CSE 151B Project Milestone Report

William Sun, wis003@ucsd.edu, https://github.com/wis003/CSE151B

1 Task Description and Exploratory Analysis

1.1 Problem A

The task is to predict and forecast the motion of agents tracked by an autonomous vehicle. It is important to develop models to accurately predict the future positions of these agents for autonomous vehicles to work properly and safely.

Where the data is sampled at 10 timestamps per second, the input to the prediction task is an input array of 50 timestamps, with each time stamp having an (x, y) coordinate. Thus, the input array has shape $(50, 2)$. After processing through the model, output needs to predict the position of the agent for the next 60 timestamps, so the output array has shape $(60, 2)$.

Thus, the prediction task will consist of taking in 50 positional coordinates which are ordered in time (5 seconds), and output the next 60 predicted positional coordinates in time (6 seconds).

1.2 Problem B

The training data set has input shape $(203816, 50, 2)$ and output shape $(203816, 60, 2)$. The test data set has input shape $(29843, 50, 2)$ and output shape $(29843, 60, 2)$. For each trajectory, the model input dimension is $(50, 2)$ and its output dimension is $(60, 2)$.

The distribution of input positions for all agents is shown as a heatmap in Figure 1. The distribution of output positions for all agents is shown as a heatmap in Figure 2. The distributions of positions for each city is shown in the following heatmaps (Figure 3-8).

Preprint. Under review.

