# CSE 151B Project Milestone Report

**Yujie Zhang**
University of California, San Diego
yuz035@ucsd.edu

**Muchan Li**
University of California, San Diego
mul005@ucsd.edu

**Zhendong Wang**
University of California, San Diego
zhw005@ucsd.edu

**Sijie Liu**
University of California, San Diego
sjliu@ucsd.edu

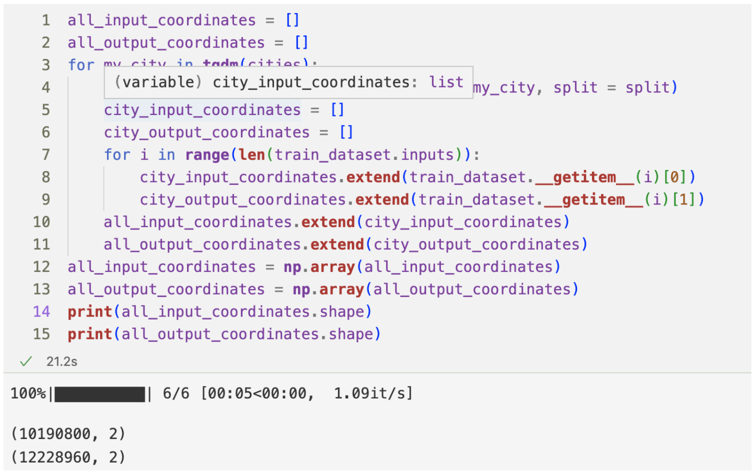
## 1 Task Description and Exploratory Analysis

In this section, we will introduce the prediction task in detail and conduct exploratory analysis of the datasets.

### 1.1 Autonomous Vehicle Motion Forecasting Task

In this competition, the task that my team's deep learning model is trying to achieve is to predict a vehicle's motion in a given city at a particular scene, as defined by it's XY position in the 2D Cartesian coordinate plane, 6 seconds into the future (broken down into 60 timestamps), given the object's current 5 seconds (or 50 timestamps) sequence of motion.
To quantify our problem's input and output space formally, our model will train on input matrices of dimension $N \times 50 \times 2$, where N denotes the number of geographical scenes where our 50-timestamp vehicle motion data are acquired from (for example, if we sample 300 scenes worth of data from San Diego, then our training dataset for San Diego would have dimension $300 \times 50 \times 2$), and attempt to predict output matrices that are of dimension $N \times 60 \times 2$.
This task is especially relevant under the current surge in autonomous driving breakthroughs, where people on one hand are beginning to realize the convenience and sustainability of this field of technologies, but on the other hand still remain doubtful about the its reliability and precision. While our deep learning model may just be a start, the larger suite of more advanced and rigorous deep learning models can bring important contributions to the field and aid the AV engineers in understanding more reliable algorithms and hence constructing more secure and more convenient autonomous vehicles.

## 1.2 Exploratory Analysis



```
all_input_coordinates = []
all_output_coordinates = []
for my_city in tqdm(cities):
    (variable) city_input_coordinates: list my_city, split = split)
    city_input_coordinates = []
    city_output_coordinates = []
    for i in range(len(train_dataset.inputs)):
        city_input_coordinates.extend(train_dataset.__getitem__(i)[0])
        city_output_coordinates.extend(train_dataset.__getitem__(i)[1])
    all_input_coordinates.extend(city_input_coordinates)
    all_output_coordinates.extend(city_output_coordinates)
all_input_coordinates = np.array(all_input_coordinates)
all_output_coordinates = np.array(all_output_coordinates)
print(all_input_coordinates.shape)
print(all_output_coordinates.shape)
```

✓ 21.2s

```
100%|██████████| 6/6 [00:05<00:00,  1.09it/s]

(10190800, 2)
(12228960, 2)
```

Figure 1. Dimensions

The input training dataset, combining all six different cities' inputs into one, is of dimension $10190800 \times 2$. The output training dataset, combining all six different cities' outputs into one, is of dimension $12228960 \times 2$.
The dimensionalities agree since we can confirm that $10190800/50 = 12228960/60 = 203816$, meaning our input and output training set contain equal number of 203816 scenes from all 6 cities.

