CSE 151B Project Final Report

Eugene Kim
Department of Data Science
University of California, San Diego
La Jolla, CA 92093
eykim@ucsd.edu

1 Task Description and Background

1.1 Problem A

This project uses the Argoverse 2 motion forecasting dataset that contains over 250,000 different scenarios from 6 different cities each recorded for 11 seconds at a frequency of 10hz. The data is given in pairs of (x, y) cartesian coordinates. The task requires a model to predict the location of a tracked object k steps into the future. Specially, the model must learn from the initial 5 seconds of the car's location and predict the next 6 seconds. Successfully solving this task can have a great impact in our daily lives by improving the safety and reliability of autonomous driving. It is important to have a prediction model since there are potential instances where autonomous cars fail to recognize their surroundings using purely computer vision. It can act as a tool to insure safety as well as accuracy.

1.2 Problem B

There are two interesting methods used in the past for motion forecasting.

1) KalmanNet: Neural Network Aided Kalman Filtering for Partially Known Dynamics [1]

In this paper, the authors proposed the use of implementing an Extended Kalman Filter with a GRU. The intention of this method was to create a interpretive model-based neural network that can use prior model knowledge while having data information. In this specific case, they used the GRU to adjust the Kalman gain and kept everything else the same. This relates to the current project as the current project also use the combination of data and Kalman Filters to produce accurate results. One of the differences is that this project fails to incorporate a deep learning model with the filter, rather it incorporates the raw data.

2) Extended Kalman Filter Design and Motion Prediction of Ships Using Live Automatic Identification System (AIS) Data [2]

In this paper, the authors proposed the use of implementing an Extended Kalman Filter with a Live Automatic Identification System (AIS) Data. This is quite similar to the given input data of this current project. The high frequency AIS data helps retrieve fast and accurate locations of the ships. This allows better predictions from Extended Kalman Filter since these models rely on the quality of past vehicle trajectories. The difference between this paper and the current project is that the AIS data has a much higher frequency rate than that of this project's 10Hz. The higher the frequency, the better the predictions.

Preprint. Under review.

1.3 Problem C

The prediction task for an LSTM and a Kalman Filter are quite different. Since Kalman Filters are purely model based algorithms, there is no need for a training step. However, the LSTM model can be described in 4 sections: training data, model selection, loss function, learning objective. [4]

1) The training data of inputs and outputs of each city are given in the format:

$$S = \sum_{i=1}^{C} \{\mathbf{X}_{ij}, \mathbf{Y}_{ij}\}_{j=1}^{N}$$

X is a 50 by 2 matrix containing the previous 50 cartesian coordinates of the tracked vehicle.
Y is a 60 by 2 matrix containing the next 60 cartesian coordinates of the tracked vehicle.
N is the total number of tracked scenario in the specified city.
C is the total number of cities in the given dataset.
S is the full dataset with all inputs and outputs of each city

2) The model selection process consists of various time series estimators:

- Long Short-Term Memory Neural Networks
- Kalman Filter (Extended Kalman Filter)

3) The mean squared error is used as the loss function to measure how well the models predicted the expected outcome.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y - \hat{y})^2$$

4) The learning objective is used for deep learning time series estimators to optimize the model parameters with the simplest being gradient descent.

$$\underset{w,b}{argmin} \sum_{i=1}^{N} L(y_i, f(x_i|w, b))$$

$$w^{t+1} = w^t - \eta^{t+1} \partial_w \sum_{i=1}^{N} L(y_i, f(x_i|w^t, b^t))$$

$$b^{t+1} = b^t - \eta^{t+1} \partial_b \sum_{i=1}^{N} L(y_i, f(x_i|w^t, b^t))$$

The weights and bias of the neural network are updated using the weight from the current time step, w^t, subtracted from the learning rate, η^{t+1}, multiplied with the gradient to minimize the loss function.

The LSTM and Kalman Filters models can have many practical uses for guidance and navigation beyond the domain of cars. Since the input and output values are just (x,y) coordinates, any object that is capable of moving can benefit from these models. For example, they can be used as a tool to help predict the locations of aircrafts and ships that have gone off the radar for a period of time. By tracking back to their last locations, these models will give an accurate measurement of the potential aircraft/ship location.

2 Exploratory Data Analysis

2.1 Problem A

The dataset consists of 233659 data points from six different cities. It is divided by 203816 training and 29843 testing data points. Each training data consists of the features and labels which are the

previous 50 steps and the next 60 steps, respectively. On the other hand, each testing data only has the previous 50 steps. It will be the models goal to produce the next 60 steps of the testing data. This forms an input dimension of 50 by 2 and an output dimension of 60 by 2. Each row consists of one (x,y) coordinate. Since the data is collected on frequency of 10Hz, each second consists of 10 points. One data sample is shown below:

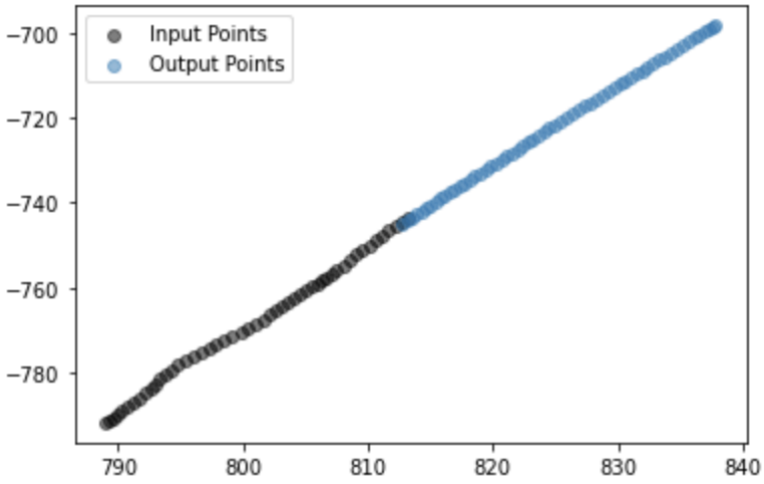

Figure 1: Sample input and output vehicle position

2.2 Problem B

The distribution of the data for each city is given below:

Palo Alto = {training: 11993, testing: 1686}

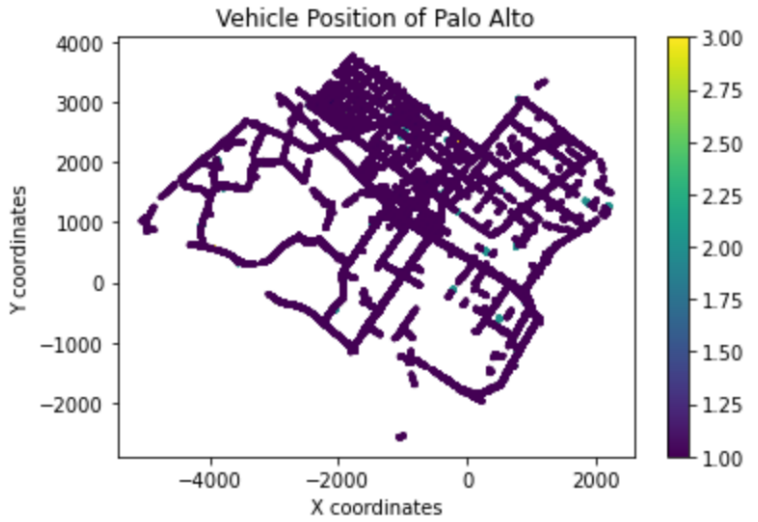

Figure 2: Vehicle position of Palo Alto

The Palo Alto coordinate graph consists of many sharp turns. From this observation, it can assumed that there is a high chance of many traffic intersections. Most of the tracked vehicle locations are connected to the main graph leading to less outliers.

Austin = {training: 43041, testing: 6325}

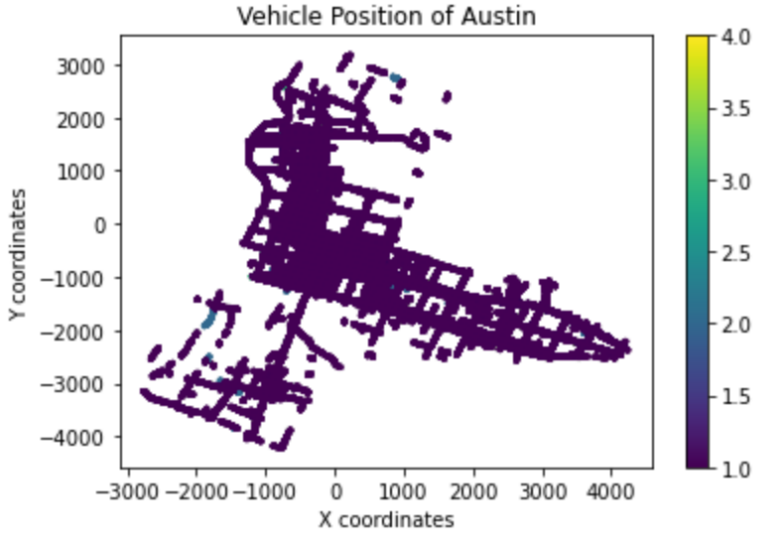

Figure 3: Vehicle position of Austin

The Austin coordinate graph can be divided into two different sections. The first section is L-shaped with a high concentration in the center compared to the second section located in the bottom left corner. Higher concentrated areas are likely to be easier for models to predict because the models will have more data to train with in that specific area.