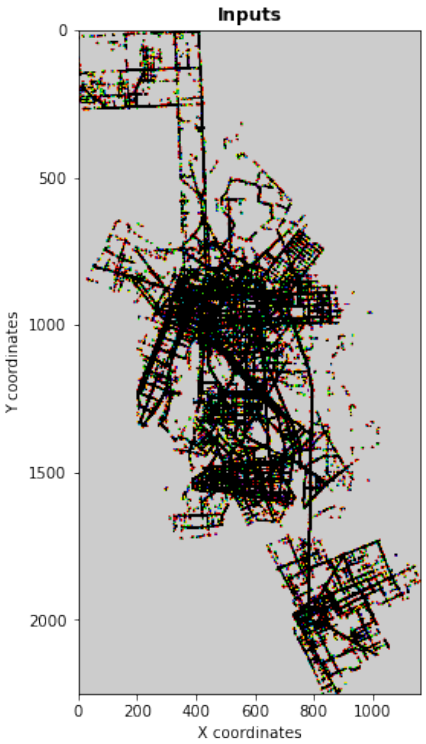

Figure 1: Input data

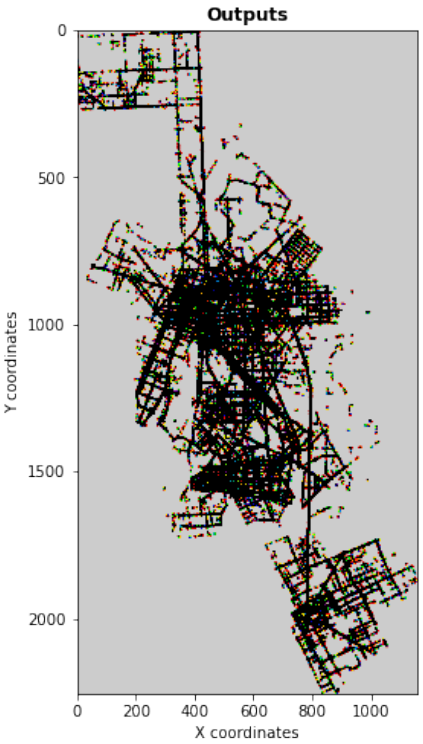

Figure 2: Output data

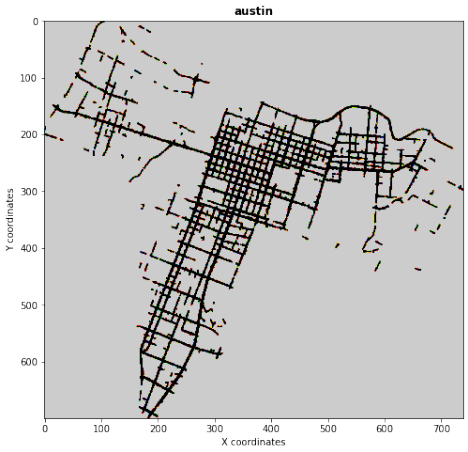

Figure 3: Austin data

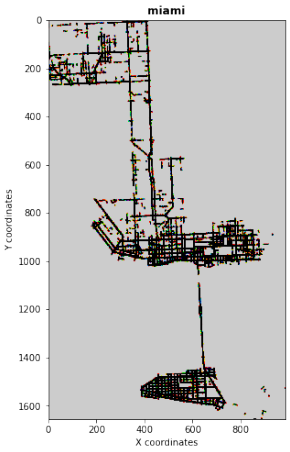

Figure 4: Miami data

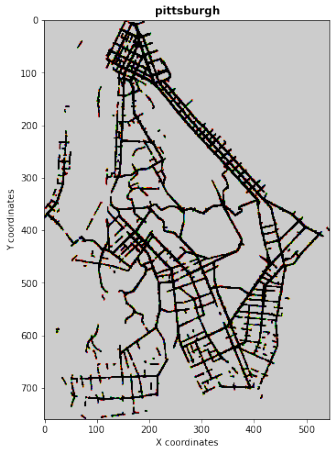

Figure 5: Pittsburgh data

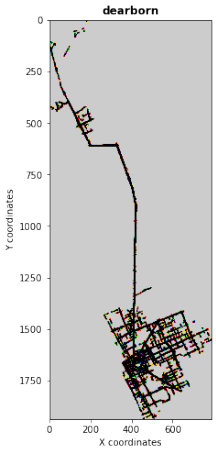

Figure 6: Dearborn data

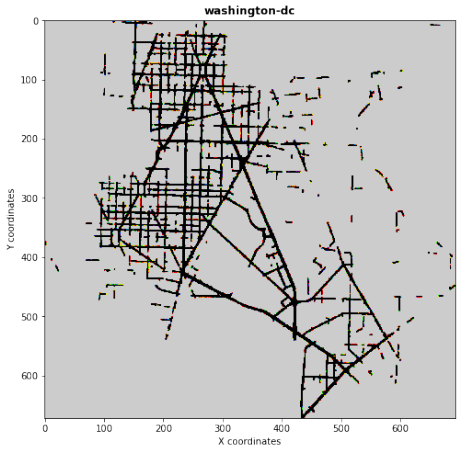

Figure 7: D.C. data

Figure 8: Palo-Alto data

The positions shown in these heatmaps illustrate that the motion/direction of agents are generally very linear and straight. Thus, in the bonus exploratory analysis, I will dive into the information provided by the motion of the agents.

1.3 Bonus exploratory analysis

In figures 9-10, we visualize the frequency of average velocity in each input set.

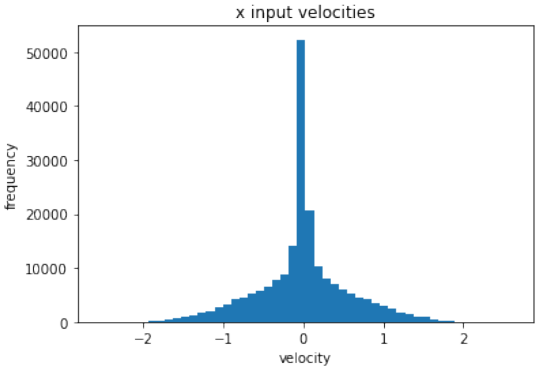

Figure 9: x input velocities

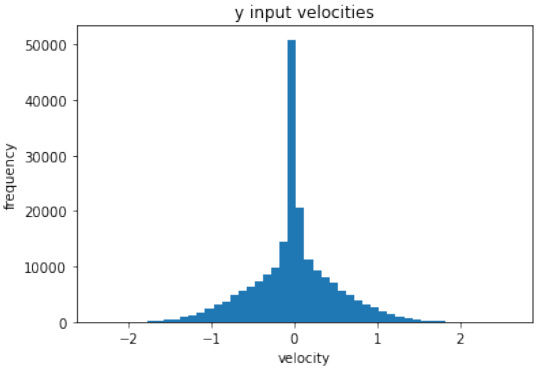

Figure 10: y input velocities

In figures 11-12, we visualize the frequency of average velocity in each output set.

