CSE 151B Project Final Report

Group: TechnoCore

Jiayan Dong
Undergraduate for Computer Science
University of California San Diego
La Jolla, CA 92093
jid001@ucsd.edu

Leo Cao
Undergraduate for Computer Science
University of California San Diego
La Jolla, CA 92093
z2cao@ucsd.edu

Hang Wang
Undergraduate for Computer Science
University of California San Diego
La Jolla, CA 92093
haw030@ucsd.edu

Github link
https://github.com/nonlighting/CSE-151B-Project

1 Task Description and Exploratory Analysis

1.1 Problem A

Task: Predict the positions of a tracked car 3 seconds into the future, given an initial 2-second observation.

With the development of AI technology and deep learning technology, self-driving car technology becomes possible. While this technology may bring great convenience to mankind, there are also many difficulties and risks. For example, distrust of computer decision-making and the danger of possible traffic accidents. Therefore, the core problem of autonomous vehicles is to accurately predict the position of the vehicle and other vehicles in the future based on the current road condition information and the location and speed information of the vehicle and other vehicles, so as to make correct and accurate decisions. Solving this task can have great impact on our daily life and the society. For example, people will trust autonomous vehicles more and laws and regulations can be passed to allow the further development of this field. Also solving this issue can also bring convenience and safety to people when driving. Neural networks and deep learning models can meet this demand well. Through the observation data in a large number of case data, the neural network can train a sufficiently accurate model to predict the future trajectory of the vehicle

1.2 Problem B

Predicting the future trajectories of on-road vehicles with a novel prediction framework called PRIME, which stands for Prediction with Model-based Planning. PRIME is designed to generate accurate and feasibility-guaranteed future trajectory predictions, which guarantees the trajectory feasibility by exploiting a model-based generator to produce future trajectories under explicit constraints and enables accurate multi-modal prediction by using a learning-based evaluator to select future trajectories.

Deep Stochastic IOC RNN Encoder decoder framework, DESIRE, for the task of future predictions of multiple interacting agents in dynamic scenes. DESIRE effectively predicts future locations of objects in multiple scenes by accounting for the multi-modal nature of the future prediction, the potential future outcomes and make a strategic prediction based on that, and reasoning not only from the past motion history, but also from the scene context as well as the interactions among the agents.

1.3 Problem C

Given the initial 19 positions of the target vehicle in 2 seconds as the input $P_{in}(x, y)[1 \ldots 19]$, output the predicted 30 positions of the target vehicle in next 3 seconds $P_{out}(x, y)[1 \ldots 30]$.

$$M(P_{in}(x, y)[1 \ldots 19]) = P_{out}(x, y)[1 \ldots 30]$$

Form this abstraction, it is possible for our model to potentially any time series prediction problem. Since our input is just a time series of position, the model can identify the trend of the time series and make predictions. So, our model may also be able to identify inventory quantities that change over time, stock trends or temperature changes, and so on.

2 Exploratory Data Analysis

2.1 Problem A

Size of the training dataset is 205942. Size of validation dataset is 3200

Each training scene contains the real position data of the first 1.9 seconds and the real position data of the last 3 seconds.

Input data dimension: 2, input data shape: (19,2)

Output data dimension: 2, output data shape: (30,2)

2.2 Problem B

Figures 1 to 4 show the distributions of the input and output data.

We mainly found that the speed of the target vehicle is mostly concentrated around 10, while some of the other vehicles are stationary and some have a speed of around 10.

We can also notice that the input and output location data are basically distributed in two different areas, which is not conducive to our prediction of location changes. So we preprocess the input data to convert the absolute position into a relative position (relative to the position of the first occurrence). Figure 5 and 6 show the distribution of the relative position of the input and output.

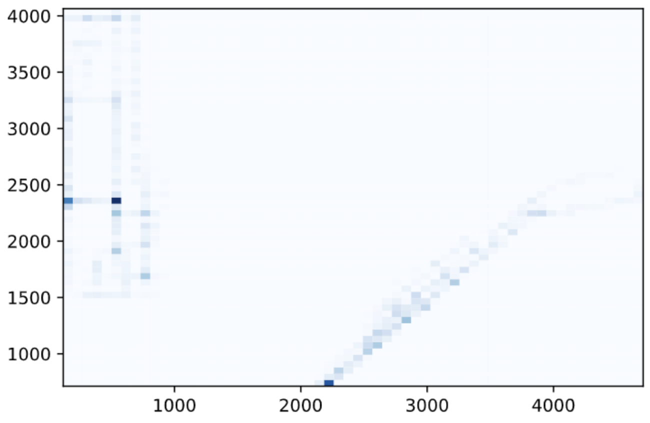


Figure 1: Distribution of input absolute positions.