Miami = {training: 43544, testing: 6361}

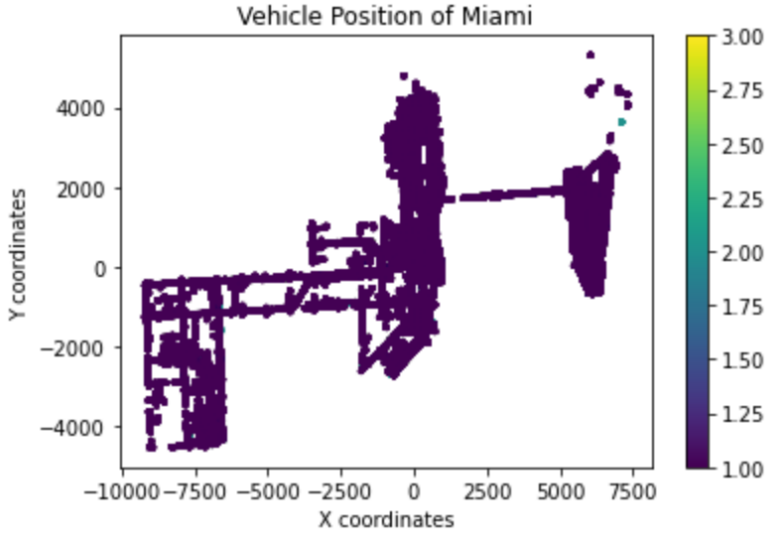

Figure 4: Vehicle position of Miami

The Miami coordinate graph is extremely wide in comparison to the rest of the cities. There are two paths that connect the middle section of the city together with the left and right side. These linear paths allow models to increase focus on velocity rather than direction to predict the vehicles location.

Pittsburgh = {training: 24465, testing: 3671}

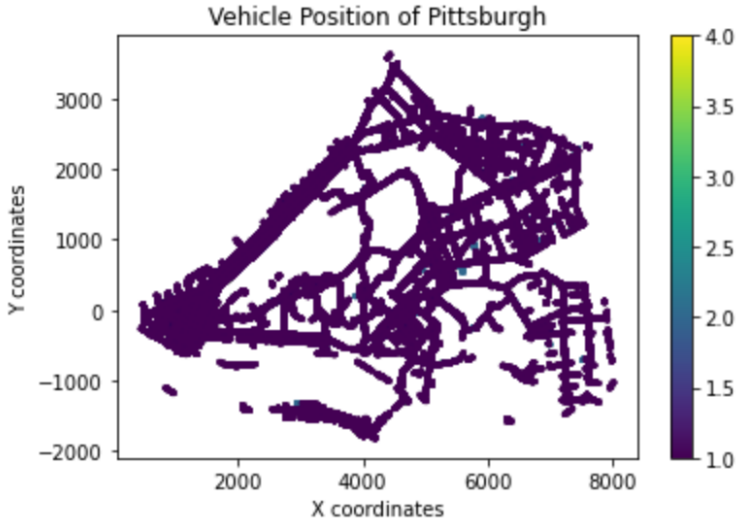

Figure 5: Vehicle position of Pittsburgh

The Pittsburg coordinate graph contains a pretty evenly distributed graph of tracked vehicles. There are a couple spots in the graph that are completely disconnected from the main vehicles of the road. The roads have many intersections which may lead to more difficulty when creating a prediction model.

Dearborn = {training: 25744, testing: 3829}

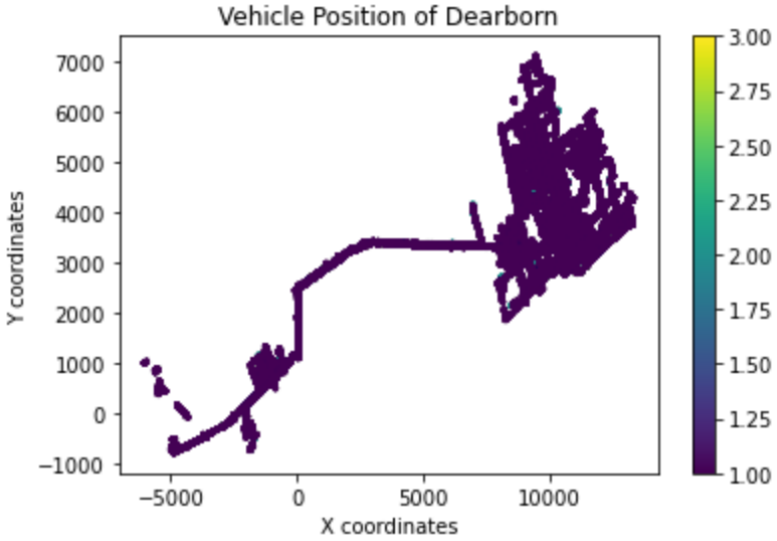

Figure 6: Vehicle position of Dearborn

The Dearborn coordinate graph has a highly condensed area around the top right corner. The rest of

the graph displays relatively straight paths with little to no curvature. There are still some points on the graph that are noticeably disconnected to the rest of the points.

