-Kaggle-Competition

References

[1] Brownlee, Jason. "3 Ways to Encode Categorical Variables for Deep Learning." *MachineLearningMastery.Com*, 26 Aug. 2020, machinelearningmastery.com/how-to-prepare-categorical-data-for-deep-learning-in-python/.

[2] "Embeddings: Categorical Input Data | Machine Learning | Google for Developers." *Google*, 18 July 2022, developers.google.com/machine-learning/crash-course/embeddings/categorical-input-data.

[3] Pascalwhoop. "An Overview of Categorical Input Handling for Neural Networks." *Medium*, 19 July 2019, towardsdatascience.com/an-overview-of-categorical-input-handling-for-neural-networks-c172ba552dee.