CSE 151B Project Milestone Report

Zairan Xiang, zaxiang@ucsd.edu **Daniel Son, dson@ucsd.edu**

1 Task Description and Exploratory Analysis

1.1 Problem A

• The task is to predict future 6 seconds trajectories for autonomous vehicles based on the movement in the past five seconds. Especially with the increase in use for autonomous vehicles, this area of study is essential to ensure people's safety by accurately predicting collision-free paths given the surrounding environment. Since autonomous vehicles are mainly electric, increasing the safety of these features will also lead to more consumers purchasing these clean energy vehicles, which in turn will have environmental benefits through reduced CO_2 emission.
• This is a regression problem as we have the x and y positions for each of the five seconds in the input and our goal is to predict what position the vehicle should go next. The input is a 5 seconds sequence $(x_1, x_2, x_3, x_4, x_5)$, output is a 6 seconds sequence $(y_1, y_2, y_3, y_4, y_5, y_6)$, and RNN is trying to estimate the probability P($y_1, y_2, y_3, y_4, y_5, y_6 \mid x_1, x_2, x_3, x_4, x_5$), and at each time step t, RNN is trying to estimate the probability P($y_t \mid y_{t-1}, ..., y_1$).

1.2 Problem B

Number of train/test trajectories in each city:
• Austin – train: 43041 test: 6325
• Miami – train: 55029 test:7971
• Pittsburgh – train: 43544 test: 6361
• Dearborn – train: 24465 test: 3671
• Washington D.C. – train: 25744 test: 3829
• Palo Alto – train: 11993 test:1686
Also shown below in the bar chart in Figure 1:

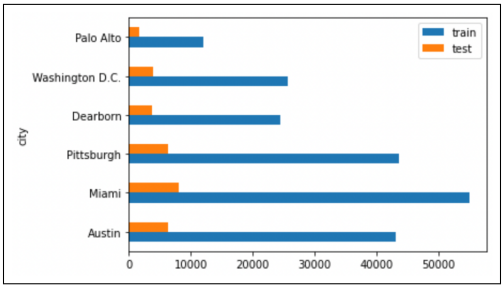


Figure 1: Number of Train and Test Trajectories by City

There are 50 points in each trajectories input, and 60 points for each output.
total train trajectories: 203816, and total test trajectories: 29843.
Two dimensions in both inputs/outputs: one for x position and one for y position. so we have 50 x 2 for each input trajectory and 60 x 2 for each output trajectory.

Preprint. Under review.

• The distribution of input and output positions for all agents is shown below in Figure 2:

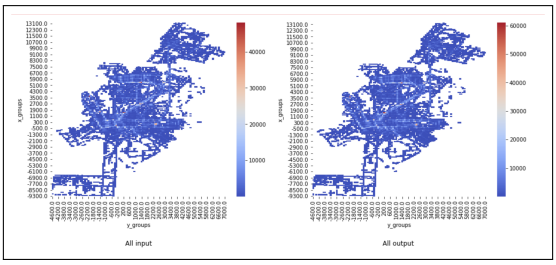
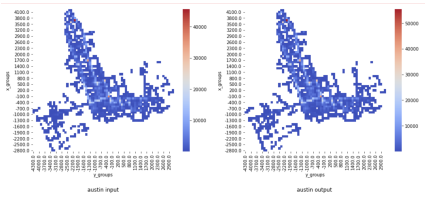


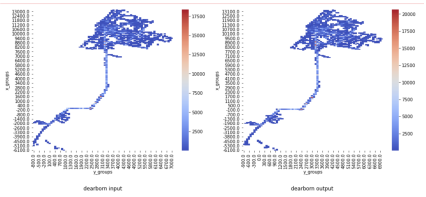
Figure 2: Distribution of Input and Output for All Agents

From Figure 2, We can see that the motions are centered with the high-frequency positions are mostly at the center of heatmaps. We can also see some patterns in the distributions that might indicate potential trajectory with high frequency. Another interesting finding is that the input and output distributions are nearly the same.

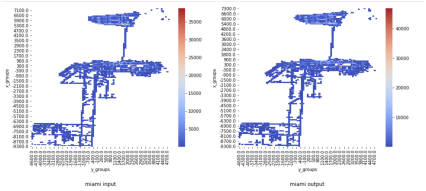
• We then get the distributions of input/output positions for each city in Figure 3.



