CSE 151B Project Final Report

Yash Puneet ypuneet@ucsd.edu

Abstract

This paper explains the process behind the development of an initial deep-learning model for taxi trip duration prediction in Porto, Portugal. It summarizes the design process, feature engineering, and decisions made, as well as the experimentation done for model development and hyperparameter tuning. Code for the models described can be found at https://github.com/cse151b-sp23/final-yash.

1 Task description and background

1.1 Problem A

The project task is to generate a model to predict the travel time of a taxi trip in Porto, Portugal in order to help the central dispatch system determine approximately when to dispatch a particular taxi for another trip. This will minimize the number of taxis required to effectively cover all trips thus allowing more efficient trip scheduling to meet demand. This task is important for multiple reasons, some of which are listed below.

- Meeting Demand: Knowing the average trip time will allow the central dispatch to determine how many taxis would potentially be available to respond to a request in the next t time units. This will help determine whether demand could be met based on trends on similar days and times in the past (separate from this model) thus helping predict average wait times and whether enough taxis have been deployed to cover expected demand.
- Profit Increase: Following from the above point of knowing how many taxis could potentially be available at a point in time, a good guess as to how many taxis in total are required to be deployed at any point in time to meet demand. This would prevent the system from having too many taxis and drivers hired for particular time slots saving both fuel and human resource costs.
- Prevent Taxi Idle time: After a trip, taxis are generally empty for some time before being assigned to another trip. Having an idea of the trip duration will allow the central dispatch system to preemptively assign trips to taxis that should almost be done with their trip (based on duration prediction) thus reducing/eliminating this idle time and saving both fuel and human resources costs as above in addition to time.

Considering real-world examples where solving such a task could have a great impact, tasks related to route optimization and resource allocation are most likely.

- One such example could be Amazon, UPS, or other delivery services. Such services dispatch vehicles to deliver packages within a particular region in a timely manner. As formulated above, such a delivery task requires meeting demand since package deliveries may be planned or could be same-day delivery requests which would require more prompt scheduling. Similar to preventing idle time, this task would benefit from preventing delivery vehicles from making unnecessary trips when another vehicle could deliver that package at an extension of its route. All of this would contribute to the overall profit increase for the delivery company since fuel costs and delivery worker wage costs can be minimized

Preprint. Under review.

while effectively scheduling deliveries to maintain reputation and avoiding losses caused by delayed shipments.

- Another real-world example is cost decision optimization for vehicle renting agencies. This is useful since these agencies can predict how much a car will be used when rented (duration of trips taken) similar to how taxi trip duration could be predicted. Idle time would be good in this scenario as the company can reduce the cost of rental to be more competitive than others while still maintaining a high-profit margin.

1.2 Problem B

Below is a mathematical formulation of the data we have and the prediction task. We formulate 2 prediction tasks since the best-performing model so far makes a velocity prediction that feeds into the duration prediction.

Training Data:

$\mathbb{S}_1 = (a_i, b_i, c_i, d_i, e_i, y_i)_{i=1}^N$

Where:
- a_i: The hour of the day in which the trip started.
- b_i: The month of the year in which the trip started
- c_i: The week of the month in which the trip took place
- d_i: the day on which the trip took place
- e_i: the call type signifying where the taxi was hailed
- y_i: the velocity derived from the polyline

$\mathbb{S}_2 = (a_i, b_i, c_i, d_i, e_i, f_i, y_i)_{i=1}^N$

Where:
- a_i: The hour of the day in which the trip started.
- b_i: The month of the year in which the trip started
- c_i: The week of the month in which the trip took place
- d_i: the day on which the trip took place
- e_i: the call type signifying where the taxi was hailed
- f_i: the velocity predicted from the previous model
- y_i: the trip duration derived from the polyline

The Loss function for the model is Root Mean Square Error defined by:

$L = \sqrt{\sum_{i=1}^{n} \frac{(ypred - yactual)^2}{n}}$

The prediction task is divided into two steps. Firstly it is to minimize the RMSE loss in predicting the average velocity of the taxi for a particular trip. This is followed by appending velocity to the dataset to predict trip duration with minimal RMSE loss.