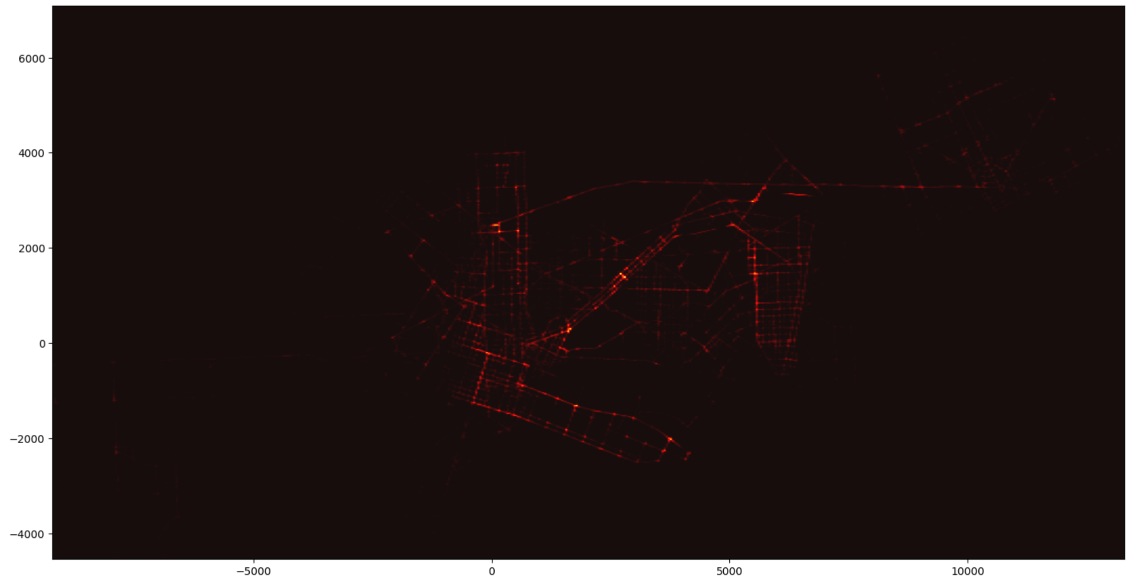


Figure 2. Distribution of input positions for all agents

The distribution of input positions for all agents looks like the graph above.

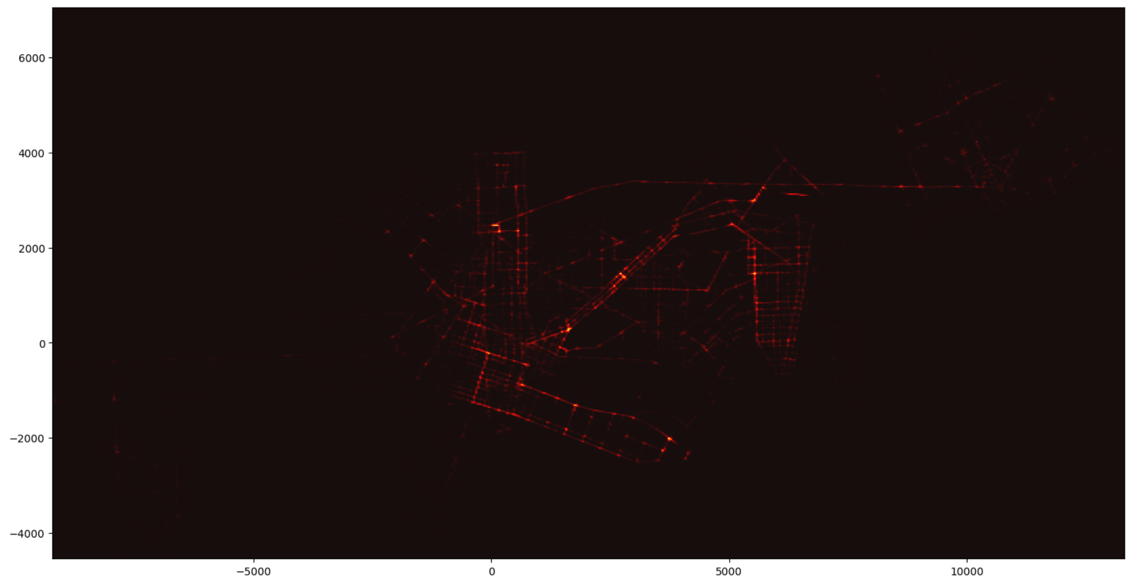


Figure 3. Distribution of output positions for all agents

The distribution of output positions for all agents looks like the graph above.