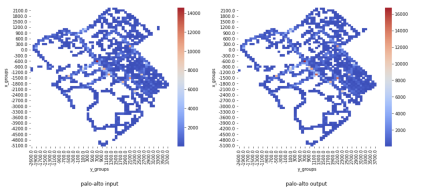
(a) Austin



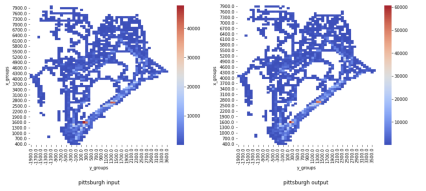
(b) Dearborn



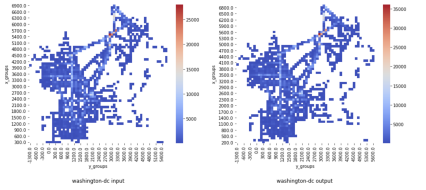
(c) Miami



(d) Palo Alto



(e) Pittsburgh



(f) Washington D.C.

Figure 3: Distribution of Input and Output for each city

From the above distributions, we can see some clear patterns in each city as well as the difference of distributions among cities, and the input / output distributions for each city are also nearly the same as we observed in Figure 2.

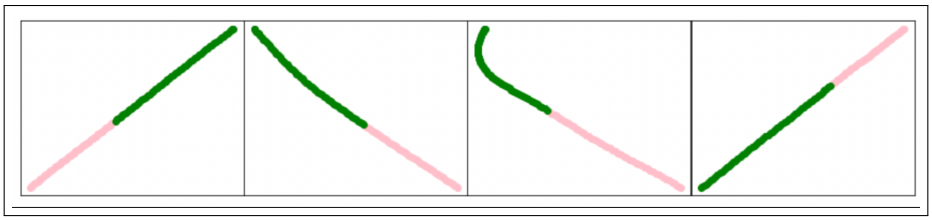


Figure 4: Trajectory Predictions for Sampled Batch

2 Deep Learning Model and Experiment Design

2.1 Problem A

For the training and testing of our model, we utilized the complementary UCSD DataHub to get access to more powerful machines with 1 GPU, 8 CPU, 16G RAM. The training and testing of the model was ran on the GPU due to it being much faster than running it on our own machine's CPU.

The final optimizer we chose for our model is the Adam algorithm from the torch.optim package. We tried Adagrad, AdamW, Average Stochastic Gradient Descent, Resilient Backpropogation, and Stochastic Gradient Descent, but from our experiments, the Adam algorithm consistently found the predictions with the smallest error. For the learning rate, we attempted to implement a self tuning learning rate updater. It did not work on the current model, and we will implement it in the final model since having an accurate learning rate is essential for the model to efficiently learn the most accurate prediction. The initial learning rate was set at 1e-3. During testing we changed the rate to be larger when the model we were testing was taking a long time to learn the data and smaller if the errors kept oscillating. In the end we found that a learning rate of 1e-3 provided a good balance between overshooting and undershooting for finding the prediction with the least error.

To make the prediction for each target agent, we utilized an encoder and decoder structure, which gives us the freedom to choose what model we want to use in the decoder phase. With this flexibility, it allows for more possibilities to implement the model which was very useful when testing difference models. The model that produced the best results so far was the one based on linear regression with four layers in each encoder and decoder. The activation function used after each linear regression to give it non-linearity was ReLU.

For the city information, we decided to evaluate each city separately due to different driving behaviors. For example, in more urban cities, there will be more obstacles that have to be taken account of in the prediction due to there being more people and structures being more densely packed. On the other hand, less urban cities will have less obstacles to navigate around and therefore straighter trajectories.

The batch size we chose was 4. We tried various batch sizes and found if we made it too large (around 1000) the training error was higher, and increasing it by a smaller amount did not improve decrease the error significantly. Due to this we kept the model simpler with a batch size of 4. The number of epochs conducted was 20. We found that around 20 epochs, the errors converged, so we stopped there since it was unlikely that the model would be able to learn anything new to produce a smaller error. Each epoch took around 20 seconds for the linear model. When we tested variations of a LSTM model, it took between 30 seconds to 5 minutes per epoch depending on the complexity of the model, which was much longer on average compared to the linear model.

2.2 Problem B

The model that we spend the most time testing was one that utilized a LSTM in the encoder and decoder. We were unsuccessful in our attempts due to the training error converging at best: 40 million on average for each city. Things that we have tried modifying in the LSTM model were the number of hidden layers, optimizer, activation functions, number of layers in the encoder and decoder, batch size, error metric, embedding layer, dropout layer, and transformations of the data.