The summarized Mathematical Abstraction is thus:

Training Data:

Stage 1:

$\mathbb{S}_1 = (a_i, b_i, c_i, d_i, e_i, y_i)_{i=1}^N$

Stage 2:

$\mathbb{S}_2 = (a_i, b_i, c_i, d_i, e_i, f_i, y_i)_{i=1}^N$

Overall Model Class:

$f(x|w, b) = ReLU(w^T x - b)$

Loss Function:

$L = \sqrt{\sum_{i=1}^{n} \frac{(ypred - yactual)^2}{n}}$

Learning Objective:

$argmin_{w,b} \sum L(y_i, f(x_i|w, b))$

This mathematical abstraction can be used to solve multiple other tasks, even those that are not similar to the given taxi trip duration problem. The given abstraction is very similar to the standard multi-layer perceptron algorithm class definition apart from the two-stage model.

- A two-stage abstraction as defined above can be used to solve tasks such as predicting a student's graduation timeline. Using classes taken and other particulars like the major, year of starting, grades, etc., the first stage can predict an approximate class schedule based on the difficulty of classes a student would probably take together. The second stage can then group such classes along with requirements for graduation to determine a graduation timeline. The model class defined above can be used in the explained scenario since the graduation timeline prediction allows student class enrollment styles to be learned, and the aim is to minimize the difference between predictions and the actual value.
- Another possible example would be to use the model to predict continued lecture enrollment at a university. The two stages here would help since data such as class average grades, student feedback, and the professor could be used to predict the difficulty of a class during a particular quarter. This data can then be augmented into the dataset for the second stage in which the difficulty plays a significant role in class student retention and thus predicted enrollment by the end of the quarter.

2 Exploratory data analysis

2.1 Problem A

Preliminary analysis of the datasets revealed multiple useful features that helped determine how to use the data for training and how this would factor into predictions being made by the model later on.

Starting with the size and dimensions of the sets we have the following information:

The training set has dimensions 1710670x9 which means that there are 1710670 rows and 9 columns. With these dimensions, the training dataset provides data about 1710670 taxi trips, each with 9 attributes. The test set was found to have dimensions 320x8 which means that there are 320 trips to generate duration predictions for the model testing.

These attributes are as follows:

- TRIP ID: This provides a unique identifier for the data. This will not be used in the model since it doesn't provide any information that is useful for making predictions on trip duration
- CALL TYPE: This provides information about how the taxi request was made. There are three possible values for this attribute making it categorical data that will be used in our model. This data is included in a model since how the taxi is requested could affect the trip duration. This effect may be in the form of a potential duration reduction if the taxi is hailed from a taxi stand when compared to other methods since the time for the taxi to drive to the passenger is not included.
- ORIGIN CALL: This field associates a phone number or identifier to record who made the call. This field is one of the 2 fields found to have NULL-valued data. The field was not used since, when deployed, the identifiers for individuals in the training set may not cover the set of all users so the attribute would not be helpful for any predictor model.
- ORIGIN STAND: This field is the second field that was found to contain NULL-valued data. It contains the identifier from the taxi stand the taxi was dispatched from. This field is one I experimented with including or removing for each model I try to see how it affects the loss and all NULL values are converted to 0s for the purpose of using it for training and prediction generation. This field warranted experimentation since it seemed like the set of taxi sands had significant overlaps but I was unsure of how this might affect new stands that have never been seen by the model.
- The Taxi ID is another field I experimented with since some taxis could be generally slower or faster than others. The attribute provides a unique identifier for each taxi. The attribute

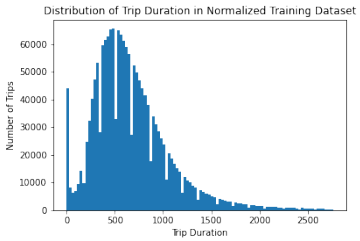

Figure 1: Trip time distribution.

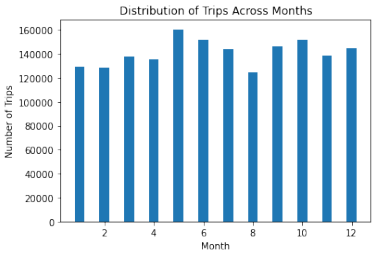

Figure 2: Trip month distribution.

was detrimental since in deployment, taxi IDs outside the 420 used cased large loss values as the model struggled to interpret these new taxi ids.