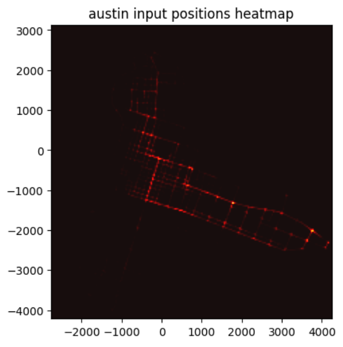


Figure 4. Austin Input

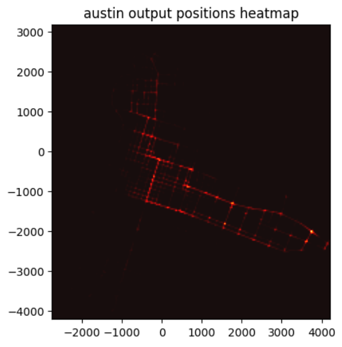


Figure 5. Austin Output

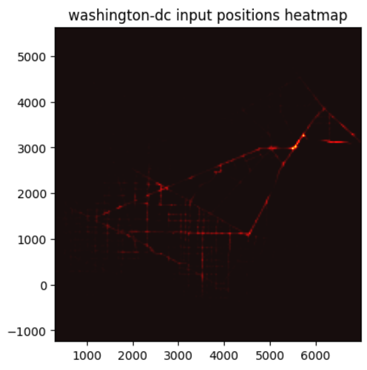


Figure 6. Washington D.C. Input

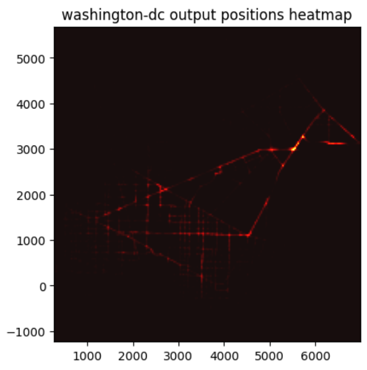


Figure 7. Washington D.C. Output

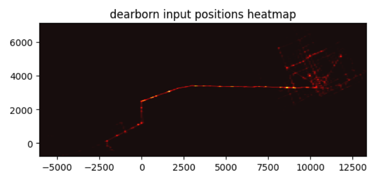


Figure 8. Dearborn Input

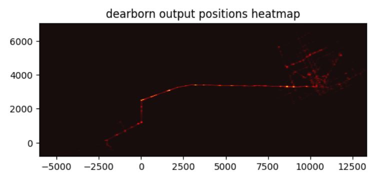


Figure 9. Dearborn Output

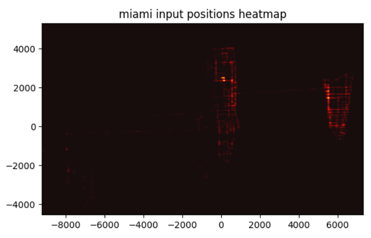


Figure 10. Miami Input

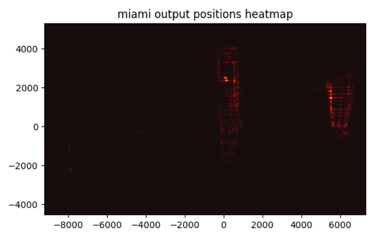


Figure 11. Miami Output

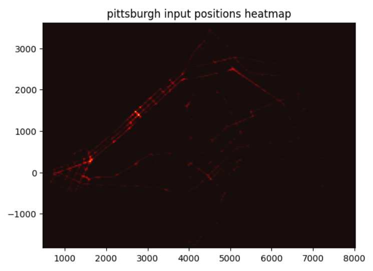


Figure 12. Pittsburgh Input

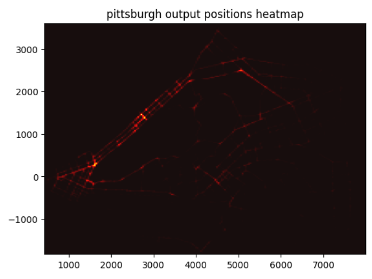


Figure 13. Pittsburgh Output

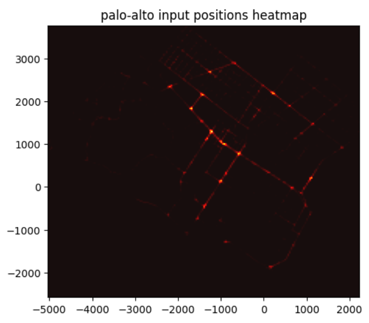


Figure 14. Palo Alto Input

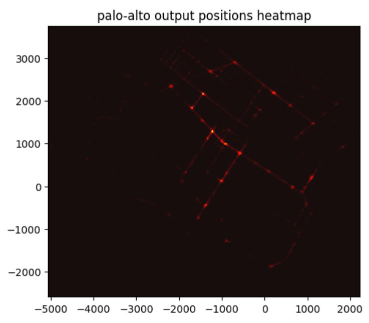


Figure 15. Palo Alto Output

### 1.3 More Visualization

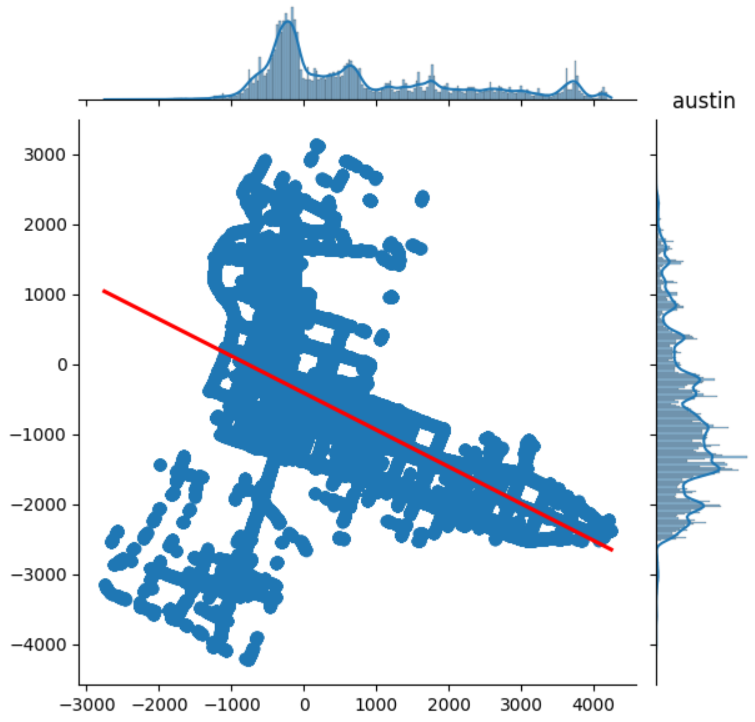


Figure 16. Joint plot of vehicle XY positions in Austin

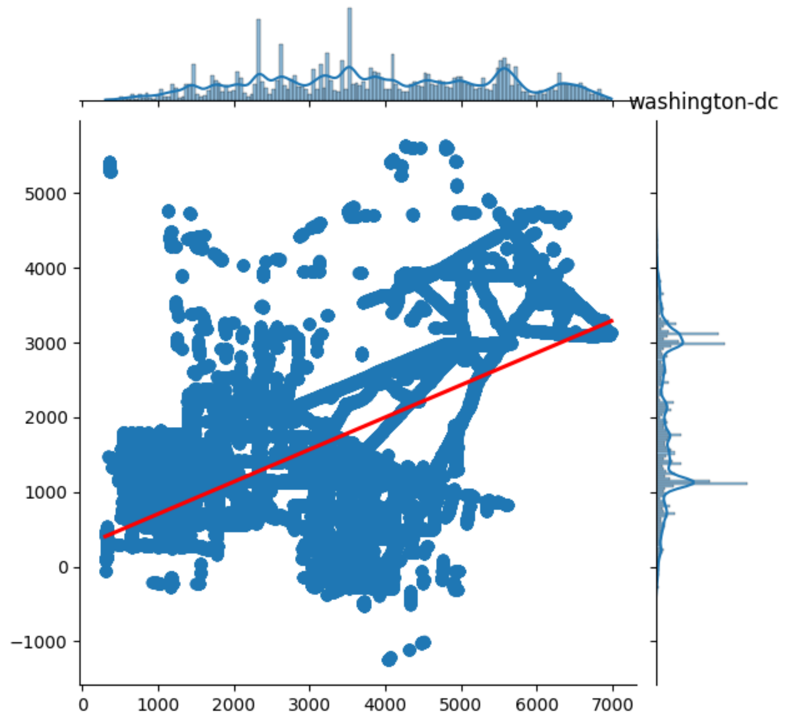