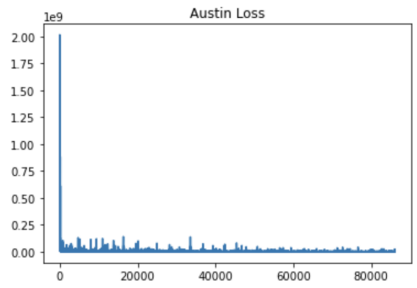
The model that had the best error for us was a linear regression model which, in comparison to the LSTM model, had approximately a 10,000 error on average for each city, making it significantly better. Due to this, we decided to work with the linear model for the milestone report. The linear

model includes an encoder and decoder state, both of which include 4 linear layers accompanied by a ReLU activation function after each. Each layer applies a linear transformation to the incoming data: $y = xW^T - b$, and each ReLU activation applies the rectified linear unit function element-wise to the incoming data. All of these specified operations are run in sequence using nn.Sequential() from the PyTorch package.

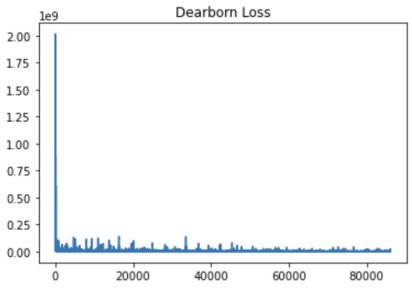
3 Experiment Result and Future Work

3.1 Problem A

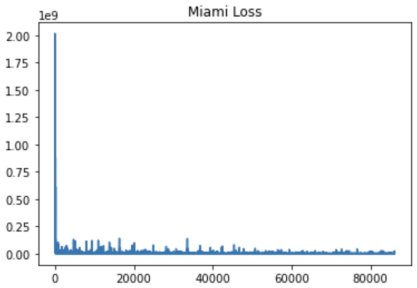
• The training loss (RMSE) value over training steps for each city:



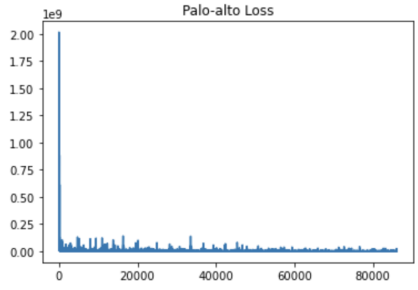
(a) Austin



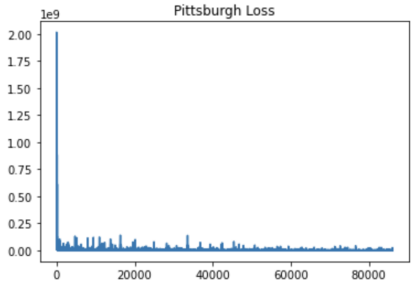
(b) Dearborn



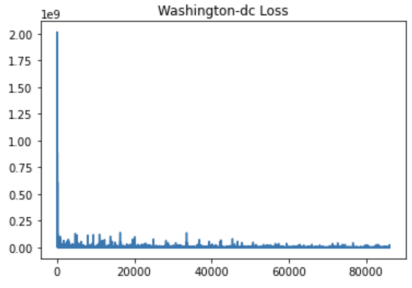
(c) Miami



(d) Palo Alto



(e) Pittsburgh



(f) Washington D.C.

Figure 5: Linear Model RMSE over training steps for each city

In above Figure 5, we can see that the training loss (RSME) over the training steps is indeed decaying exponentially for each city (we have 6 separate plots here because we train on each city separately).

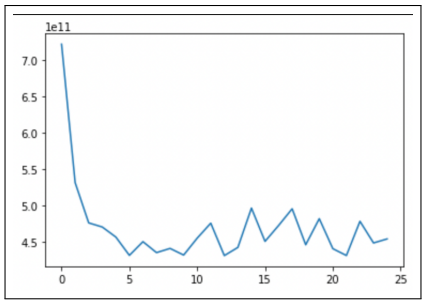


Figure 6: LSTM Model's Training Loss RMSE Over Training Steps

• In comparison to the loss from our linear model as illustrated in Figure 5, Figure 6 shows the training loss from our LSTM model. As shown by the y-axis, the loss from the LSTM model is much higher than the loss from the linear model. As the LSTM model goes through more epochs, we can see that the error starts oscillating. To fix this, we plan to implement an adaptive learning rate so the model will not overshoot the minimum error due to the learning rate being too large and also plan to implement batch normalization to reduce the spread of the data which can result in the model being better able to find the most optimal solution.

• Randomly sample a one training sample after the training has finished for each city and visualize the ground truth and predictions. The comparisons are shown in the below Figure 7.