- TIMESTAMP This is one of the more important data attributes presented in the data as it provides information about the exact time of the trip start. This is split into the Year, Month, and hour primarily as taxi speeds may be different based on these factors since there may be variations in weather, road, and traffic conditions.
- DAY TYPE This is another categorical data attribute that determines whether the day is a holiday, holiday eve, or a normal day. This is an important attribute to determine the trip duration, however, could not be used in the model since all training set data points have the same 'normal' day type.
- MISSING DATA This field identifies whether a data point has missing attribute values
- POLYLINE: This is an attribute that was found to be unique to the training set and was not found in the test set as it was highly relevant to the prediction task. This attribute contains a list of longitude and latitude GPS coordinates taken at 15-second intervals. This is initially used by the model to determine duration labels for training. An extension to the model then used these to determine the velocity of taxis and build a precursory model that predicts velocity for use in the model to predict travel time.

After looking through the data and provided attributes, the data can be normalized to remove outliers that may skew the data. These would be extremely high and extremely low trip durations. After this, the distribution of data can be effectively visualized.

The distribution of trip time, the label for the training set, can be seen in Figure 1. As seen, the distribution is more heavily concentrated around the 500-600 range, which is towards the left of the distribution plot. The distribution reveals that there are some long trips (even after clipping the dataset) though there aren't as many outlying short trips. Moreover, trips are seen to mostly lie in the 500-600 range with a large portion of trips being within the 250-1000 range.

Looking at the month (Figure 2), day (Figure 3), and hour (Figure 4) distributions, we see that these did not have major differences, though differences did exist. This part of the data reveals that trip counts did vary depending on the time, day, and month however this difference was not very significant. For example, if we look at the month-trip count distribution, the number of trips for each month can be seen to be very close based on the bar chart. However, a difference in trip timing may still be valuable to explore since the y-axis scale is separated by 20,000. The same is true for the day and hour distributions.

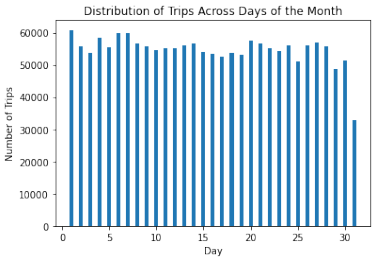

Figure 3: Trip day distribution.

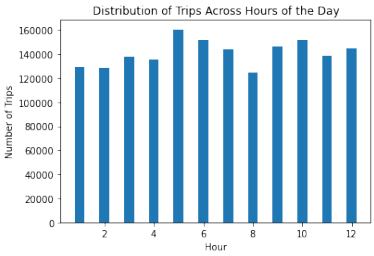

Figure 4: Trip hour distribution.

The polyline (location heatmap distributions) are represented using the starting and ending positions. The starting position distribution is shown in Figure 5 and the ending position distribution is shown in Figure 6.

Another interesting distribution to look at was the distribution of velocities based on the polyline coordinates. This helped clear out further outlier data and provided another input attribute to the model to predict trip duration as a taxi's velocity would directly relate to this. This distribution is seen in Figure 7.

One sample for this data is shown in Figure 8. This is taken from the provided data and does not reflect the velocity data that is generated from the polylines in the training set and is generated by the first stage model for the testing set. Moreover, Figure 9 presents one sample from the processed dataset that splits timestep into its constituent parts and replaces all N/A values with 0.

2.2 Problem B

Splitting the provided data into a training and validation set was a challenging problem for me since I felt that the style of models I would be designing would require a large amount of data to properly converge. This led me to split the data such that 80% of the data was used for training and the remaining 20% of the data was used for validation To avoid bias, I also used an interleaved data split such that while going through the data, one row would be taken for the validation set and 4 rows were

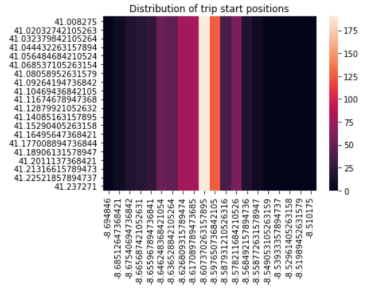

Figure 5: Trip starting location distribution.