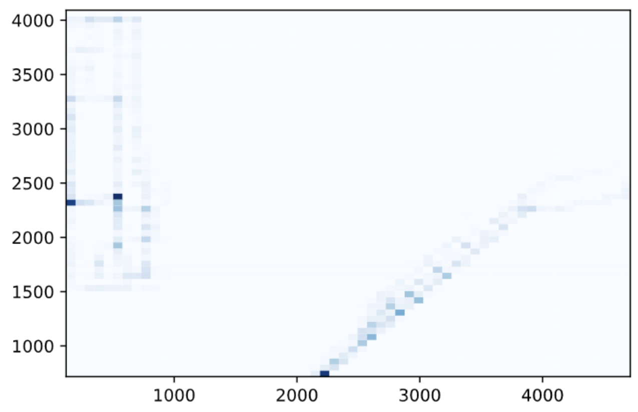


Figure 2: Distribution of output absolute positions.

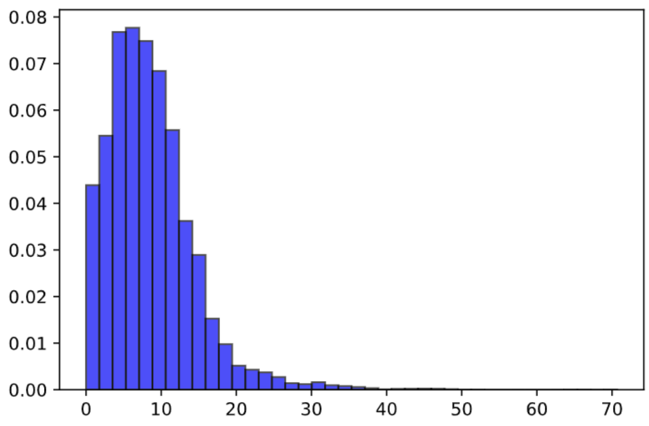


Figure 3: Distribution of velocity (magnitude) of the target vehicle.

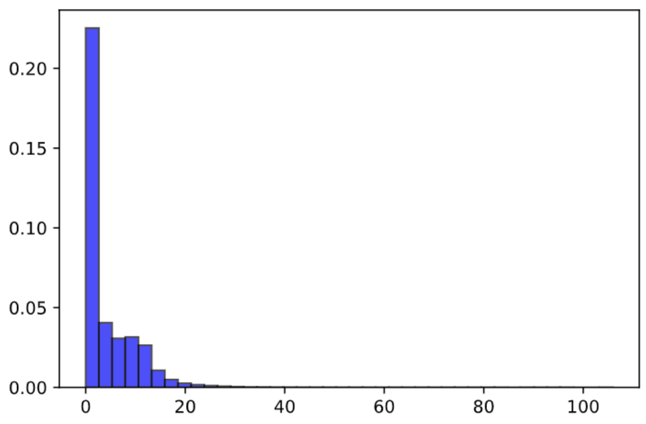


Figure 4: Distribution of velocity (magnitude) of other vehicles.

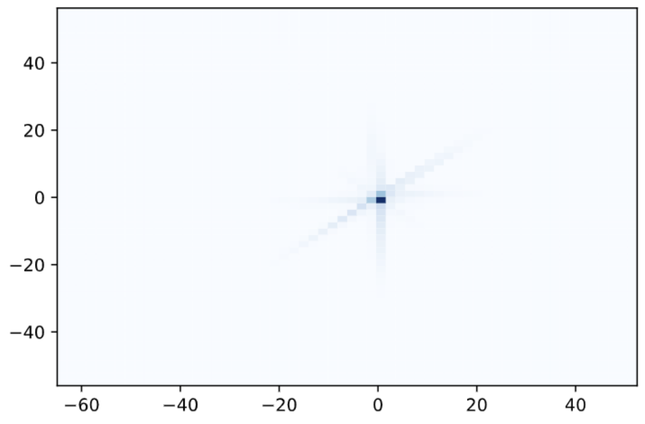


Figure 5: Distribution of input relative positions.

