CSE 151B Project Milestone Report

Benjamin Becze, Alex Makhratchev
Department of Computer Science
UC San Diego
La Jolla, CA
https://github.com/Benjamin242/av_trajectory_cse151b
bbecze@ucsd.edu, amakhrat@ucsd.edu

May 22, 2022

Abstract

With the rise of electric vehicles, people have also started thinking more about the implementation of autonomous driving for vehicles. Aside from computer vision, a big part of autonomous driving is trajectory prediction, trying to predict where a vehicle will be based on its current trajectory. Here we explore deep learning methods to solve this issue.

1 Task Description and Exploratory Analysis

1.1 Problem A

Autonomous driving vehicles have been around for decades but there is still lots of research to be done in the field to perfect it. One large subtask that has gotten a lot of focus recently is motion prediction. It is mandatory for a car to predict where the objects around it are going in order to plan a safe route and avoid obstacles. In this project, we are given the first five seconds of motion of a vehicle, our task is to predict the following six seconds of it. Mathematically, our input and output data is given in a coordinate system form, with the location of the object being reported with a frequency of 10Hz (10 times a second). Thus our input has 50 (x,y) coordinate pairs, and our output should have (60 x,y) pairs. Our model will intake a predetermined step size model and attempt to predict the next single (x,y) coordinate pair. This will be a supervised learning task because we are fortunate enough to receive the labeled data.

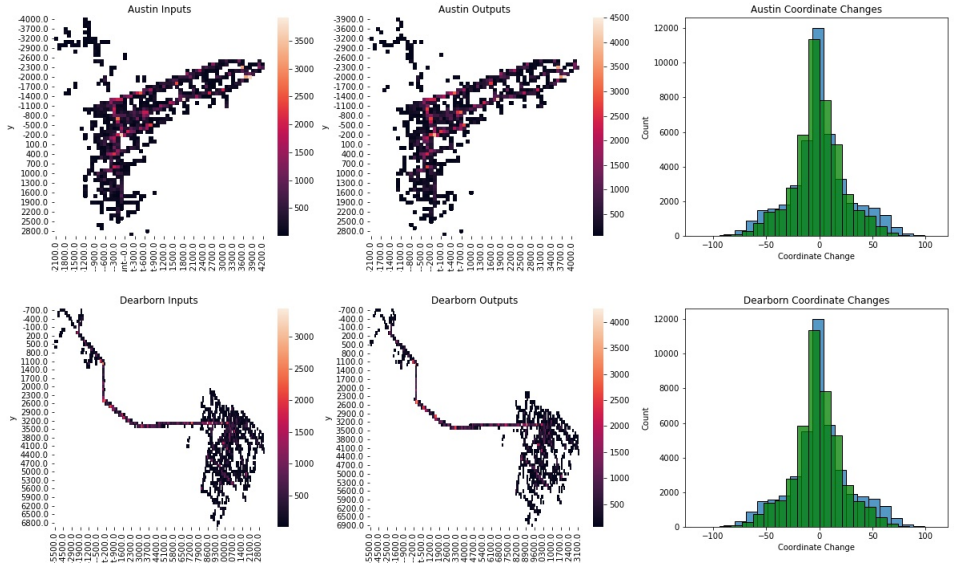
1.2 Problem B

The data is shaped as sequences of X and Y coordinate pairs of length 50, indicating a vehicle's position over the course of 5 seconds as the input sequence. The output sequence is in the same format, only with a length of 60, indicating

the vehicles position over the next 6 seconds. Each city contains a different amount of data to train on, and they have the following sizes [data points, features, sequence length]:

- Austin: Train[43041, 2, 50], Test[6325, 2, 60]
- Miami: Train[55029, 2, 50], Test[7971, 2, 60]
- Pittsburgh: Train[43544, 2, 50], Test[6361, 2, 60]
- Dearborn: Train[24465, 2, 50], Test[3671, 2, 60]
- Washington DC: Train[25744, 2, 50], Test[3829, 2, 60]
- Palo Alto: Train[11993, 2, 50], Test[1686, 2, 60]

We can assume that changes by city are vastly different, as there are certainly cities that are more dense with sharper and more frequent turns and bends, while others are more straightforward with more sparse roads. To see this difference we plotted the distribution of coordinate changes on the X and Y axis for each city's data. We also plotted heatmaps of random samples of 5000 tests from each city to show the distributions of trajectory.