Washington DC = {training: 55029, testing: 7971}

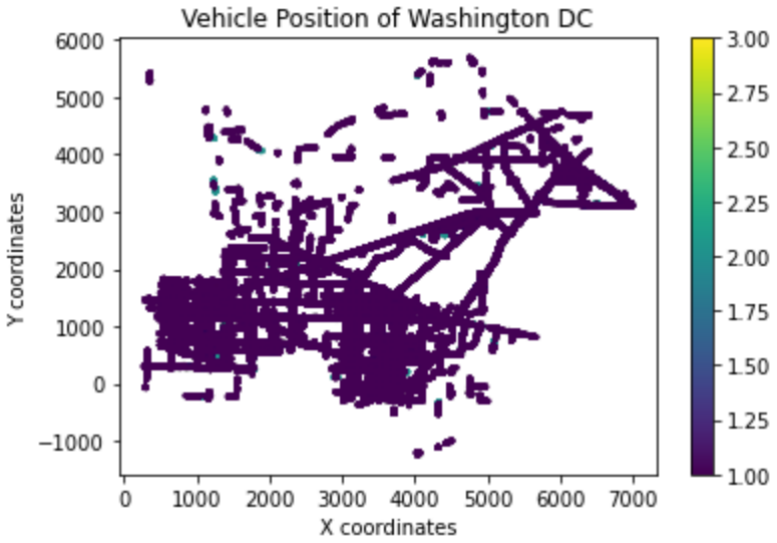

Figure 7: Vehicle position of Washington DC

The Washington DC coordinate graph contains many points that are disconnected from the main graph. This can be the result of insufficient data which can lead to misrepresentation of the true results. In the highly concentrated areas of Washington DC, there seems to be many sharp turns and intersections.

The total distribution of output and input positions for all agents are shown below.

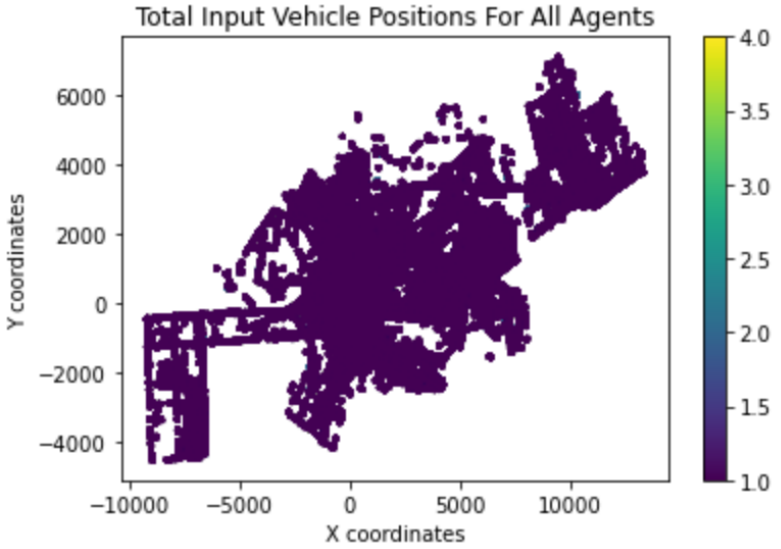

Figure 8: Total input vehicle positions for all agents

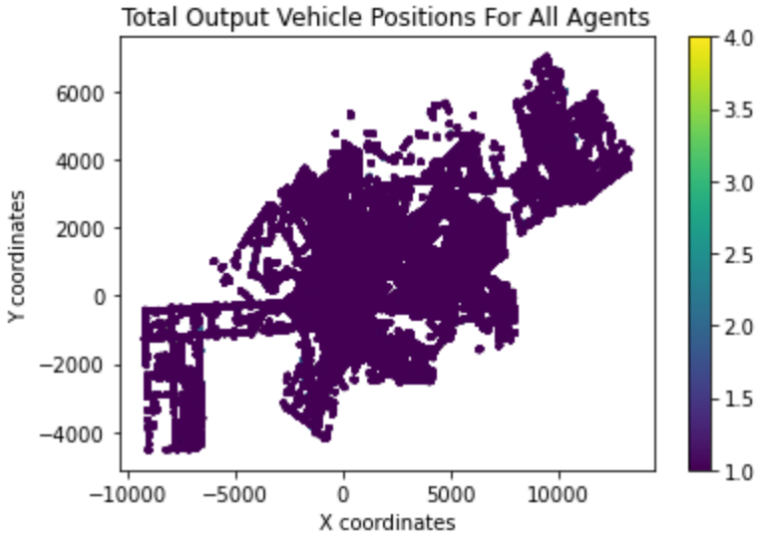

Figure 9: Total output vehicle positions for all agents

Figure 8 and Figure 9 displays a combination of the total input and output vehicle positions from six different cities. This graph has two blank areas, one on the top left and bottom right. Based on this map, the cities overlap with one another indicating that the coordinates were not unique to each city. The prevent misrepresentation of the data, there needs to be separate models for each city.

2.3 Problem C

There were four main steps to data processing: training/validation data split, feature engineering, normalization, city based tuning

Training/Validation Data Split: The training and validation data was split in an 8:2 ratio. This provided the right amount data for training without decreasing the performance. If the size of the training data was too small, this could potentially reduce performance. On the other hand, if the size was too large, the validation set would not be representative of the testing set.