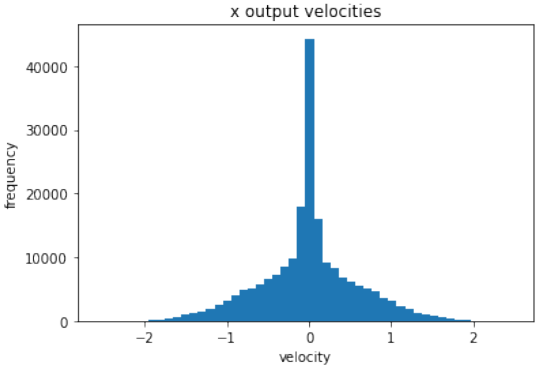

Figure 11: x out velocities

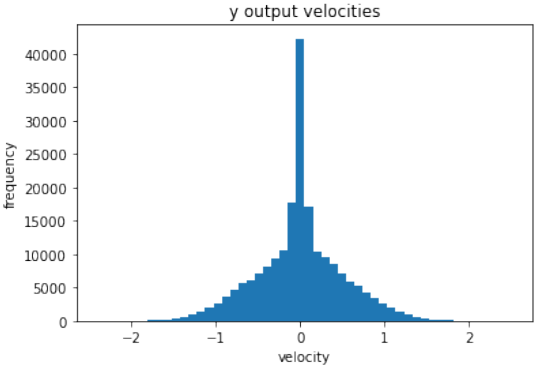

Figure 12: y out velocities

In figures 13-14, we visualize the frequency of average acceleration in each input set.

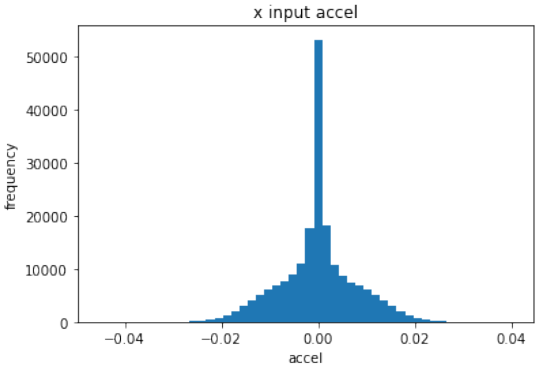

Figure 13: x input accel

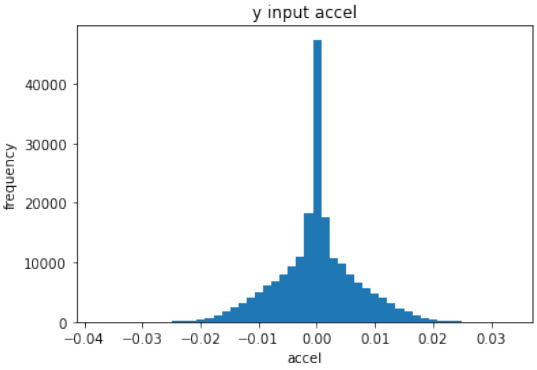

Figure 14: y input accel

In figures 15-16, we visualize the frequency of average acceleration in each output set.