Figure 17. Joint plot of vehicle XY positions in Washington D.C

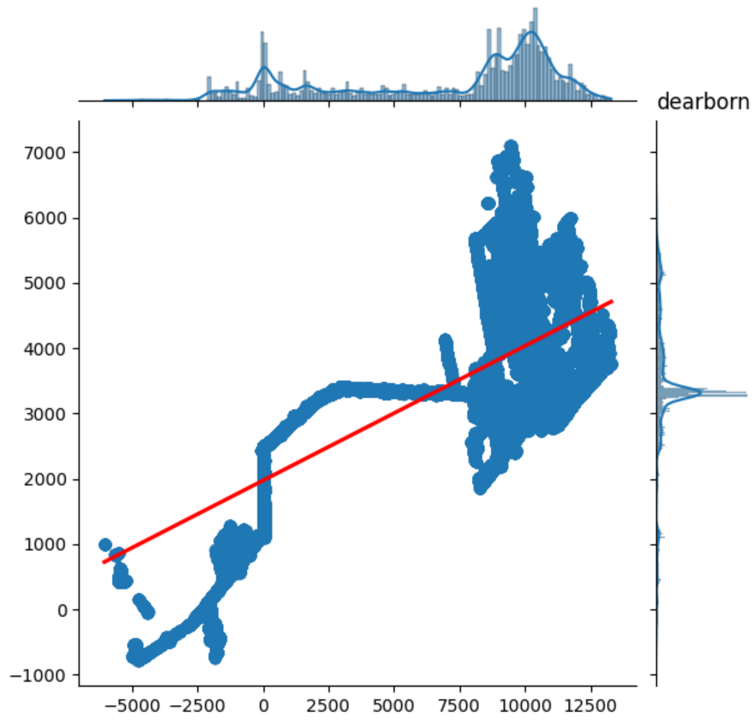


Figure 18. Joint plot of vehicle XY positions in Dearborn

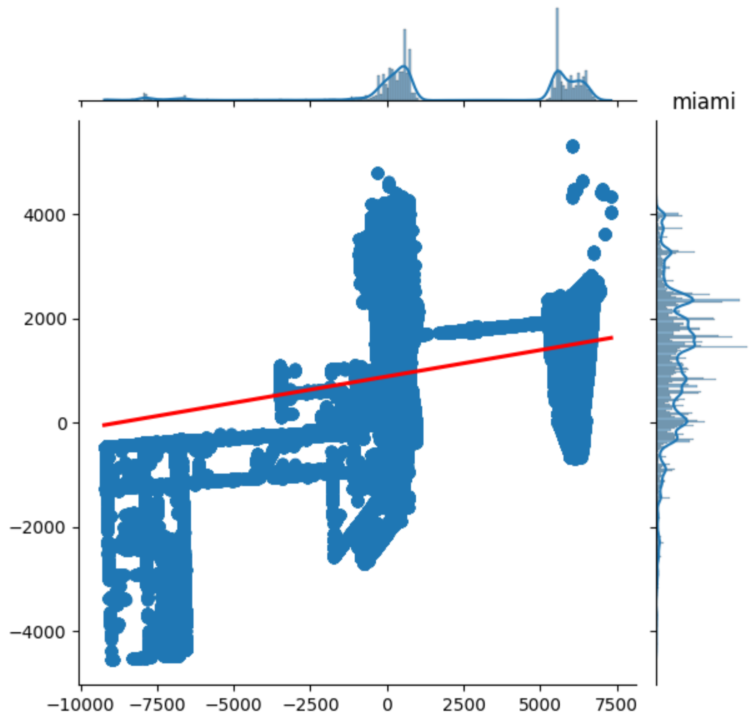


Figure 19. Joint plot of vehicle XY positions in Miami

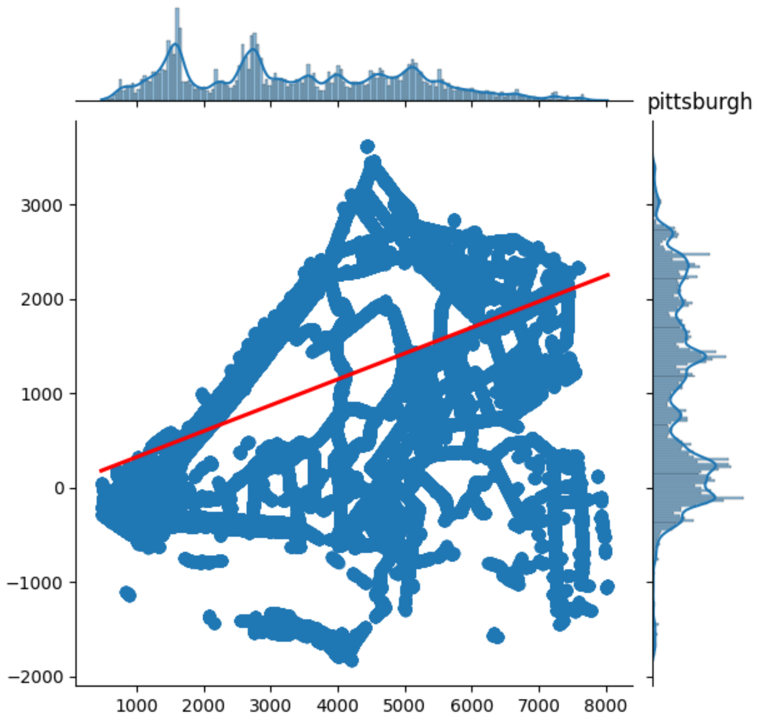


Figure 20. Joint plot of vehicle XY positions in Pittsburgh

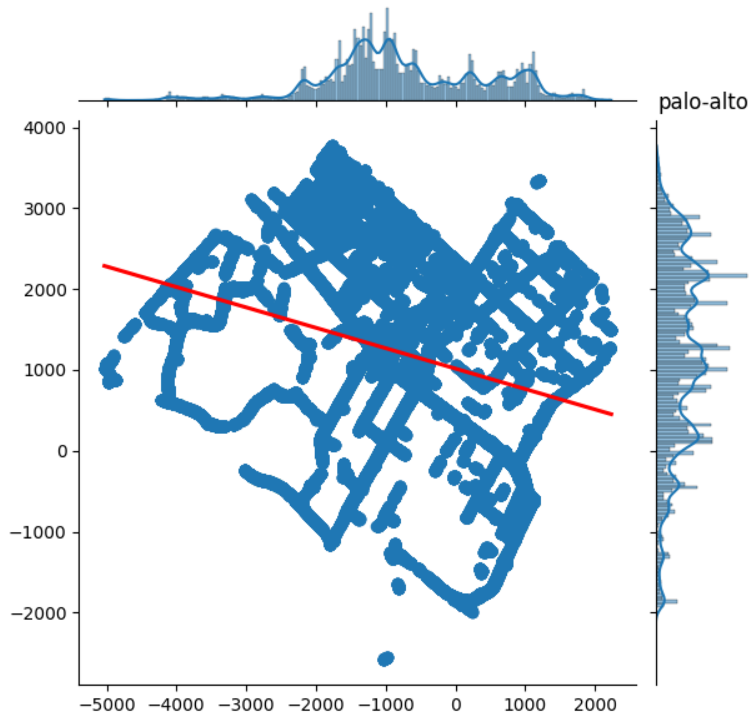


Figure 21. Joint plot of vehicle XY positions in Palo Alto

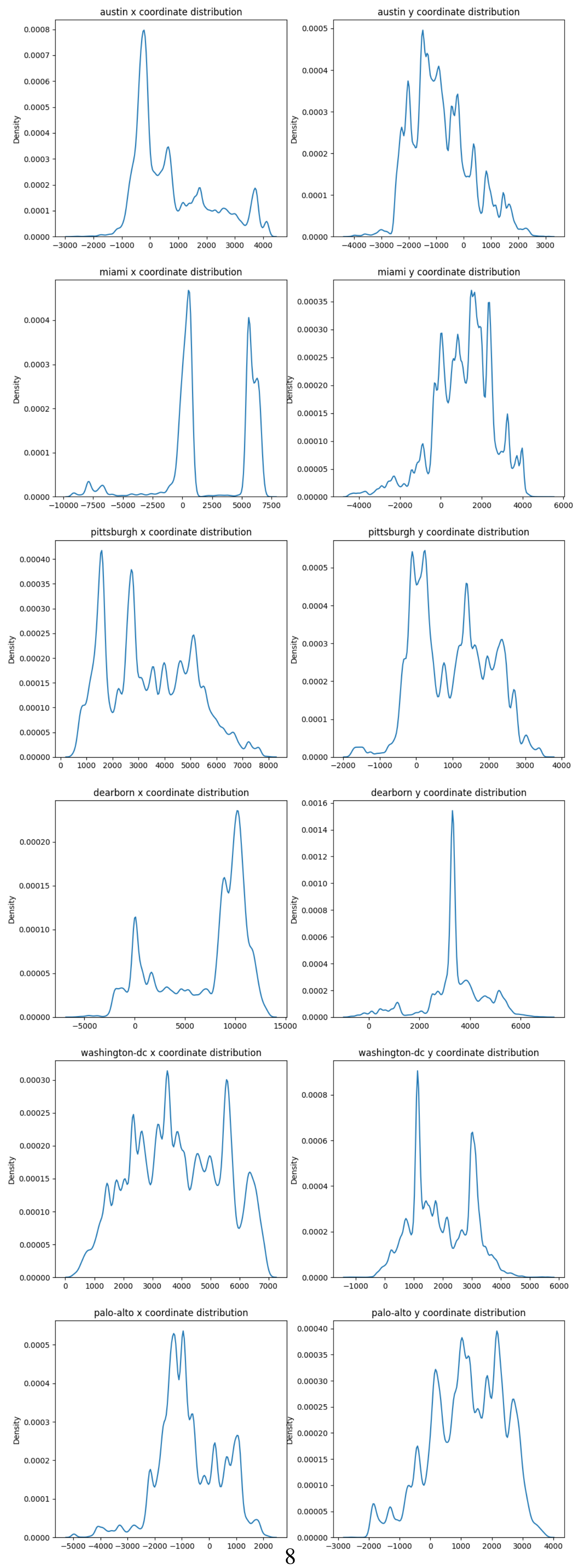

austin x coordinate distribution
austin y coordinate distribution
miami x coordinate distribution
miami y coordinate distribution
pittsburgh x coordinate distribution
pittsburgh y coordinate distribution
dearborn x coordinate distribution
dearborn y coordinate distribution
washington-dc x coordinate distribution
washington-dc y coordinate distribution
palo-alto x coordinate distribution
palo-alto y coordinate distribution
Density

Figure 22. x and y coordinate distributions for 6 cities