Feature Engineering: This project did not require any new features but it changed the output size of each sequence. Instead of producing 60 steps, the model produced 1 step for each input by using the rolling window technique. The deep learning model found the local minimum quicker, but it took a longer training time for each epoch.

Normalization: Scikit-Learn's Min-Max Scaler was used to normalize the data from 0-1. After training the model with scaled data, the inverse transformation was applied to the output data. This ultimately helped reduce the time it took for the model to find the local minimum during gradient descent.

City Based Tuning: Each city had a different approach when creating the models. For cities that have straighter roads like Dearborn, I applied less (x,y) acceleration variation to the Kalman Filter model. On the other hand, cities with many turn like Pittsburgh required more (x,y) acceleration variation to account for turns.

3 Machine Learning Model

3.1 Problem A

A simple machine learning model used for this project was the Kalman Filter. This model uses the previous time steps to predict the next time step. Thus, no prior training is required. The input

features are the 50 input points with each having an (x,y) coordinate. There is no extra features needed as the filter computes the velocity and acceleration within the algorithm. This allows the next 60 can be generated in linear time, $\Theta(n)$.

The Kalman Filter is a regression model that can be adjusted using any loss function. The loss function used in this project is mean squared error. Depending on the mean squared error, the input for the next time step is adjusted to minimize the next loss.

3.2 Problem B

The deep learning model used in this project is LSTM. There were two main models used to predict the 60 steps. For each model, various hyperparameters were tested to find the one that produced the least mean squared error. The descriptions of both models are shown below.

LSTM Model 1: The input features of the first model were the initial 50 (x,y) coordinates. In this specific model, there were no normalization or window shifting done to it. The architecture consisted of two hidden layers with each layer having size 100. The learning rate was 0.001. This allowed for sufficient complexity without a long training time. The final output was a 60 by 2 tensor with each row consisting of an (x,y) coordinate. The loss function during the training was the mean squared error.

LSTM Model 2: The input features of the second model were the initial 50 (x,y) coordinates as well. For this model, the data was normalized using Scikit-Learn's Min-Max Scaler and the window shifting technique was used. Thus, the final output was a 1 by 2 tensor that contained only 1 (x,y) coordinate. This required generating 60 separate output values from the model. The window shifting technique requires approximately 60 times the regular training time for an epoch, so the architecture for this model was only one hidden layer with a size of 50 and a learning rate of 0.01. The loss function during the training was the mean squared error.

3.3 Problem C

Two main models used to make predictions were Kalman filter and LSTM.

Kalman Filter [3]

- An iterative algorithm that is used to estimate the position of an object using previous information
- Contains two main steps for motion forecasting, the prediction phase and correction phase
- In the prediction phase, the model predicts the state estimate and error covariance which are based on the previous state estimate and error covariance
- Statistical noise in the form of a Gaussian distribution is added to the state estimate and error covariance
- In the correction phase, the Kalman gain is computed to help adjust the state estimate and error covariance. This process is then repeated.

The algorithm of a Kalman filter is displayed below.

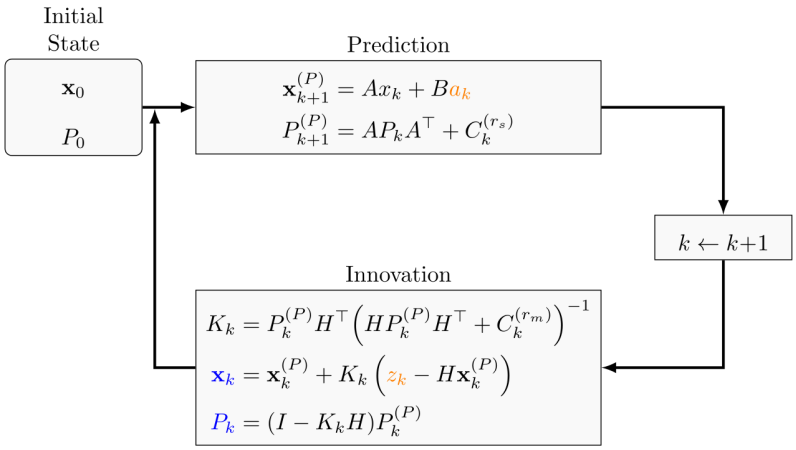

Figure 10: Kalman filter algorithm

$\mathbf{x}_k$, P_k, K_k are the state estimate, error covariance, and Kalman gain at time step k, respectively.

Long Short-Term Memory Neural Network (LSTM) [4]

- A type of recurrent neural network (RNN) that works well with time series predictions
- Three components to an LSTM which are the forget gate, input gate, and output gate
- Solves exploding and vanishing gradients by efficiently removing unnecessary information from the previous time steps in the forget gate
- Retrieves important information in the input gate and passes in along the output gate
- Output gate consists of the cell state and hidden state
- The cell state is considered the "long term" memory, while the hidden state is the "short term" memory
- Necessary parameters are number of hidden layers, hidden layer size, learning rate, optimizer, and dropout

Implementing a dropout in the model did not create better prediction because the number of hidden layers and size were relatively small to begin with. Dropout is usually effective when the there hundreds of hidden layers. The architecture of an LSTM is displayed below.

