CSE 151B Project Final Report

Haitong Chen hac020@ucsd.edu
Michael Chen mic012@ucsd.edu
Sky Li yul055@ucsd.edu
Yining Gu yig019@ucsd.edu

1 Task description and background

1.1

Our task is to give a reliable prediction of the positions of a tracked Autonomous Vehicles (AV) 6 seconds into the future, given an initial 5-second observation of its trajectory, taking into account for the complex movement of traffic agents around the AV, such as cars, cyclists, and pedestrians.

As AV is welcomed by more and more people, the task of trajectory prediction becomes more and more important for its real-world impact:

- Driving safety: a reliable prediction can help the AV make better decisions even when facing complex traffic conditions like random obstacles, moving pedestrians, vehicle traffics, etc. Good predictions can help avoid those dangers by sensing them earlier.
- Route planning: a good prediction of the traffic condition can lead to a better route plan by taking into account the future traffic instead of purely the real-time conditions. One example can be planning a faster route that might be longer in distance but will be having fewer traffic.

As the importance being discussed above, while this task is an important topic in solving the forecasting problem in the academia, successfully solving this problem will have a positive impact in the society and people's daily lives. Some real-world examples are as below:

- Safety: When sensor on a 40mph AV saw a kid falling on its 6s future trajectory, it will be able to break immediately or dodge to avoid accident from happening.
- Convenience: On a crossroads, if the AV sense a huge traffic at the next traffic lights that it will be driving to, it can make a decision of going around and save time for passengers.
- Social Impact: Successfully solving this task gives pedestrians, cyclists and other drivers safe feelings of having AV on the road and wouldn't worry about getting hits on the roads.

1.2

Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art by Joel Janai, Fatma Güney, Aseem Behl and Andreas Geiger talks about computer vision application in capturing road conditions for prediction of trajectory. His paper digs into the topics of 'recognition, reconstruction, motion estimation, tracking, scene understanding, and end-to-end learning for autonomous driving'. (https://www.nowpublishers.com/article/Details/CGV-079)

Planning and Decision-Making for Autonomous Vehicles by Wilko Schwarting, Javier Alonso-Mora and Daniela Rus1 discusses some challenges and trend of the current autonomous driving industry. They proposed the use of Neural Network-based Perception Systems on motion planning. Also they introduced Game-Theoretic Approaches and Probabilistic Approach in searching for best decision under certain circumstances. (http://pdf.xuebalib.com:1262/3rhl2iRxzAso.pdf)

Preprint. Under review.

Table 1: Data size

City	Train data size	Test data size
Austin	43041	6325
Miami	55029	7971
Pittsburgh	43544	6361
Dearborn	24465	3671
Washington-DC	25744	3829
Palo-alto	11993	1686

1.3

We are given a large dataset to train our prediction model. Let (x_i^t, y_i^t) denote the observed traffic during time t in the i^th trajectory. The numbers of trajectories depend on the cities where we extract the data from. Given a sequence $\{(x_i^t, y_i^t)\}$ of observed traffic data, $t \in [0, 50]$, where we have 5 seconds and data is given every 0.1 second; $i \in [1, m]$, where m is the number of trajectories for each city, also known as scene. Define the dataset as S, where $S = \{(x_i^t, y_i^t)\}^N$, where (x_i^t, y_i^t) are the data points, and N denote the different cities (scenes). In this case, according to the data given, $N \in [1, 6]$ since we have data from 6 cities.

We define the model class as $f((x_i^t, y_i^t)|w, b)$, and the loss function as $L(y_i^t, \hat{y}_i^t) = RMSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^n (y_i^t - \hat{y}_i^t)^2}$. Our task is to predict the future trajectory of the AV. Say our prediction data is $\hat{y_i^t}$, where t maps the time of 6 seconds, and i maps the positions. Our goal, learning objective, is to train a good model that makes good prediction, so that we get $argmin_{w,b} \sum_i^N L(y_i^t, \hat{y}_i^t) f((x_i^t, y_i^t)|w, b))$.