## 2 Deep Learning Model and Experiment Design

In this section, we will introduce how we set up the training and testing design for deep learning and the prediction models that we tried.

### 2.1 Training and Testing Design

For the training and testing, we used local machine CPU as the computational platform.

The optimizer we used for the encoder decoder model is Adam (adaptive moment estimation) optimizer. Different with the stochastic gradient descent, which gives the same learning rate for all weight updates and the learning rate does not change, Adam optimizer derives different learning rates for different parameters from estimates of first and second moments of the gradients. In this task, the learning rate that we used is 0.001. The exponential decay rate for the first moment estimates is 0.9. The exponential decay rate for the second-moment estimates is 0.999. And epsilon is 1e-8. Because of the self-tuning feature of Adam optimizer, we got good results without tuning the parameters.

We used the encoder decoder model to make multi-step predictions for each target agents. The input is the trajectory data of past 50 timestamps, then we put it in encoder with three linear layers. We used the Log-Sigmoid function as the activation function. Then use decoder to generate the output, which is the predicted next 60 timestamps' trajectory.

We trained six encoder decoder models using different city training data. For each city, we tuned the model parameters and made predictions for the corresponding city's testing data. After making predictions for all six cities, we combined the predictions based on the required format.

For each city, we trained 30 epochs. For Austin, the batch size is 20, training time for one epoch is 2.85s. For Miami, the batch size is 18,training time for one epoch is 3.52s. For Pitts, the batch size is 10,training time for one epoch is 2.80s. For Dearborn, the batch size is 35,training time for one epoch is 1.50s. For Washington DC, the batch size is 20,training time for one epoch is 1.47s. For Palo-alto, the batch size is 20,training time for one epoch is 0.68s.