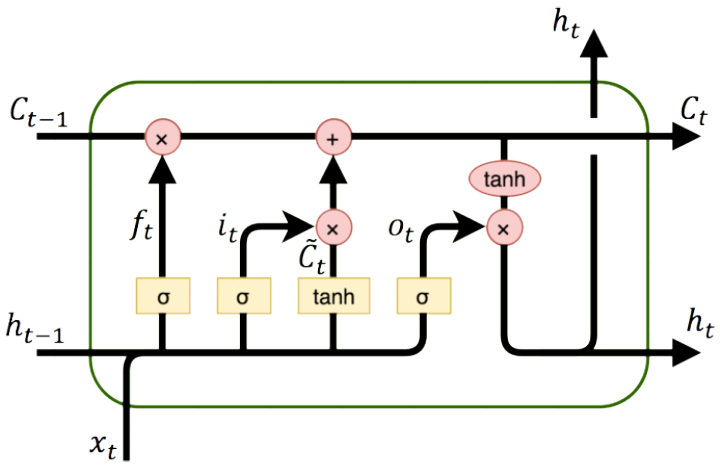

Figure 11: LSTM architecture

x_t, h_{t-1} and C_{t-1} are the input, hidden state, and cell state at time step t, respectively. Information is then passed along to the forget gate, f_t, and the input gate, i_t. This information is then passed through the output gate, o_t, for the next time step, $t + 1$.

4 Experiment Design and Results

4.1 Problem A

Computational platform/GPU for training/testing information

The deep learning models use a single GPU and an 8-core CPU with 16GB of ram from the UCSD's Data Science and Machine Learning Platform to train and test the model. GPUs allow multiple calculations to be made simultaneously which significantly improves the training process. Multiple GPUs can be used together to preform even faster calculations. However, the given resource for this project was a single GPU for training and testing.

Optimizer and learning parameter information

The deep learning model uses the Adam optimizer instead of the traditional stochastic gradient descent. The Adam optimizer has an adaptive learning rate with a momentum algorithm for a better convergence. Momentum helps find the direction of convergence by accumulating the gradients from past steps. For LSTM Model 1, the optimal learning rate turned out to be around 0.001. This learning rate was large enough to converge in 20 epochs but not large enough to miss the local minima. For LSTM Model 2, the optimal learning rate turned out to be around 0.01. Since this model took a longer time for training, only 10 epochs were used. Even with ten epochs, the momentum from the Adam optimizer helped quickly achieve the local minima.

Prediction implementation information

In order to make multistep predictions, there were two different methods used by separate models.

LSTM Model 1: The second model is trained on the original dataset that contains 50 points as the input and 60 points as the output. Both the input and output are flatten for easier computations. This form of training requires less overall calculations leading to faster training times for each epoch. The sacrifice to this faster training time is the large number of epochs required to make accurate predictions. Since the model is trying to predict 120 points, it needs to run through the data many more time than the first model.

LSTM Model 2: The first model is trained based on a sliding window technique. The model requires a flatten data of 50 points as the input and return an a flatten data of 1 point as the output. This allows the model to learn each point individually and provide accurate predictions one point at a time. However, this form of training requires drastically increases the training time. The number of training epochs much be sacrificed when using this method.

Training based on location information

There were two different approaches to using the city locations as part of the model design.

One model encompassing all cities: One approach is to create a single model that can be used across all cities. However, this approach is prone to having high bias and variance. From the data exploration, it is worth noting that the cities did not have unique (*x*,*y*) coordinates. This can potentially create difficulty for the model to recognize the patterns across each input. A strange overlap of two sequences from two different cities may cause wrong predictions in the testing phase. A single model does come with benefits. Only one model's parameter needs to be tuned. This decreases the overall time need in parameter tuning. Furthermore, there is overall more data for the model to learn from.

Separate models based on each city: The other approach is to create a separate model for each city. By creating specific models tailored toward a single city, the complexity of model such as number of hidden layers and size can be reduced. Even with less complexity, the model produces the same or even better results as the other model. By decreasing the number of hidden layers and size, the training time is reduced significantly. The downside to having separate models is the manual labor required to tune the learning parameters for each model. Since every city is different, different parameters are required to produce accurate predictions. 6 separate models were created for each city in this project.

Epoch and batch-size information

For LSTM Model 1, 20 epochs with a batch-size of 50 were used during model training. For LSTM Model 2, 10 epochs with a batch-size of 50 were used during the model training. The LSTM Model 1 was able to converge within 20 epochs indicating enough time for training. The batch-size of 50 was the a nice balance between batch and stochastic gradient descent. This helped with generalization without significantly slowing down the training time. The average time for one epoch was around 60 seconds for LSTM Model 1. The average time for LSTM Model 2 for one epoch was around 3600 seconds. The limitation in training time and GPU prevented more epochs in training.