From the mathematical abstraction, when used correctly, this model can potentially solve various types of forecasting problems as long as the input data can be represented as coordinate points of two variables.

Besides this project, the model can solve other potential tasks. One of the example is predicting index in economy include but not limit to CPI, interest rate and unemployment rate etc. Policy makes and regular investors can make decisions regarding the financial markets and the society in general on time with the forecast data. Another application is in medical field, where we could forecast the spread of current Covid-19 disease and distribute more vaccination and create better medical products to prevent further possible spread.

2 Exploratory Data Analysis

2.1

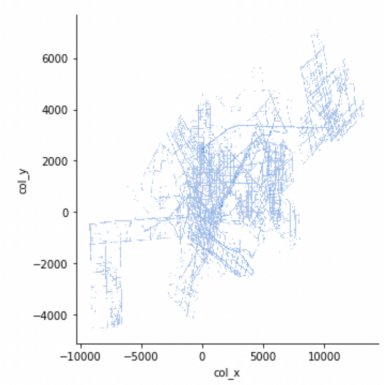
- The training and test data size of different cities are captured in the table. See Table 1.
- dimensions of inputs/outputs: each one data point represents the position at a certain 0.1 second time step in one of the trajectories in one of the cities. For each trajectory, the input dimension is $50 * 2$, where 50 denotes the 50 0.1 second in 5 seconds, and 2 denotes the two dimension of the position, x and y. The output dimension is the $60 * 2$, where 60 denotes the 60 0.1 second in 6 seconds, and 2 denotes the two dimension of the position, x and y.

2.2

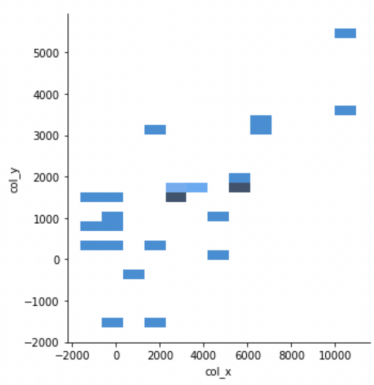
distribution of input positions for all agents
input of all agents: See Figure 1a

distribution of the output position for all agents
output of all agents: See Figure 1b

distributions of positions for different cities
Distributions of the input position for different cities:



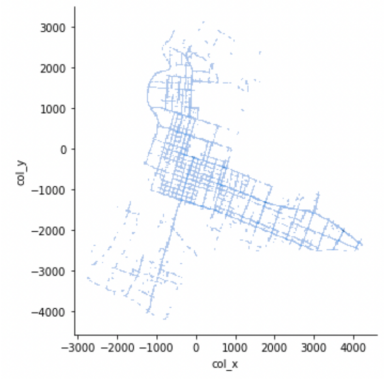
(a) Input distribution for all agents.



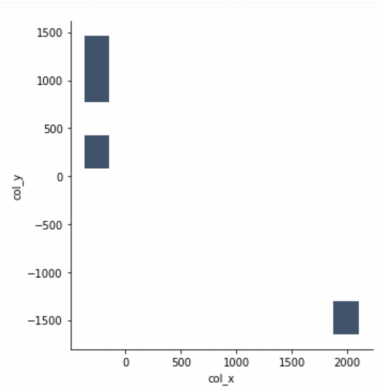
(b) Output distribution for all agents.

Figure 1: Input and output distribution for all agents.

input of Austin: See Figure 2a
output of Austin: See Figure 2b



(a) Austin input distribution.



(b) Austin output distribution.

Figure 2: Input and output distribution for Austin.