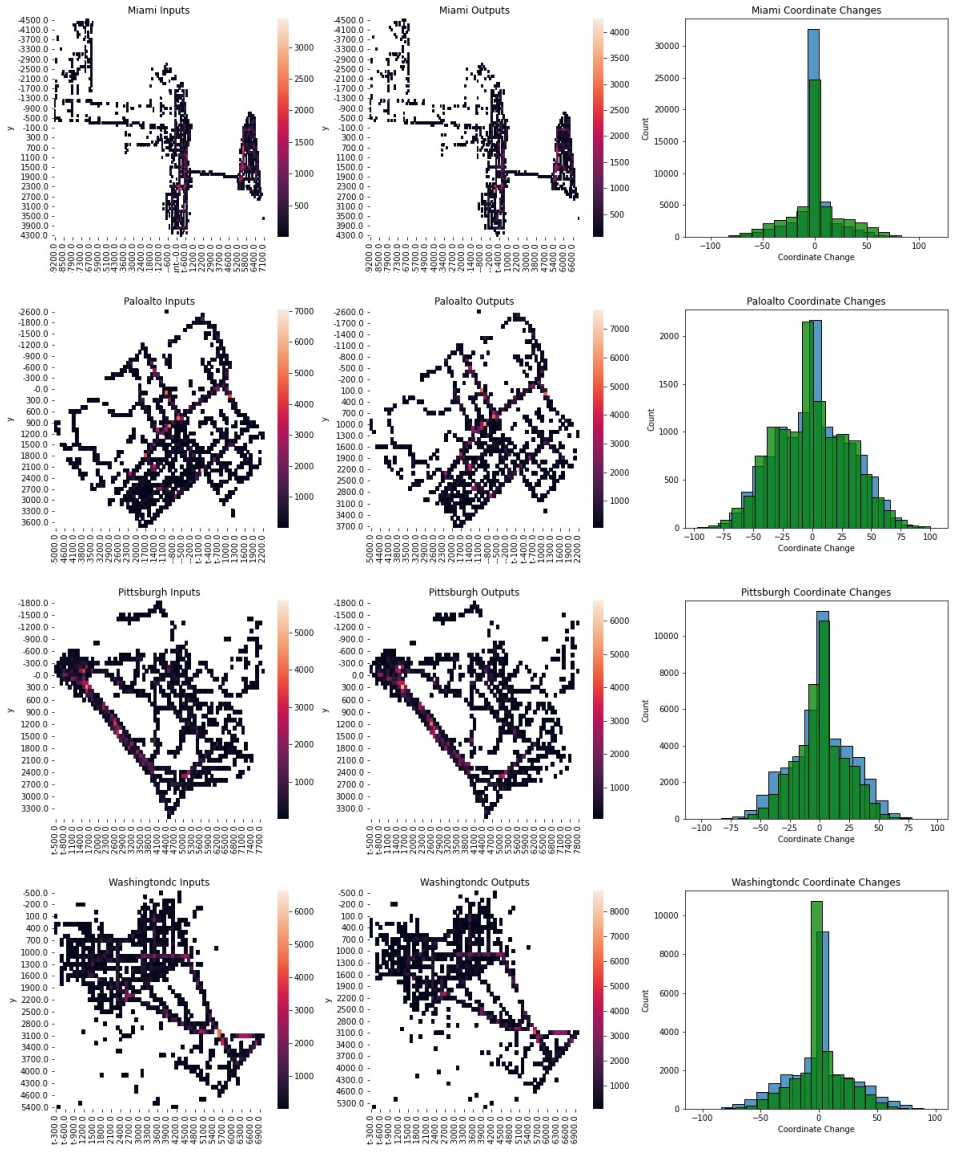


Figure 1: Trajectory and trajectory change distributions by city.

It can be seen that cities like Palo Alto and Pittsburgh contain a higher amount of higher degree turns. To account for this, we may try to scale the degrees for each city to normalize between them.

2 Deep Learning Model and Experiment Design

2.1 Problem A

For our primary deep learning model, we set up the problem to run on a GeForce RTX 3060ti with 4864 cuda cores in order to maximize the speed of training. We initialized the model with the following parameters:

- Optimizer: Adam
- Learning Rate: 0.5
- Epochs: 1000
- Hidden Size: 16
- Loss: Mean Squared Error
- Batch Size: 258

At the moment, we have trained one model using the palo-alto dataset, but in the future we plan to try to combine the city data but normalize each city by its own range of degrees, as some cities contain degree changes that are more dramatic than others. We chose the Adam optimizer because it has worked well in previous projects and it seems to be a common optimizer used in neural networks. After trying several learning rates, we are staying at 0.5 for now because changing it did not seem to have any affect on convergence. We will definitely tune the learning rate later in the project, but currently we must find what is making the MSE of the model so large. The hidden size we used is 16, which was a difficult one to pick as there was not much information to go on, but we found that using a multiple of the input size of 2 was beneficial. Our batch size was 258 because the training data was not too large as to not fit into memory (for palo alto), but we still wanted training speed to be quite quick. With all of these settings, the time to train one epoch is about 2 seconds.