### 2.2 Prediction Models

In this section, we will introduce the prediction models we used for this project.

1. Linear Regression Model

Linear regression model regards the input data points as x axis, and calculate a y value based on the k, b value we trained. The math formula is y = kx + b.
This simple regression model did not perform well at all since the relationship between X positions and Y positions cannot be explained by a linear line, as already shown by each city's joint plots we presented in section 1.3. The next step for us is to come up with a model that would take the geographical representation of XY positions, as revealed in each city's scatter plots above, into account and recurrently pass it on so the XY position from one point in time is indicative of the possible XY position in the moment following.

2. Encoder-decoder Model

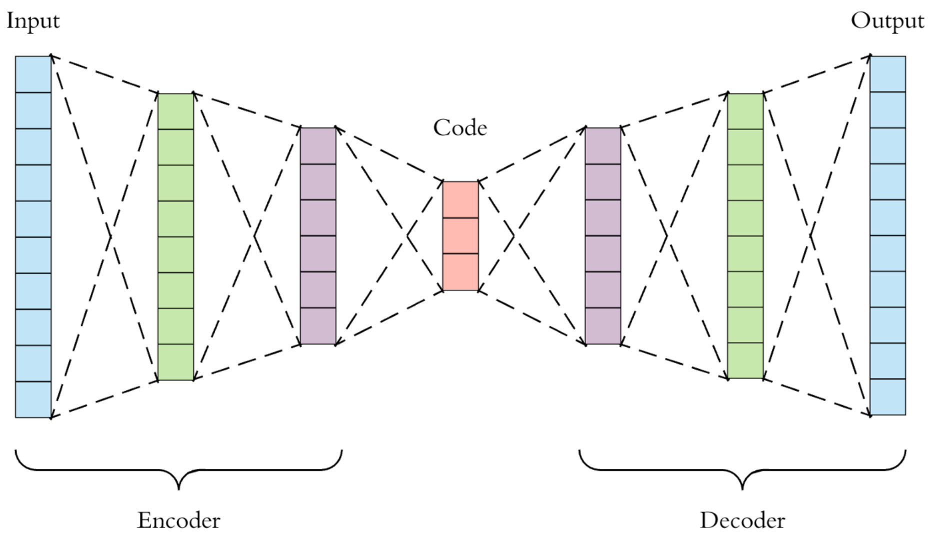


Figure 23. Encoder-Decoder [1]