input of Miami: See Figure 3a
output of Miami: See Figure 3b

input of Pittsburgh: See Figure 4a
output of Pittsburgh: See Figure 4b

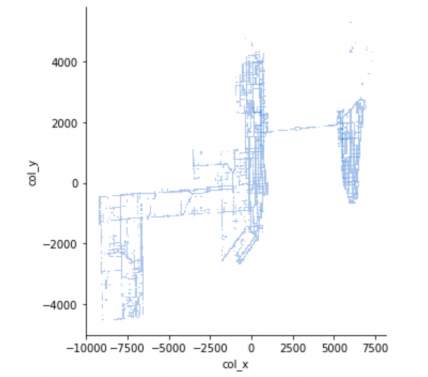
input of Dearborn: See Figure 5a
output of Dearborn: See Figure 5b

input of Washington-DC: See Figure 6a
output of Washington-DC: See Figure 6b

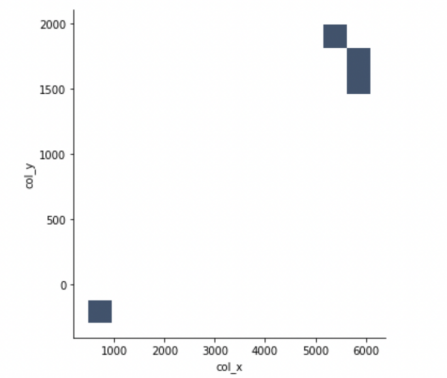
input of Palo-alto: See Figure 7a
output of Palo-alto: See Figure 7b

2.3

- How did you split the training and validation data? What is the size of your train and validation dataset?
  Because a large number of training data are provided, we split the training and validation data in the ratio of 10:1. Say the input size is s, then the training data size is $\frac{10}{11}s$ and the validation data size is $\frac{1}{11}s$.

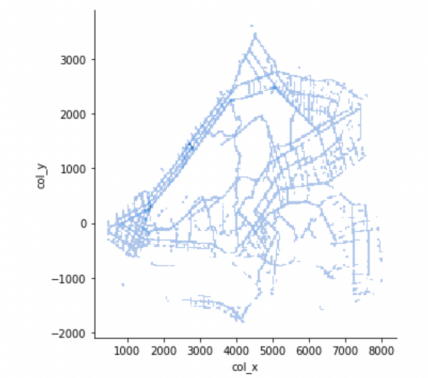


(a) Miami input distribution.

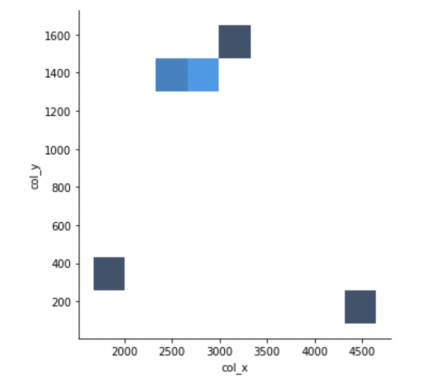


(b) Miami output distribution.

Figure 3: Input and output distribution for Miami.

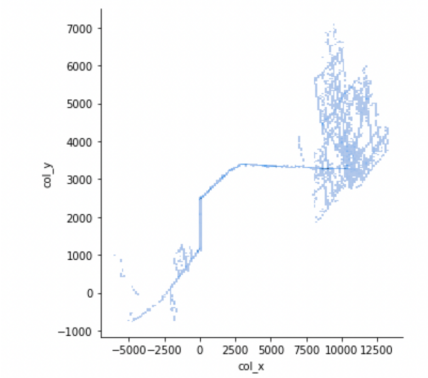


(a) Pittsburgh input distribution.

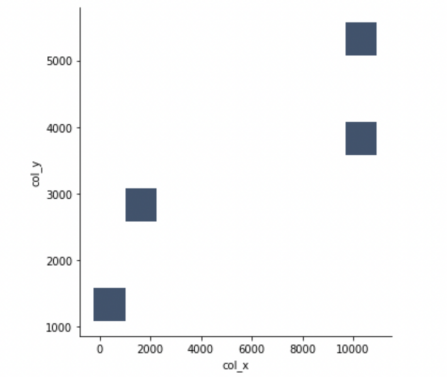


(b) Pittsburgh output distribution.

Figure 4: Input and output distribution for Pittsburgh.