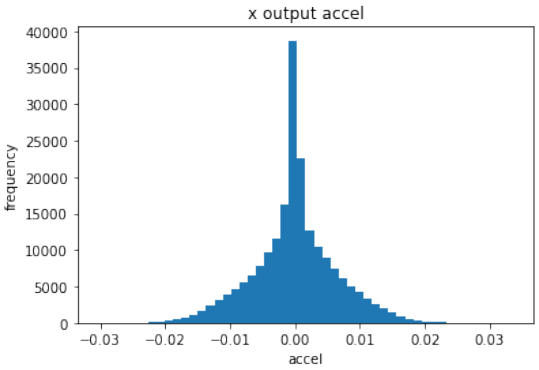

Figure 15: x output accel

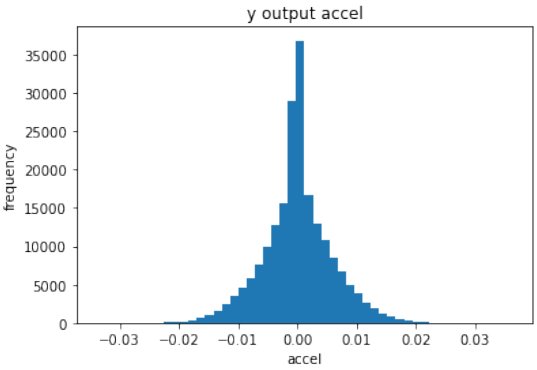

Figure 16: y output accel

In figures 17-20, we take the the previous data and visualize them in 2D.

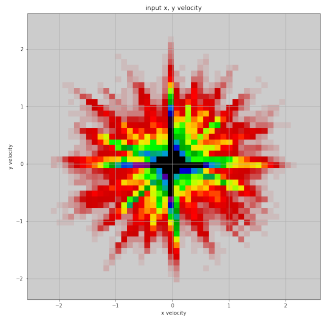

Figure 17: 2D in velocity

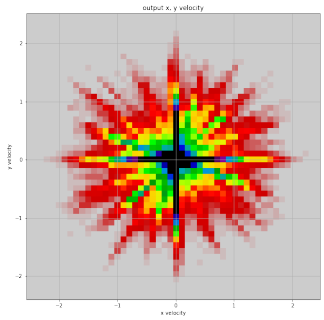

Figure 18: 2D out velocity

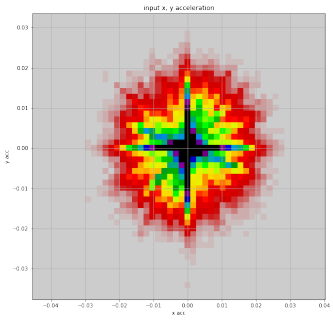

Figure 19: 2D in accel

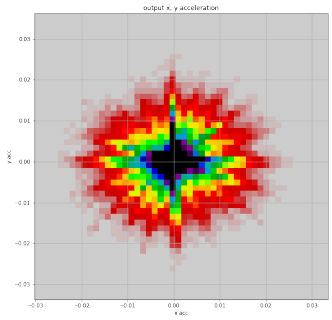

Figure 20: 2D out accel

In figures 21-24, we show the frequency of differences between input and output data.