We believe a RNN model would fit our subjective better since it recurrently takes in information and pass them down to the following stages of model training. The information that happen earlier in time would influence what our model gain from training with the information passed in later down the stream–this is exactly what happens with driving!
More specifically, we choose to rely on a encoder-decoder model for the task. This model consists of three parts: encoder, latent space, decoder. The encoder processes the input and encode all the information to a fixed-length vector. And this vector is sent to decoder and predict our targets based on the read-in vector and output predictions based on the shape of target/label.

## 3 Experiment Results and Future Work

In this section, we will show the experiment results by visualizing training loss (Root Mean Squared Error) value over training steps, analyze the results, and discuss plans to improve the performance in the future.

To visualize the experiment results, I will use the model trained on "austin" dataset as an example. The training loss (RMSE) over 30 epochs is visualized below.

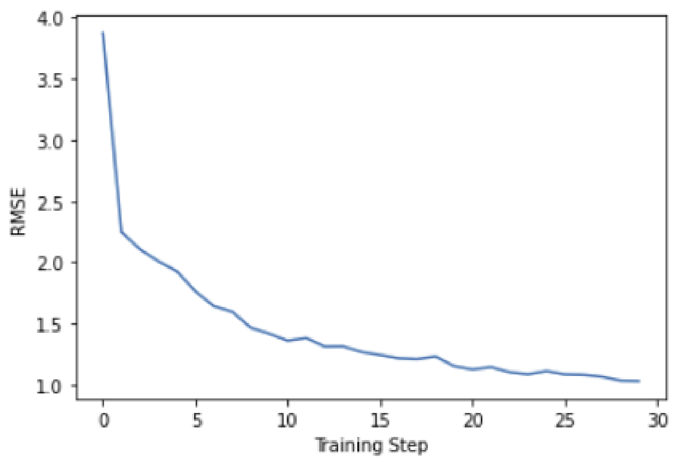


Figure 24. Training Loss

We randomly sampled 4 training samples and compared their ground truths and predictions.

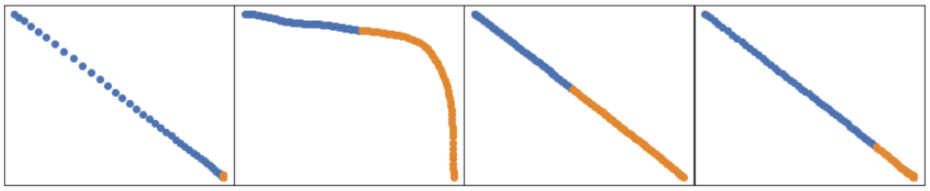


Figure 25. Ground Truths

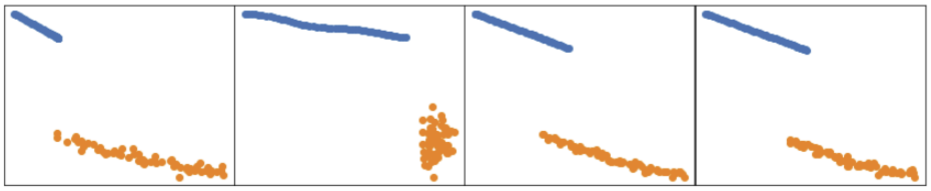


Figure 26. Predictions

Our current ranking on the leaderboard is 25, and the test RSME for "austin" model is 1.315.

In summary, the training loss of all six models followed the shape of the exponential decay. The models can roughly predict the direction of the car if the the car is moving in a line, but the model has tough time predicting the trajectory if the car makes some turns. In addition, as we can see from the training samples, there is a huge deviation between the start of the prediction and the end of the input.

To solve the issues mentioned above, we plan to try different models like LSTM which can learn the information from the past trajectory and hopefully solve the issue of model only generating good predictions for car moving in a line. We will also increase the complexity of simple models like MLP and hopefully capture more useful information from the inputs.

## References

[1] Dertat, Arden. (2017) *Applied Deep Learning - Part 3: Autoencoders*. Towards Data Science.