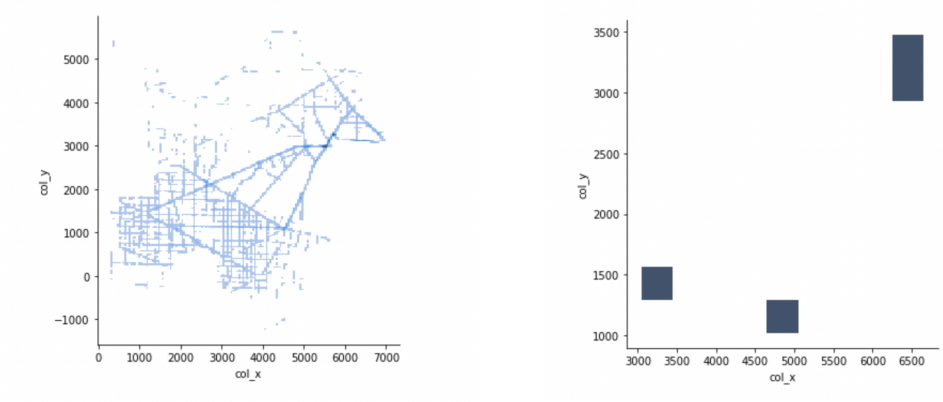


(a) Dearborn input distribution.



(b) Dearborn output distribution.

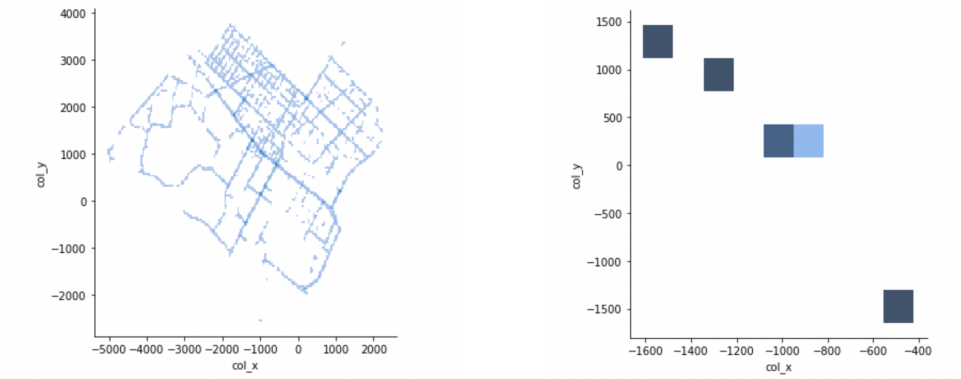
Figure 5: Input and output distribution for Dearborn.



(a) Washington-DC input distribution.

(b) Washington-DC output distribution.

Figure 6: Input and output distribution for Washington-DC.



(a) Palo-alto input distribution.

(b) Palo-alto output distribution.

Figure 7: Input and output distribution for Palo-alto.

Another future work that we can try to optimize our model is to adjust the ratio of the training and validation data size. Using the classic 5-Fold Cross Validation as a baseline, we can try to find a good ratio to separate the data so that we can train and test more efficiently and precisely.

- Did you use any feature engineering? If yes, how did you design your features? Explain your rationale.
  We have not yet incorporated feature engineering in this submission, in which we have extensively relied on the interaction of the layers and network. However, factoring in some more feature engineering is in our plan for the future work when we want to optimize this model even more. Having picked good features and incorporated good feature engineering vectors as input, the model is likely to perform better for the presence of insights and history.
- Did you use the city information provided in the dataset. If yes, how did you exploit this information.
  We didn't use the city information, but we merge all datasets into a big training set.

3 Machine Learning model

3.1

- What are the input output features that you end up using for prediction with simple machine learning models?
  Input is 50 timestamps of positions and output is 60 Timestamps trajectory.

- Which model class did you pick? What is your loss function?
  VAR is the model class and MSE is the loss function

3.1.1

- What are the input output features that you end up using for prediction with deep learning models?
  Input is 50 timestamps of positions and output is 60 Timestamps trajectory
- What is your model architecture and loss function? If you have multiple alternatives, discuss your ideas and observations.
  The model consists of the LSTM and MLP layers. The loss function is MSE. Hyperparameters are tuned using Grid Search. We have experiment with 1) transformer model which suffers from slow training with our limited computation resources; 2) vector autoregression which suffers from failing to learn even basic physics rules; and 3) recursive LSTM which suffers from error propagation.