4.2 Problem B

Prediction Performance of Different Models

	Kaggle Private	Kaggle Public
LSTM Model 1	165.97017	166.83031
LSTM Model 2	365.02955	367.29788
Kalman Filter	161.59184	162.87345
Kalman Filter with Data Knowledge	144.30619	145.28286

Figure 12: Prediction performance of different models

The results of LSTM Model 1 and LSTM Model 2 were not expected. Since normalization and window shifting was done to LSTM Model 2, I was expecting the scores to be higher. However, the vanilla LSTM model performed better. This might have been due to the low epochs of LSTM Model 2. Because the training cost was extremely high, it was harder for the model to train through more epochs. In terms of the Kalman Filter, it was pretty expected that these models would perform better than the LSTM models since the Argoverse dataset was reliable. Kalman Filters heavily rely on the quality of the data to have accurate results. The Kalman Filter with Data Knowledge required more time to predict values since it needed to find nearby points form the training data. It was expected to produce the best results.

The average time of LSTM Model 1 in minutes for one epoch is around one minute, while the average time of LSTM Model 2 in minutes for one epoch is around 56 minutes. In order to improve these training times, the hidden layers and size were reduced. The overall MSE did go up by around 5 when decreasing the hidden layers and size, but the training time reduce by almost 20 percent.

There are 5 parameters for both LSTM models: hidden layer, hidden size, epochs, learning rate, and optimizer. There are 5 parameters for Kalman Filters: acceleration in x-direction, acceleration in y-direction, noise magnitude, standard deviation of measurement in x-direction, and standard deviation of measurement in y-direction.

4.3 Problem C

Since the Kalman Filters do not have training phases, LSTM Model 1's training loss (MSE) is shown below.

```
epoch 0 loss: 45197.85139149468 time: 81.32127464748919
epoch 1 loss: 23118.283575672296 time: 60.786855711601675
epoch 2 loss: 12705.117485537836 time: 64.80841833818704
epoch 3 loss: 11193.30382433031 time: 60.6983584491536
epoch 4 loss: 8961.029735381488 time: 50.104679338634014
epoch 5 loss: 5312.688951102252 time: 52.6899180226028
epoch 6 loss: 3147.1090403181674 time: 51.222293325699866
epoch 7 loss: 1995.1636960600376 time: 52.779935609549284
epoch 8 loss: 1303.4354266623345 time: 54.49712388496846
epoch 9 loss: 840.8809161731942 time: 57.29647368192673
epoch 10 loss: 515.2359067190431 time: 55.01913494616747
epoch 11 loss: 330.4647473304849 time: 53.19486269261688
epoch 12 loss: 221.82319317615503 time: 56.79449853207916
epoch 13 loss: 176.69021558731777 time: 51.492338351905346
epoch 14 loss: 116.1904770320521 time: 49.21750392578542
epoch 15 loss: 80.79067114598806 time: 54.19032142404467
epoch 16 loss: 78.84875121841287 time: 56.8934826599434
epoch 17 loss: 71.59967037834723 time: 55.30700240191072
epoch 18 loss: 54.56340833850421 time: 51.39958494156599
epoch 19 loss: 48.10866160628347 time: 52.105046501383185
epoch 20 loss: 59.16958322976315 time: 50.88373112399131
```