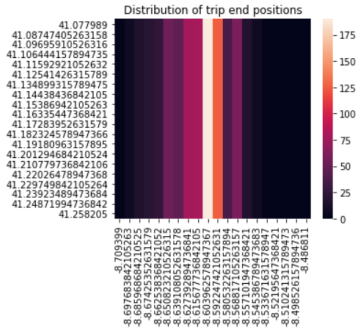

Figure 6: Trip destination distribution.

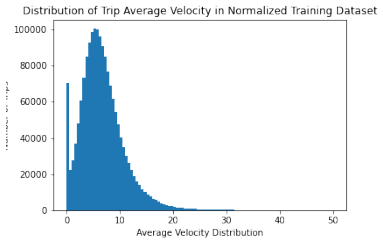

Figure 7: Trip velocity distribution.

	TRIP_ID	CALL_TYPE	ORIGIN_CALL	ORIGIN_STAND	TAXI_ID	TIMESTAMP	DAY_TYPE	MISSING_DATA	POLYLINE
0	1372636858620000589	C	NaN	NaN	20000589	1372636858	A	False	[[-8.618643,41.141412],[-8.618499,41.141376],[...
1	1372637303620000596	B	NaN	7.0	20000596	1372637303	A	False	[[-8.639847,41.159826],[-8.640351,41.159871],[...
2	1372636951620000320	C	NaN	NaN	20000320	1372636951	A	False	[[-8.612964,41.140359],[-8.613378,41.14035],[-...
3	1372636854620000520	C	NaN	NaN	20000520	1372636854	A	False	[[-8.574678,41.151951],[-8.574705,41.151942],[...
4	1372637091620000337	C	NaN	NaN	20000337	1372637091	A	False	[[-8.645994,41.18049],[-8.645949,41.180517],[-...

Figure 8: Input dataset sample.

```
trimmed_taxi_data.head()
```

Out[74]:

	CALL_TYPE	ORIGIN_STAND	TAXI_ID	TIMESTAMP	POLYLINE	LEN	YR	MON	DAY	HR	WK
0	C	NaN	20000589	1372636858	[[-8.618643,41.141412],[-8.618499,41.141376],[...	330	2013	7	1	0	0
1	B	7.0	20000596	1372637303	[[-8.639847,41.159826],[-8.640351,41.159871],[...	270	2013	7	1	0	0
2	C	NaN	20000320	1372636951	[[-8.612964,41.140359],[-8.613378,41.14035],[-...	960	2013	7	1	0	0
3	C	NaN	20000520	1372636854	[[-8.574678,41.151951],[-8.574705,41.151942],[...	630	2013	7	1	0	0
4	C	NaN	20000337	1372637091	[[-8.645994,41.18049],[-8.645949,41.180517],[-...	420	2013	7	1	0	0

In [75]:
```
trimmed_velo.head()
```

Out[75]:

	velocity
0	11.937889
1	14.555896
2	11.808893
3	5.345865
4	10.378641

Figure 9: Processed dataset sample.

taken for the training set. This interleaved approach ensured that the validation data wasn't comprised of just one category (a month-based category in this case) which does not reflect the general dataset as a whole. Moreover, this also ensured that the training data does not lose one entire category of data thus making it less generalizable.

Feature Engineering was a major part of my project design and can be summarized in four major Feature Engineering through data processing methods as well as a creation of a new feature using a preliminary prediction model. The feature Engineering methods used are:

- Imputation: For my model, imputation was done to handle missing data in the overall dataset and in the origin stand column. For the overall dataset, all entries with missing data were removed to effectively trim the dataset. This missing data was identified by the dataset itself signifying this in its MISSING DATA column. To deal with null values in the origin stand, two approaches were experimented with, first, all null values were replaced with 0s, and in the second experiment, they were replaced with the most common taxi stand. The second approach yielded better results in most cases but not always since this over-saturated the most common taxi stand thus the 0 approach was adopted for the final model submission.
- Categorical Encoding: The CALL TYPE column was categorically encoded to avoid issues with the model struggling to interpret the letters used to describe categories in the attribute. This used a simple dictionary approach to replace the letters with unique numbers in a loop.
- Feature Splitting: This was performed on the TIMESTAMP to split it into its constituent parts, Year, Month, Week, Day, and Hour. Though the distribution for some of these was not very useful, specifically the year as seen above, the data provided more features that split the data across to reach better loss values and convergence.
- Outlier Handling: This was mainly performed on Velocity and duration columns to trim the dataset further. Exploratory analysis revealed that these two fields had highly left-skewed data because some trips were unreasonably long or fast. Further investigation revealed that, since these both were primarily derived from the POLYLINE column, errors there could have caused this detrimental distribution, thus, the dataset was clipped at 3 standard deviations from the mean for both of these fields
- Feature Creation: This was done to add an additional feature, velocity, to the model which could be used to determine trip durations. Velocity has a direct correlation to the duration of a journey so I determined that using this as an additional feature would be helpful. This feature was created by first finding average velocity using the displacement of consecutive Polyline coordinates and then using this to train a preliminary model which then predicts this information for the test set that doesn't provide polyline coordinates. The distribution of this feature was seen in the previous section as well.

One-Dimensional Batch Normalization layers were mainly used across model experiments for my project and the final scheme settled on two possible variations. One Variation had a normalization layer after 2 fully connected layers and a second variation had it after every internal hidden layer, that is, after layers 1, 2, and 3. The best-performing variation after a few runs seemed to be the second variation providing consistently lower training losses but only by a small amount (RMSE 40-50). The rationale behind this approach was to limit skewed data from entering any layer. With normalizations in these internal layers, the data input for each one was centered around a point and removed extreme outliers. Moreover, the initial dataset was trimmed based on the mean and standard deviation of the trip duration and velocity columns thus acting as normalization of training data but not the model itself as these fields are not available in the testing data and served as targets.

CALL TYPE was used in the model I design and it was possible to use this data due to Categorical Encoding as described in the Feature Engineering strategy descriptions. The rationale behind using this column was that different call types could result in different aspects contributing to the duration of the trip. If a taxi needs to drive to pick up a passenger from a requested location, the duration of this trip will be a fixed amount higher than if the taxi was hailed from a nearby taxi stand. One other way this data could have been used would be to split the data set by call type and have different models for each one which is then combined into a final result through an aggregating layer. This approach was not attempted due to time constraints but would be interesting to consider for future experimentation.

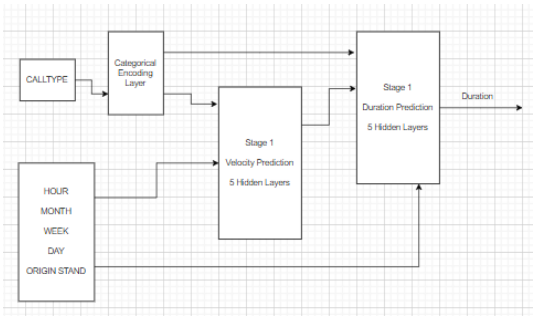

Figure 10: Final Model Architecture and Pipeline.

3 Deep learning model and experiment design

3.1 Problem A

The provided features used for the prediction task were TIMESTAMP split into the hour, month week, day; the origin stand, and the call type.

The deep learning pipeline can be seen in a simple form in Figure 10.

First, feature engineering is done as extensively described in the previous sections. Specifically, TIMESTAMP data is split into its constituent parts, and CALL TYPE Data is categorically encoded. This data is then packaged into a tensor containing hour, month, week, day, origin stand, and call type data.

Next, this tensor is passed into the first stage model (the velocity prediction model), which allows for the creation of a new feature, the velocity of the taxi. This feature is packaged into a tensor with all the features from the previous stage and passed into the final model used to predict the trip duration.

Both models have the same architecture and internal layer dimensions. They only differ by input dimensions since the duration prediction model has one extra feature. Inside each model, the data tensor is passed through a series of 5 hidden fully connected layers with a 0.5 dropout and ReLU activation function. After each of the first three layers, the data is also sent through a one-dimension batch normalization before entering the next layer. The output is then generated through a ReLU activation layer at the end of the model. The layer dimensions first extrapolated the input dimensions to 128 followed by the next few layers bringing this down to 64 then 16 then 8 and finally an output dimension of 1.