Grid search:

LR=[0.001, 0.0001, 0.00001]

LSTM layer=[2,3,4]

MLP layer=[2,3,4]

LSTM unit=[32,128,256]

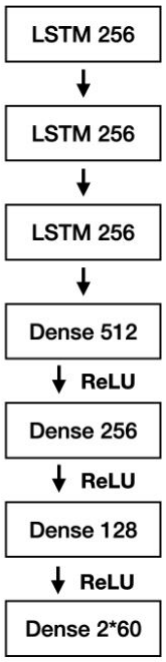
MLP unit=[128,256,512]

3.2

- Use an itemized list to briefly summarize each of the models, their architecture, and parameters, and provide the correct reference if possible.
  We have experimented with 4 different models.
  - Model 1: Temporal Fusion Transformer
    - Summary: This is based on a paper published on 2019, where an attention based model is purposed and outperforms many previous model in time forecasting.
    - Architecture: The model consists of a variable selection layer to select most salient features; a static covariant encoder with LSTM encoder-decoder architecture; a static enrichment layer with GRN; a temporal self attention layer that considered all outputs from all time stemp all at once; and finally, a GRN and dense layer to output.
    - Parameters: default parameters are used as suggested by the paper: 500 epochs, 2 LSTM layer, 64 hidden nodes, 3-headed attention layers and 32 batch size.
    - Reference: Lim, B., Arik, S. O., Loeff, N., Pfister, T. (2019). Temporal fusion transformers for interpretable multi-horizon time series forecasting. arXiv preprint arXiv:1912.09363.
  - Model 2: Vector Autoregression (VAR)
    - Summary: This is a statistical regression method to produce the future 60 data points.
    - Architecture: VAR generalize the univariate autoregression models for all its dimensions. For each variable in the vectors, an equation is used to model the variable's evolution over time using past value (lag).
    - Parameters: default parameters / NA
    - Reference: Toda, H. (1991). Vector autoregression and causality (Doctoral dissertation, Yale University).

  - Model 3: LSTM, recursive
    * Summary: This model uses LSTM layers to produce a single future output based on previous inputs. Recursively, it can produce all 60 trajectories.
    * Architecture: LSTM layer + Dense layer. The results are computed recursively.
    * Parameters: the model consists of 1 LSTM with 4 hidden units, and 1 dense layer with 2 hidden units. SGD is used, Adam optimizer is used.
    * Reference: N/A
  - Model 4: LSTM + MLP
    * Summary: The model consists of multiple LSTM and MLP layers. The input is 50 timestamps of positions and output is 60 timestamps. Compared to the previous model of recursive LSTM, we predicted the output at the same time. The average training cost turned out to be 2218s per epoch.
    * Architecture: a stacked LSTM which fed into a multi layer perception.
    * Parameters: The stacked LSTM consists of 3 LSTM layer each with 256 units; the MLP consists of 4 dense layers, stimulating an encoder architecture with 512, 256, 128 and 120 layers respectively.
      · optimizer: Adam
      · lr: 0.00001.
      · Loss function: MSE
      · batch size: 1 (SGD)
      · epoch: 30
    * Reference:
      https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html,
      https://pytorch.org/docs/stable/nn.html

- If you end up designing your own model architecture, include a picture/sketch of your model architecture. Explain why you choose such a model.



  We chose this model because multiple layers of LSTM and MLP make up a complex model that has more parameters to help us train better on our input and make more accurate predictions.

- Describe different regularization techniques that you have used such as batch-norm, dropout, and max-pooling in your model.
  We used sample wise norm, which was to normalize each sample's data points to the origin

(starting point) so we have relative positions instead. The data therefore, were transformed to float between 0 and 1. The advantage was that we could prevent gradient explosion problem.