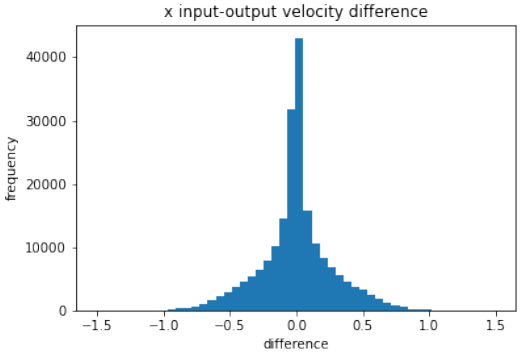

Figure 21: x velocity diff

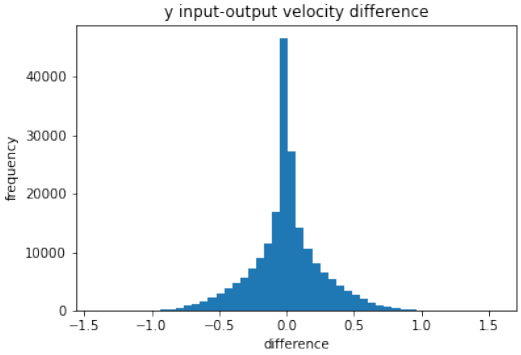

Figure 22: y velocity diff

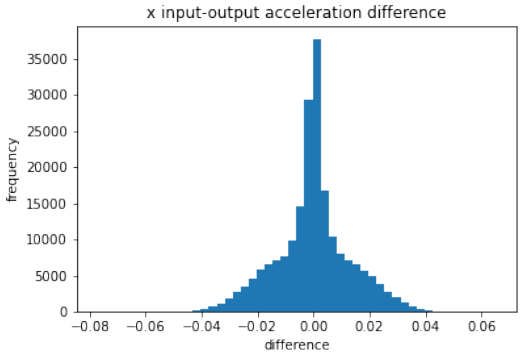

Figure 23: x accel diff

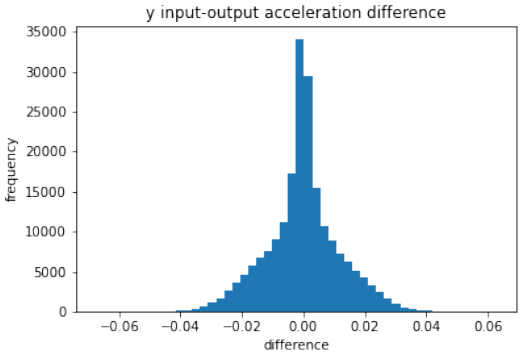

Figure 24: y accel diff

The further exploratory data analysis reveals that a linear prediction model will be very accurate for the data set, because we can see the density of velocity/acceleration is greatly concentrated near zero, and also the difference between the input and output data is typically zero. Thus, in general a straight linear prediction based on input data should accurately predict the output positions.

2 Deep Learning Model and Experiment Design

2.1 Problem A

The computational platform used for both training and testing was DSMLP, where the environment was specifically "ucsdets/scipy-ml-notebook:2022.2-stable, Python 3, nbgrader (1 GPU, 8 CPU, 16G RAM)"

The optimizer used was PyTorch's Adam. Learning rate was tested multiple times, where 0.01 and 0.001 both worked equally well. Learning rate decay and momentum were kept using PyTorch's default parameters and were not touched.

In our best deep learning implementation, multistep prediction for each target agent was made through an encoder-decoder architecture, where the decoder outputs a 120 element array, reshaped into the expected prediction output shape of $(60, 2)$.

The city information was not specifically used to modify prediction. In general, data for all cities were used in each epoch to train the model.

The MLP model was trained in 40 epochs. The batch size was 4. It took on average 2.5 minutes to train the model through one epoch, in which one epoch went through the entire training data set.

The above design choices where made due to the inspiration of the discussion code. I started developing and testing the model beginning with the sample discussion code, and began to tweak hyperparameters and tried different architectures to see what generally works the best. In the end, computational limitation did play a factor in the design choice. For example, I tried the LSTM and transformer implementations from PyTorch, but they both take too long to train (>10 min per epoch), while the training loss was unreasonably high (in the millions). Thus, simpler models for prediction that were more lightweight were chosen.

2.2 Problem B

The following models are ones that I have tried to make predictions.

1) Initial Architecture (Discussion 7 Architecture): Encoder - 3 hidden layers, Decoder - 3 hidden layers, Batch-size - 4, lr - 1e-3, 10 epochs per city, 80/20 Training/Validation. This model was based off the encoder-decoder architecture provided in discussion. Submission score: 5564.16475

2) Deeper Architecture: Encoder - 4 hidden layers, Decoder - 4 hidden layers, Batch-size - 4, lr - 1e-3, 20 epochs per city, 80/20 Training/Validation Split. This model is similar to the first one, except it is deeper with more hidden layers. Submission score: 1733.08812

3) Transformer Model: Encoder - 4 hidden layers, Decoder - 4 hidden layers, Batch-size - 4, lr - 0.1, dropout - 0.2, dmodel = 2, nhead = 2, 5 epochs per city. This model utilizes the transformer model provided by PyTorch. Submission score: 13579726.63092

4) Average Distance Prediction. This model takes the average distance between two time intervals from the inputs and uses this value to calculate the next 60 positions. Acceleration is also taken into account which increases the velocity over time. Submission score: 71.73503

5) Linear Regression: 10 epochs. This model runs a linear regression model on 50 input data points, and generates 60 separate outputs, which will be the next 60 predicted positions. Submission score: 9548.50214

3 Experiment Results and Future Work

3.1 Problem A

The training loss (RMSE) value over training steps for my best performing MLP model is shown in Figure 25.