As shown in Figure 7, the training sample closely reflects the trajectory for linear paths. It was noticed however, that for less linear trajectories, our model was not able to predict the general shape of the path accurately. This issue should be resolved with an implementation of a LSTM model that is able to learn these nonlinear paths.

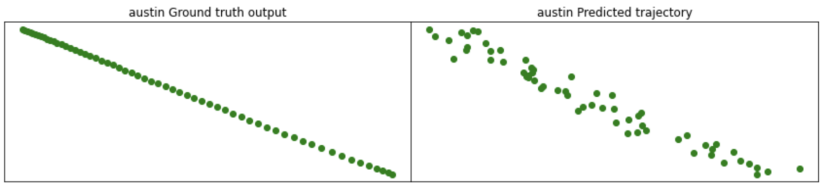
• The current ranking of our model on the leaderboard is 23. Our test RSME is 544.36019.

• In regards to what we plan to do in the future weeks, it mainly involves moving away from a linear model and optimizing a LSTM model for this prediction task. Batch normalization along with a self tuning learning rate are some of the main optimization tasks that we are looking into. By getting these two working, it will aid in reducing the training error by minimizing the spread of the data. We are also planning on moving away from using nn.Sequential() to run certain operations in a specified sequence to using a for loop to give us more freedom in what operations we can use within it and allow for easier debugging.

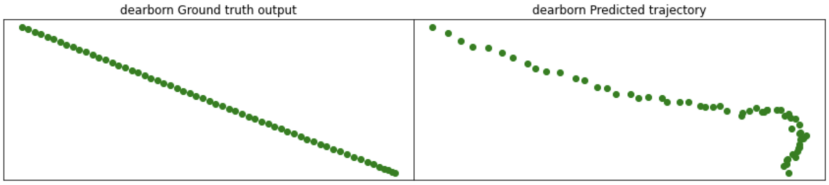
Additionally, from our comparison between ground truth and the current predictions, it can be concluded that the current model is underfitting the data due to having a higher training error than the test error. The training error is about twice as large as the testing error. In the following week, we plan to transition to a more complex model by switching to a LSTM model and adding more layers. By increasing the complexity of the model it should solve the underfitting problem.

4 GitHub Link

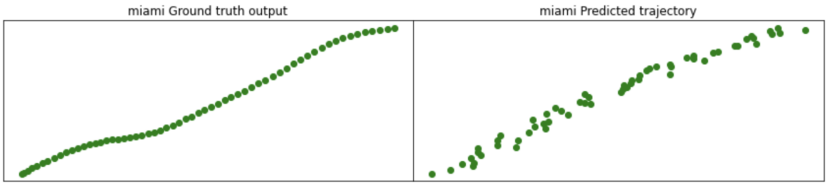
`https://github.com/zaxiang/Autonomous-vehicle-motion-forecasting`



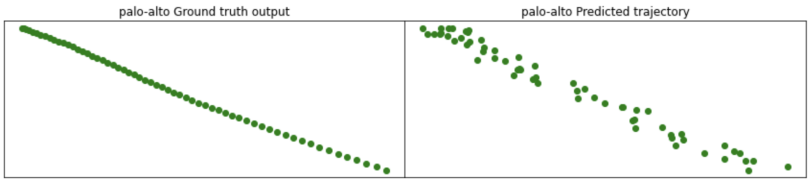
(a) Austin



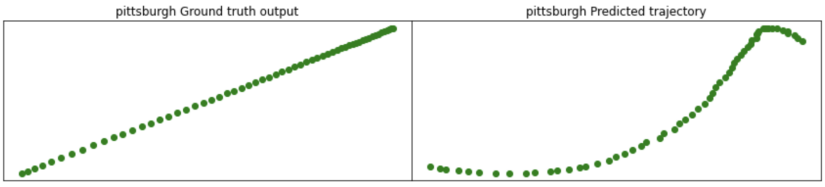
(b) Dearborn



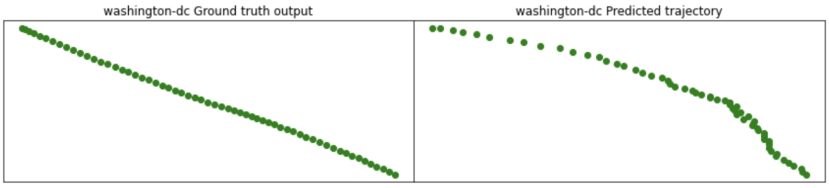
(c) Miami



(d) Palo Alto



(e) Pittsburgh



(f) Washington D.C.

Figure 7: Sampled Ground Truth and Predictions For Each City