Many alternatives were used for prediction and their resultant loss values will be summarised in the next section. As for learnings and observations, each model contributed a key takeaway that I used for the iterative development of my model architecture to finally arrive at this final architecture. The first model I tried was a simple multiple linear regression model which provided a very high training loss of, on average, 1382 RMSE. This model revealed to me that deep learning models would be required for this task leading me to Multi-Layer Perceptron models. The first model architecture I tried was a single-stage model with 1-2 hidden layers. This shallow model had a training RMSE of 590 and revealed to me that a higher number of layers could be used for a deeper model as overfitting was not an issue yet and the model was struggling to converge. The next model remained mostly the same, however, I experimented with 3-8 layers finding that overfitting started becoming an issue after 5 layers thus this was the optimum layer count. I then moved onto a two-stage model but with lower dimensional extrapolation going to a maximum of 20. This performed better but led me to realize that higher dimensions could be used to better separate the data leading to my final model architecture.

The model functions used for training were as follows. The optimizer was an Adam Optimizer as it performed slightly better than SGD with a learning rate of 0.0001. The loss function used was Root Mean Square Error as this was used in the competition for evaluation.

Table 1: Model description table

Model	Description
Multiple Linear Regression	6 Hyper-plane separation of data for prediction
Shallow Fully Connected	Single stage model using ReLU Activation
Deep Fully Connected	Single stage model using ReLU Activation
2 Stage Low Dimensions (20)	2 stage model with ReLU activation and max dimension of 20
2 Stage High Dimensions (128)	2 stage model with ReLU activation and max dimension of 128

Table 2: Model summary table

Model	Hidden Layers	Dropout	Diagram	Optimizer	Learning Rate
Multiple Linear Regression	N/A	N/A	N/a	N/A	N/A
Shallow Fully Connected	1-2	0.3	Figure 11	SGD	0.001
Deep Fully Connected	3-8	0.2	Figure 12	SGD	0.001
2 Stage Low Dimensions (20)	5	0.5	Figure 10	Adam	0.00001
2 Stage High Dimensions (128)	5	0.5	Figure 10	Adam	0.00001

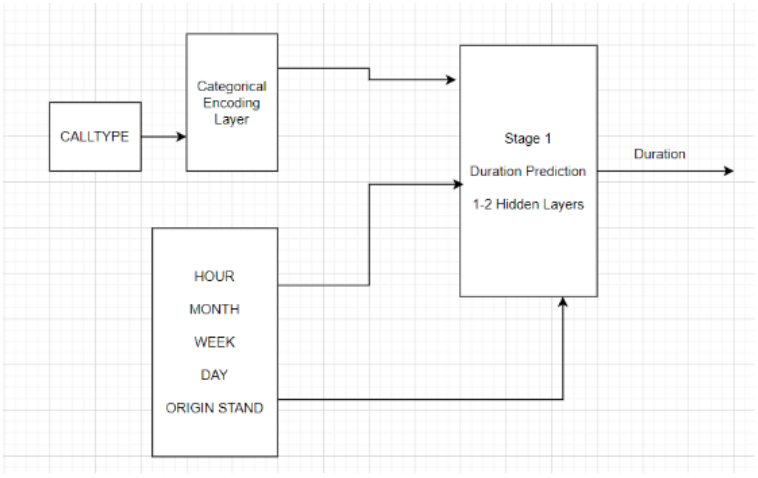

Figure 11: Single Stage Shallow Architecture.