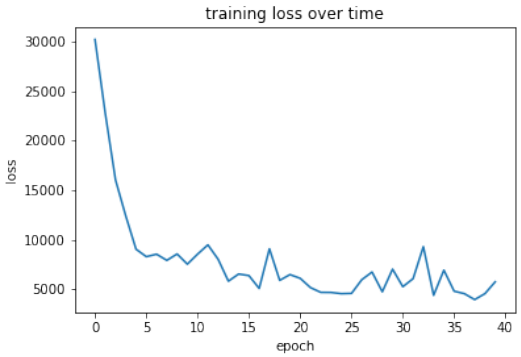

Figure 25: Loss over time

Figures 26-29 visualize the ground truth and the MLP model's predictions on a 2D plane.

Table 1: Summary of experiment results

Design	Description	Score
(1)	Encoder-Decoder	5564.16475
(2)	Deeper Architecture	1733.08812
(3)	Transformer	13579726.63092
(4)	Linear Regression	9548.50214
(5)	Average Velocity w/ Acceleration	71.73503

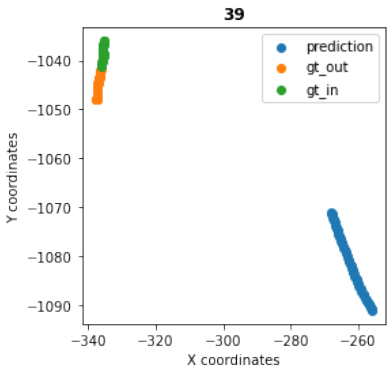

Figure 26: Sample 39

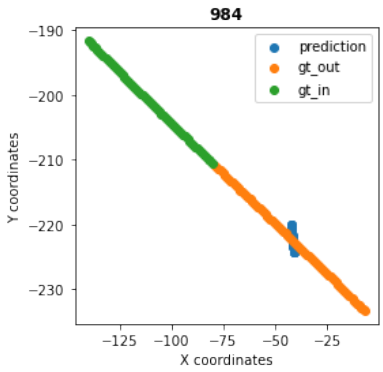

Figure 27: Sample 984

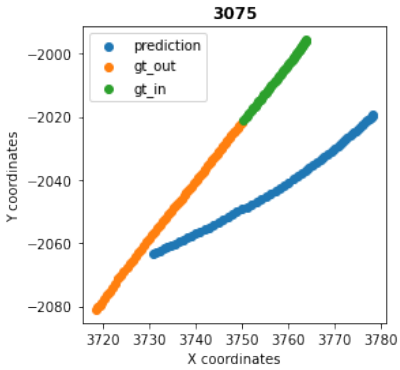

Figure 28: Sample 3075

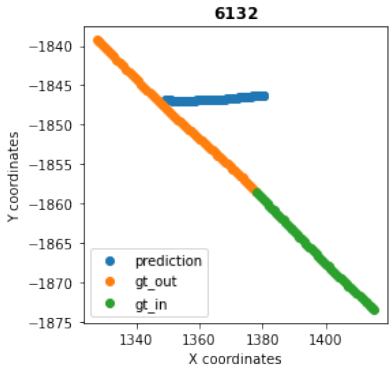

Figure 29: Sample 6132

My current ranking on the leaderboard is 10, with a score of 71.73503.

In summary, current experiment results show that a non-learning based approach works best for the data set. Both the results and the data analysis point to the fact that a linear prediction generally matches the data set. Random samples of the data also show that most datapoints are linear. A comparison of different experiment designs are shown in Table 1. Lessons/issues I have learned so far is that the simpler model is definitely the correct approach to start with to understand the data. It is both more efficient to train, and easier to implement/deploy.

To improve the model's performance, I plan to test the following things. First, is to test if a weighted average for velocity towards the tail end of the input is a better predictor. Second is to implement preprocessing/postprocessing methods such as a Kalman Filter on the data.

References

[1] Argoverse 2: Next Generation Datasets for Self-Driving Perception and Forecasting, Wilson et al. *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[2] What-If Motion Prediction for Autonomous Driving, Khandelwal et al. *arXiv preprint arXiv:2008.10587*, 2020.