4 Experiment design and results

4.1

- What computational platform/GPU did you use for training and testing?
  GPU: RTX2070 and GTX1080ti
- What is your optimizer? How did you tune your learning rate, learning rate decay, momentum and other parameters?
  Adam. Grid search. Learning rate decay, momentum and other parameters are set to default parameters of Adam optimizer and / or handled by the Adam optimizer. The following learning rates are tested before which best performing learning rate of 0.000001 is selected $0.3, 0.0003, 0.00003, 0.0000003, 0.0000001$
- How did you make multistep (30 step) prediction for each target agent?
  By extrapolation. The predicted coordinates of the previous step is used as input for future prediction.
- How did you utilize the city information?
  The city information is not considered as part of the input. All cities data are integrated into one dataset with large number of samples.
- How many epoch did you use? What is your batch-size? How long does it take to train your model for one epoch (going through the entire training data set once)?
  100 epoch were chosen as per the constrain of our computational resources available. We used SGD with 1 batch-size. It was trained in 7 mines for each epoch, and 700 mines total.
- Explain why you made these design choices. Was it motivated by your past experience? Or was it due to the limitation from your computational platform? You are welcome to use screenshots or provide code snippets to explain your design.
  Because we need uses extrapolation in generating outputs, minimizing error propagation plays a large role in our model design. To do so we decide to use SGD instead of batch gradient descent as it tends to better reach minima despite risking worse generalization. This is chosen also because we have a relatively large dataset. For number of epochs, we chose 100 because 50-100 epochs is commonly used in deep learning training. The reason is that below 50 epochs could cause under fitting while above 100 could cause overfitting and produce bad generation result. We experiment with 300 epochs, and observes an increase in testing loss after 200 epochs.

4.2

We have selected four of our models as representatives to discuss here. Model 1: Temporal Fusion Transformer; Model 2: Vector Autoregression (VAR); Model 3: LSTM, recursive; Model 4: LSTM + MLP. The details are compared below.

- Use a table to compare the prediction performances of different feature and model designs. What conclusions can you draw from this table. See Table 2.
  The 4th model, which is our final model is the best performed one regarding training accuracy and time. We can therefore conclude that multiple layers of two different classes: LSTM and MLP is effective in increasing our prediction accuracy, while it kept training time relatively low compared to other model due to our structure and data normalization.
- Provide an estimate of training time (in flops or minutes) for different models. What did you do to improve the training speed?
  Training time for TFT: NA (we never finish one epoch to print the time per epoch)
  Training time for VAR: NA (statistical model do not need training)
  Training time for recursive LSTM: 46s/epoch (single trajectory output) 2760s/sample (all 60 trajectories)
  Training time for LSTM+MLP: 2218s/epoch
  Strategy used: Nvidia RTX 20XX and newer GPU is equipped with tensor core which speed

Table 2: Prediction performances of different models

Model	MSE score	Problem
1	N/A	1) Computationally expensive. 2) If the learning rate is too large, the result will diverge, if the learning rate is too small, the training process is too slow
2	48.29499	1) Sparkly large loss data points increases the average loss dramatically. 2) No learning, defy basic physics
3	407.31436	Error propagation
4	19.09141	N/A, can still be improved

Table 3: LSTM recursive Parameters

Modules	Parameters
rnn.weight$_i h_l 0$	80
rnn.weight$_h h_l 0$	400
rnn.bias$_i h_l 0$	40
rnn.bias$_h h_l 0$	40
fc.weight	20
fc.bias	2

up the training significantly. However it's only designed/capable for doing 16bit matrix manipulation. Therefore we converted all the data to 16bit half precision fp to utilize the tensor cores.

- Count and report the number of parameters in different models.

  Number of parameters for recursive LSTM: 582 (See table 3)

  Number of parameters for LSTM+MLP: 1630200. (See Table 4)

Table 4: Number of parameters for LSTM+MLP

Modules	Parameters
encoder.weight$_i h_l 0$	2048
encoder.weight$_h h_l 0$	262144
encoder.bias$_i h_l 0$	1024
encoder.bias$_h h_l 0$	1024
encoder.weight$_i h_l 1$	262144
encoder.weight$_h h_l 1$	262144
encoder.bias$_i h_l 1$	1024
encoder.bias$_h h_l 1$	1024
encoder.weight$_i h_l 2$	262144
encoder.weight$_h h_l 2$	262144
encoder.bias$_i h_l 2$	1024
encoder.bias$_h h_l 2$	1024
mlp.layer0.weight	131072
mlp.layer0.bias	512
mlp.layer1.weight	131072
mlp.layer1.bias	256
mlp.layer2.weight	32768
mlp.layer2.bias	128
mlp.layer3.weight	15360
mlp.layer3.bias	120