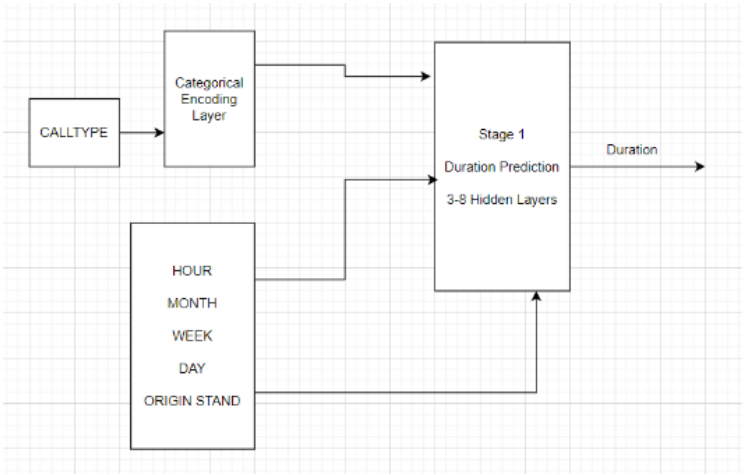

Figure 12: Single stage deep architecture.

3.2 Problem B

The major models I used for experimentation are summarized in Tables 1 and 2 with references to their architectures summarized in figures.

For Normalization techniques, I mainly used 1 dimensional Batch Normalization and Dropout to avoid overfitting to the training data. Batch Normalization was used since there was a clear left skew in the provided input data and some semblance of the skew remained in the cleaned and trimmed data set. The Batchnormalization was added to avoid skew from becoming too pronounced as they pass through the hidden layers in the model as this skew may be characteristic to the training data and not a general case. Dropout layers were added to reduce the number of activated neurons as, with a 5-layer model with 128-dimensional extrapolation, the risk of overfitting was high, and using a 0.5 dropout reduced this risk.

4 Experiment design and results

4.1 Problem A

Training and testing for the model were done on the UCSD DataHub Platform using GPU resources to speed up training. This is because the designs experimented with used deep fully-connected layers which require significant hyperparameter tuning for optimal performance and to prevent overfitting. Speeding up training would allow for more time for experimentation with these parameters and model architectures. Additionally, some models designed and experimented with used one fully-connected layer model predicting velocity and feeding that prediction into another fully connected layer, both of which required training. This effectively doubled the length of training required and the amount of hyperparameter tuning.

Two optimizers were experimented with for all models designed and used for the experiments. These were the Adam optimizer and Stochastic Gradient Descent. Both have their advantages ad drawbacks, in theory, thus analyzing model performance for both optimizers helped decide which ones would be optimal for the purpose of predicting trip duration using the provided data. Based on research, it seems that the Adam optimizer converges faster than SGD and this was seen as the training loss values per epoch reduced quickly before somewhat stabilizing. This required much fewer epochs to train to convergence, however, the model did not perform as well with test data suggesting that it was converging but overfit to the training data. When training the same model with SGD, the model took much longer to train with training loss reducing very slowly. However, the model performed more consistently (training loss was higher than Adam but testing loss was closer to training loss for SGD rather than being much larger). This meant that improving the model would reflect on testing performance more closely for architecture types similar to the ones experimented with so far and thus I chose to use Stochastic Gradient Descent as the optimizer.

When it came to parameter tuning, the process adopted took the longest time out of all the components of designing the model. The major parameters considered were the dropout percentages, learning rate, and batch size. Out of these, the hyperparameter experimented with the most was dropout percentage since there were many different dropout layers at different depths in the model, and finding an optimal or close to optimal combination and placement took a significant number of models trained. After each experiment, the loss values were noted and compared across models for loss decay and final loss value. Currently, the best-predicting model does not have the lowest final loss as this model was likely overfitting to training data. The best model instead has a comparatively low RMSE Loss, however, the most important aspect was the loss decay over epochs. Learning rate and batch size did not seem to have significant impacts on model performance apart from the fact that a larger learning rate resulted in quite a few predictions being 0 thus significantly increasing the loss. This was adjusted until this stopped being an issue and values were more reasonable trip duration times.

To train the model, experiments used ranges of epochs between 10-25 epochs initially based on the optimizer. This was increased to 20 epochs for the velocity model and 50 for the duration model after some parameter tuning had been completed for a more thorough investigation. The range of epochs was also used to determine whether the model was actually converging by testing larger than required epoch values to see if the loss values stop reducing significantly. The batch size used is 64.