Figure 13: LSTM Model 1's MSE loss with time

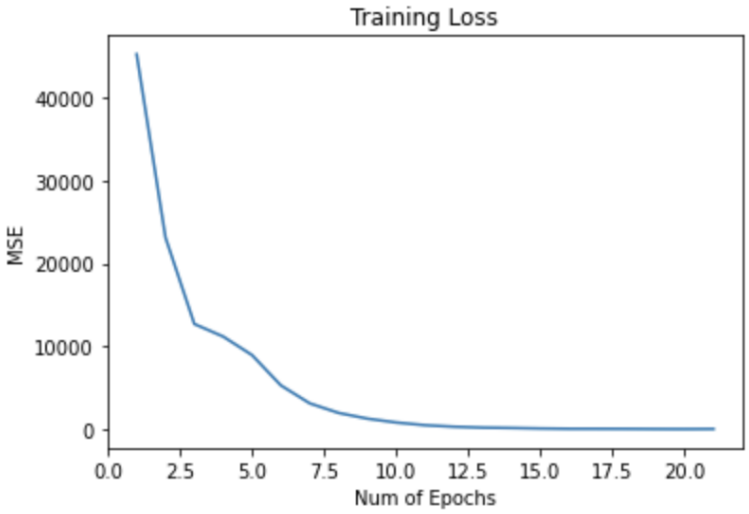

Figure 14: LSTM Model 1's MSE loss graph

Ground truth and prediction visuals

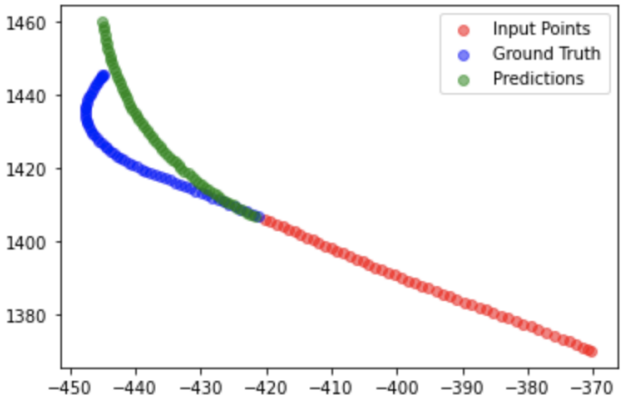

Figure 15: Kalman filter example

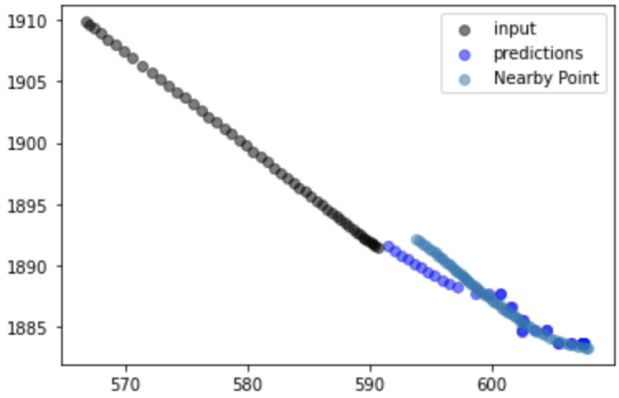

Figure 16: Kalman filter with data knowledge example

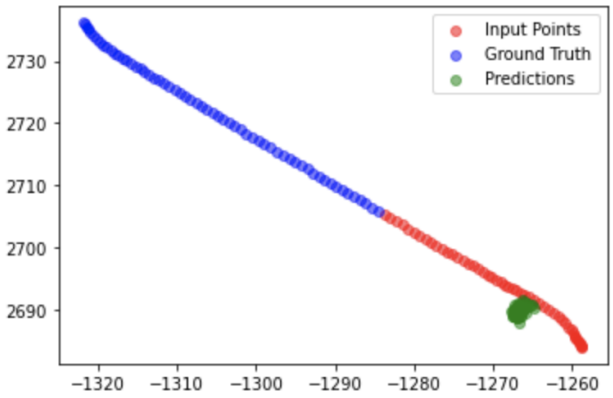

Figure 17: LSTM Model 1 example

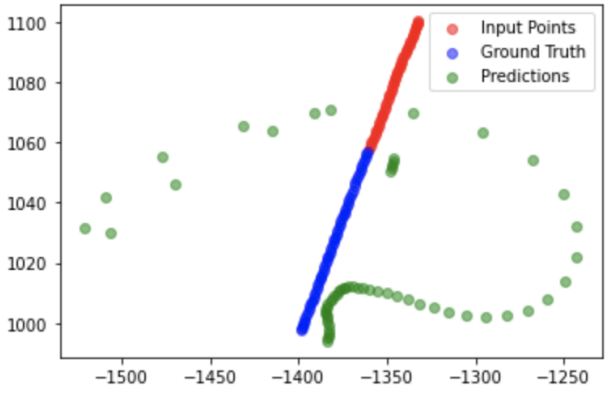

Figure 18: LSTM Model 2 example

Current ranking on leaderboard

- Rank: 34
- MSE: 144.30619 (Kalman Filter with data knowledge)

5 Discussion and Future Work

5.1 Problem A

From experimenting with different forms of feature engineering, I felt that the most effective feature engineering strategy was normalization. Although LSTM Model 2 did worse than LSTM Model 1 even with normalization, normalization helped find the local minima quicker. In the case of LSTM Model 1, more epochs were required since normalization was not done. I found that hyperparameter tuning was extremely helpful in improving my rank. The learning rate was especially crucial since a large learning rate would create an oscillation while a small learning rate would take forever to converge. The biggest bottleneck in this project was the computational platform. If I had access to better GPUs, it would have made it easier to experiment with more hyperparameters and model designs. I would advise a deep learning beginner to create a simple machine learning models such as MLP before diving into more complex models like LSTM. This can help create a stronger foundation with deep learning implementation. If I had more resources, I would like to dive deeper into the field of computer vision in relation to motion forecasting. The combination of computer vision and motion forecasting can largely benefit today's generation with autonomous vehicles.

References

[1] Guy Revach, Nir Shlezinger, Xiaoyong Ni, Adria Lopez Escoriza, Ruud J. G. van Sloun, Yonina C. Eldar, "KalmanNet: Neural Network Aided Kalman Filtering for Partially Known Dynamics" (2022), Institute of Electrical and Electronics Engineers (IEEE).

[2] S. Fossen and T. I. Fossen, "Extended Kalman Filter Design and Motion Prediction of Ships Using Live Automatic Identification System (AIS) Data," (2018) 2018 2nd European Conference on Electrical Engineering and Computer Science (EECS).

[3] Welch, Greg, and Gary Bishop. "An introduction to the Kalman filter." (1995)

[4] Staudemeyer, Ralf C. and Morris, Eric Rothstein, "Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks," (2019)