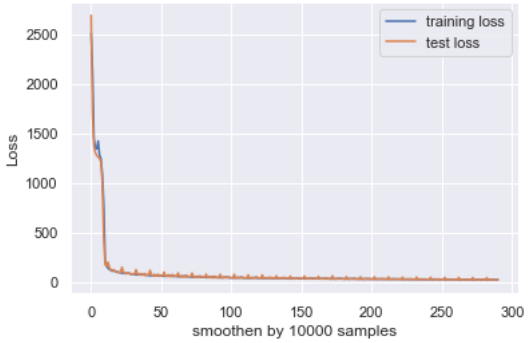


Figure 8: MSE over training steps.

4.3

- Visualize the training/validation loss (MSE) value over training steps (You should expect to see an exponential decay). See figure 8.

- Randomly sample a few training samples after the training has finished. Visualize the ground truth and your predictions. See figures 9

- Your current ranking on the leaderboard and your final test MSE.
  Our current ranking : 8 (7 if excluded number 1).
  Final test MSE: 19.09141

5 Discussion and future work

5.1

- What do you think is the most effective feature engineering strategy?
  Since we haven't integrated feature engineering in our model training, we answer this question based on our empirical experience. Most of the time, people used the raw feature to perform feature engineering, like the velocity and position of the vehicles. However, the raw feature can also be further processed to train the model, like the square of velocity, the acceleration rate, the position relative to other agents, etc.
  Sometime people also explore more features that don't look intuitive at the first glance, like the color of the vehicles, the weather, the model of the vehicles, the driving age of the drivers, etc (if can be obtained). Although those features can be obscure to the problem at first, it might turn out to be closely related and can bring out further interesting hypothesises.
- What techniques (data visualization/model design/hyper-parameter tuning) did you find most helpful in improving your ranking?
  Speaking of the whole competition process, data visualization helps us improve our ranking the most. Although we have specifically made use of other techniques like different model design and tuning hyper-parameter at the end, visualizing the data helps us understand our models more.
  By visualizing the loss over steps and the final predicted trajectory output, we have more intuitive ideas about how well our models are fitting the problems, and what aspects in our models that we can improve on. By constantly monitoring the data, we make progress constantly and can adjust anything that went wrong in a more timely and more efficient manner.
- What was your biggest bottleneck in this project?
  The biggest bottleneck is to lower the rmse further given the limitation of time frame and equipment we have, and the caution to avoid overfitting.
- How would you advise a deep learning beginner in terms of designing deep learning models for similar prediction tasks.