2.2 Problem B

To begin with, we made a linear regression model to provide a baseline that we would try to improve upon with neural networks. To do this we first had to prepare the input sequences of 5 seconds by transforming them to 6 seconds using a fourier resampling method with scipy.signal.resample. We decided it would be a sufficient solution to having mismatched input and ouptut sequences because the data is time series data that is continuous by nature, so resampling the input arrays with interpolated values would not be too far from the original. We then flattened each sequence and added timestep as a feature alongside the X and Y coordinates, and created a linear regression model for each city. The result was an MSE of 918.10013, placing number 19 on the Kaggle leaderboard as of 5/21/2022 6:22pm PST.

After creating the linear regression model we decided that a sequence to sequence LSTM or GRU based model could work well for predicting future trajectory sequences. The model would consist of an encoder with an LSTM layer, and a decoder with an LSTM layer and a fully connected layer.

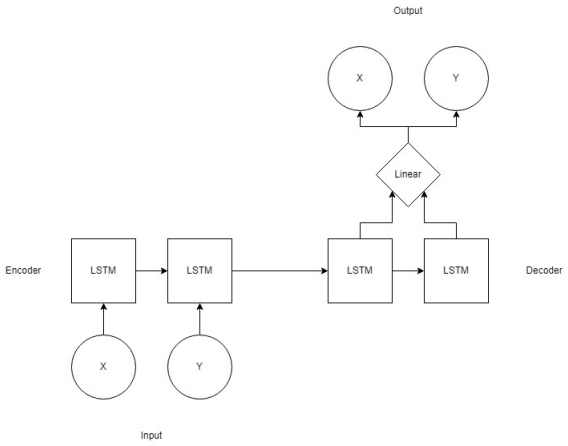


Figure 2: Seq2seq structure.

Right now the linear regression model we have beats the seq2seq model by a longshot, but we plan to improve upon this model greatly in the future by adding attention mechanisms, dropout, learning rate optimization and more.

3 Experiment Results and Future Work

3.1 Problem A

With our current iteration of the seq2seq model, the training loss is extremely high on the palo alto dataset with the following MSE trend:

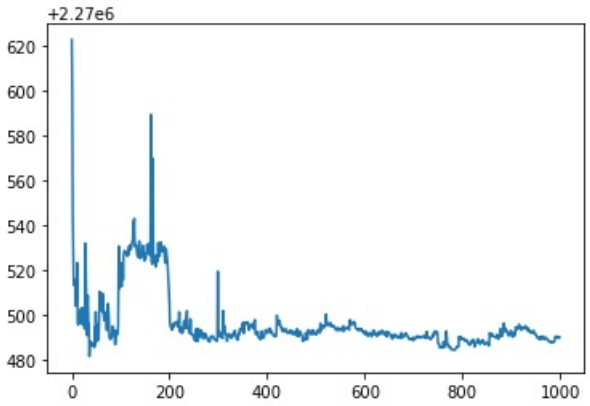


Figure 3: MSE loss of Seq2seq model.

Comparing our current best model (linear regression) with the ground truth predictions for a few of the palo alto trajectories we get the following:

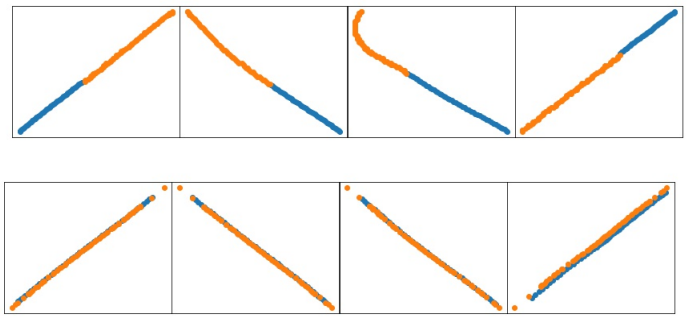


Figure 4: Ground truth vs linear regression trajectory. Orange=output, Blue=Input

We can see that the linear regression model can somewhat predict the linear trajectories, but when there is a curve like in the third example, the model fails. Our MSE for the linear regression model was 918.10013, placing number 19 on the Kaggle leaderboard as of 5/21/2022 6:22pm PST and the MSE for the seq2seq model was about 2 million, and would place 31st on the Kaggle leaderboard.