Table 3: Model performance table

Model	Training RMSE	Testing RMSE	Diagram	Training Time
Multiple Linear Regression	1328	N/A	N/a	10 mins
Shallow Fully Connected	590	850	Figure 11	40 mins
Deep Fully Connected	360	820	Figure 12	2-5 hrs
2 Stage Low Dimensions (20)	350	805	Figure 10	2hrs
2 Stage High Dimensions (128)	216	786	Figure 10	2.4hrs

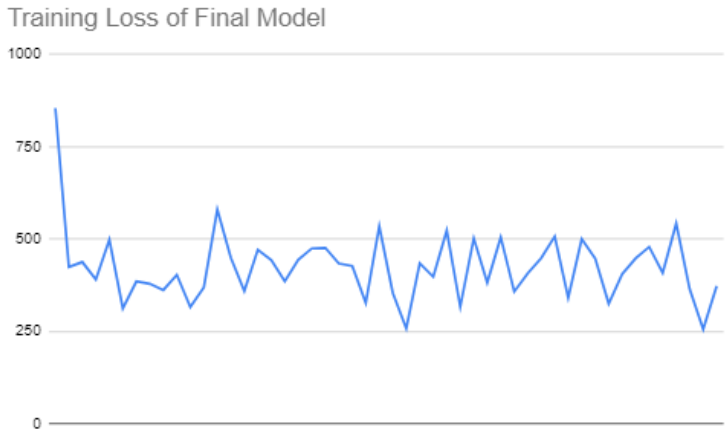

Figure 13: Final model RMSE training loss exponential decay.

The newest model takes a compound training time of approximately 5 minutes per epoch to train where I am adding the time of for each epoch of the two models used in conjunction to predict trip duration. Over the experiments, epoch training times have been in the range of 2 minutes to 17 minutes.

4.2 Problem B

During the experimentation phase, multiple models have been tried. Performance metrics for the 5 models discussed above can be found in table 3.

Though training times remained high, the best way I found to reduce training time was to trim the dataset further and to increase the dropout percentage. This resulted in slightly faster training times but did not provide a very significant boost.

4.3 Problem C

My current ranking on the Kaggle leaderboard is 82nd with an RMSE Loss of 786.2.

Figure 13 represents the best training loss curve achieved using 50 epochs. This was achieved using the model showcased in Figure 10 which uses Adam as the optimizer function. There are still a few overfitting problems that occurred with this model.

5 Discussion and future work

5.1 Problem A

After all the experimentation, data exploration, and feature engineering done in the past few weeks, I realized that a lot is still to be learned from the provided dataset. The distribution plots reveal significant portions of data seen in the velocity and trip duration data are at 0 which causes the model to skew to the left. The feature engineering done removed missing data and outlying high trip durations and velocities, however, this did not eliminate all the 0s. Exploration still needs to be done

to determine the root cause of these 0s and remove them from the velocity predictions. Overall, I believe that the best feature engineering strategy so far was creating the velocity feature since it was a unique idea across teams and helped boost my model architecture's performance. With more work and tuning, this feature could provide much better results.

The latest model has the potential to perform well, however, it is prone to overfitting. Experimenting with this model provided insight into the importance of hyperparameter tuning experimentation with layer dimensions. This experimentation is one of the key aspects of future work to be done with this model.

In terms of architecture types, the provided dataset does not provide any spatial or temporal locality and thus CNNs and RNNs would not be useful in this scenario. I would be interested in focusing on tuning multi-layer perceptron-based models and performing additional feature engineering to improve performance and structure data such that it could be used in LSTMs as many other teams did.

For the overall process, I found the biggest bottleneck to be training times. With each model's training taking multiple hours, it was hard to find time to extensively investigate these models and tune their parameters. This is one of the reasons why I focused a lot more on feature engineering as experimenting with that was a lot more feasible due to time and resource constraints. With more resources, I would definitely be interested in exploring more complex model types and stackings. I have a keen interest in human and animal cognition which I have incorporated in Robotics research in the past. Given resources and time, I would investigate natural cognition models such as the dual process theory and short-circuiting memory to improve performance. I would also experiment a lot more with LSTMs since they fit this goal and area of research as well.

For a beginner, I would suggest starting early and discussing ideas with others as much as possible, Working on this project alone, I often struggled to find the flaws in my ideas. Talking them through with others and the instructional staff truly helped me deepen my understanding of deep learning networks and improve my models during the course of the competition.