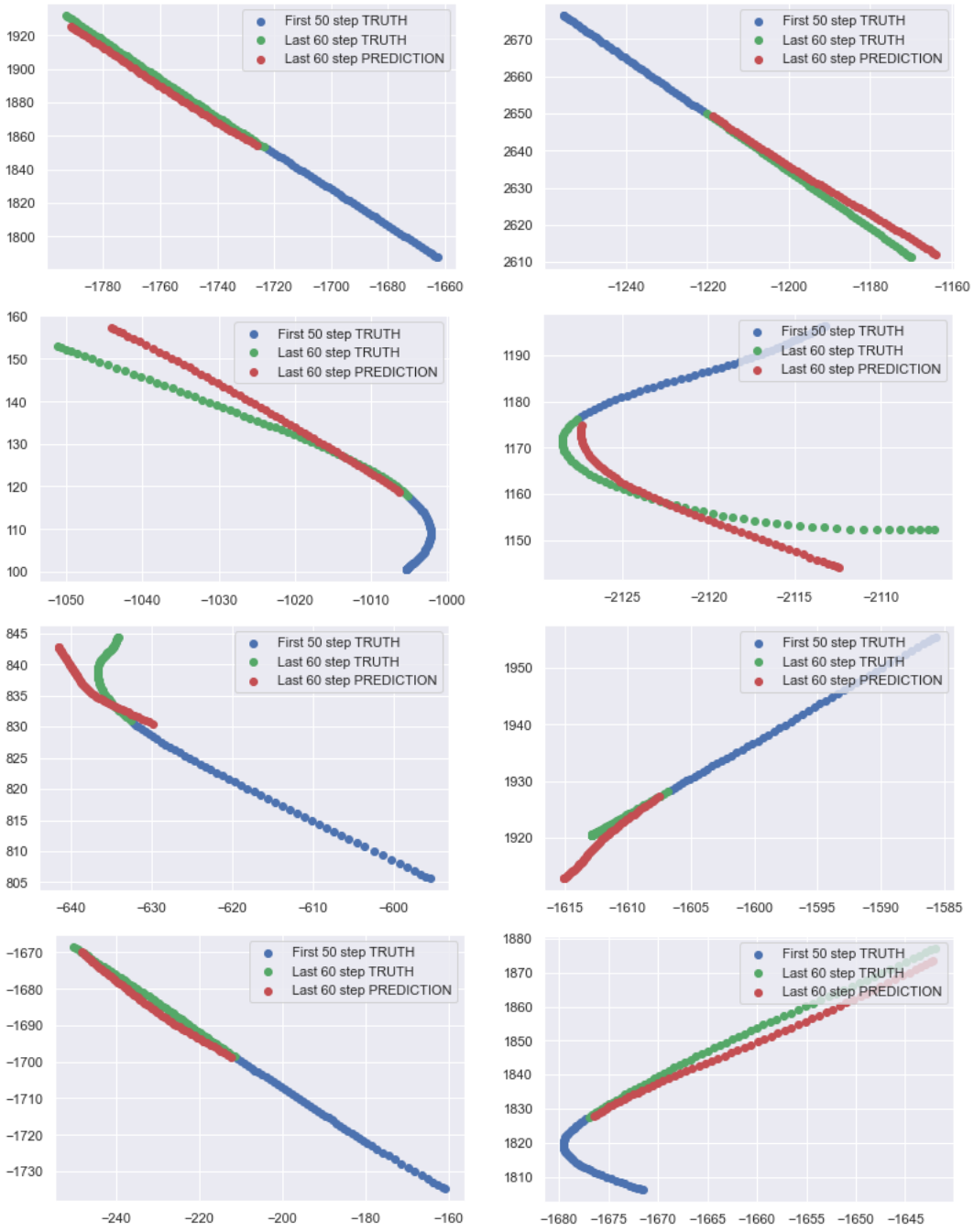


Figure 9: Training samples.

Our suggestion is to always start from simple models and add more features/layers/designs based on the performance of each try. We started from very simple models just to get started and have an initial score. With the initial score, we can compare our model's performance with others', and we have better ideas of what techniques work and what doesn't work. We gradually add more designs to our existing models or switch to other models if we find better models that can be adapted.

- If you had more resources, what other ideas would you like to explore?
  We will also like to try how to incorporate feature engineering with our model. Our current model doesn't involve feature engineering, but from what we can see from other scholar's claim, right feature engineering can be very helpful in forecasting problems.

References

[1] Alexander, J.A. & Mozer, M.C. (1995) Template-based algorithms for connectionist rule extraction. In G. Tesauro, D.S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems 7*, pp. 609–616. Cambridge, MA: MIT Press.

[2] Bower, J.M. & Beeman, D. (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* New York: TELOS/Springer–Verlag.

[3] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* **15**(7):5249-5262.

[4] Joel Janai, Fatma Güney, Aseem Behl, Andreas Geiger (2020) Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art *Foundations and Trends® in Computer Graphics and Vision*

[5] Wilko Schwarting, Javier Alonso-Mora, Daniela Rus1 (2018) Planning and Decision-Making for Autonomous Vehicles *Annual Review of Control, Robotics, and Autonomous Systems*