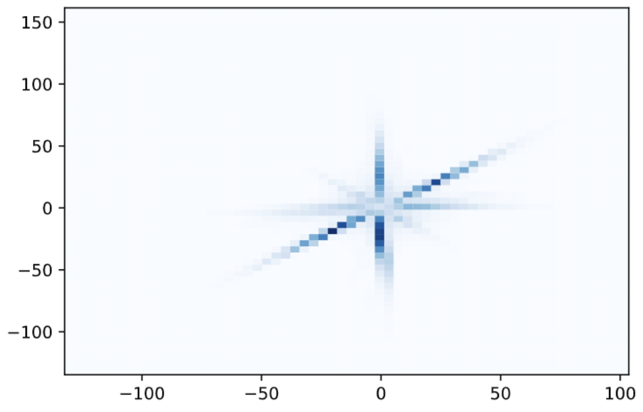


Figure 6: Distribution of output relative positions.

2.3 Problem C

We can now notice that the relative input positions are all concentrated around ±20, while the relative output positions are all concentrated around ±30. This is in line with our speed data. We also noticed that the positions of some vehicles only change on a single axis, while the positions of other vehicles change at the same time on both axes.

For feature engineering, we only pick the position feature as it is the feature we need to predict. We also convert all the position information into a relative position relative to the first position.

In the actual neural network model, the model that uses the relative position after experiment is better than the model that uses the absolute position. Therefore, before model training, we preprocess all positions as relative positions, and then convert relative positions to absolute positions during prediction.

We also tried to normalize all training data based on input length, but this had a serious impact on our model and made it impossible to train. In the end, we gave up the use of normalization and chose to use only relative positions.

We didn't use lane information provided in the dataset. Our model shows that only location information can output good prediction results.

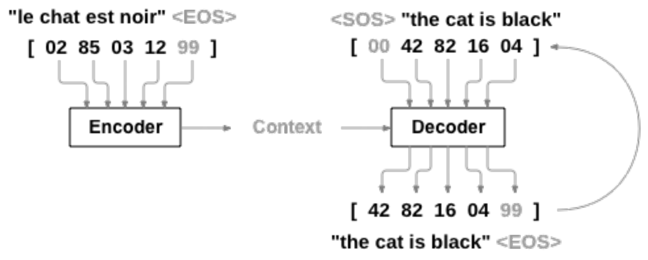


Figure 7: Sequence-to-sequence Model

3 Deep Learning Model

3.1 Problem A

The input now is the initial 19 relative positions of the target vehicle in 2 seconds, and the output now is the next 19 relative positions of the target vehicle in next 3 seconds.

Our goal is to predict time series, so we first thought of Recurrent neural network (RNN) because it can handle time series well. Since we need to predict a time series, so we use RNN with LSTM. Because LSTM can record information in the time series. Then we noticed that the input and output lengths are not the same, so we naturally thought of using Sequence-to-sequence (Seq2Seq), also called Encoder-Decoder model. Our Seq2Seq uses LSTM to store memory, and outputs sequences of unequal length through Encoder and Decoder.

Our Encoder consists of a linear layer and a 4-layer LSTM. Encoder accepts an input of size $(19, \text{batch size}, 2)$, converts it to size $(19, \text{batch size}, 128)$ through a fully connected linear layer, and finally passes through four layers in LSTM, and then converts to size $(4, \text{batch size}, 256)$, get output, hidden state and cell state. Among them, we use ReLu in the linear layer to speed up, and use dropout in LSTM to prevent overfitting.

Similarly, our Decoder consists of two linear layers and a 4-layer LSTM. Encoder accepts an input of size $(\text{batch size }, 2)$, converts it to size $(\text{batch size}, 128)$ through a fully connected linear layer, then adds the hidden state and cell state of the Encoder, passes through four layers in LSTM, and then gets The output of size $(1, \text{batch size}, 256)$ is finally converted to the prediction result of $(\text{batch size}, 2)$ through the linear layer.

Figure 7 is a example of using Seq2Seq in language translation, this model can also be used for time series forecasting.

Our loss function is MSE, and the square root of loss is used to calculate the final Kaggle score.

3.2 Problem B

We tried the linear regression model at the beginning, and tried to use the time series of p_x and p_y separately, and the result was about an RMSE value of 10.This linear regression model has a good performance when the vehicle is driving at a constant speed, but it is only based on each individual scene and has no learning ability. Its predictions of other scenes are pretty bad. Figure 8 shows the linear regression model predict a vehicle with constant velocity.

Then we used the Seq2Seq model, and at the beginning we tried a smaller-scale neural network. We used an embedded layer with a size of 8 and two hidden layer with a size of 16. This model is fast to train, but it cannot accurately predict the result. The training loss of this model cannot converge below 100, which means that our neural network is too simple and underfits the real model. Then we increased the neural network step by step, and finally got a more satisfactory model when the size of the embedded layer was 128 and the size of the 4 hidden layers was 256. On the validation set, it has an RMSE of about 3.8, which is a pretty good result.

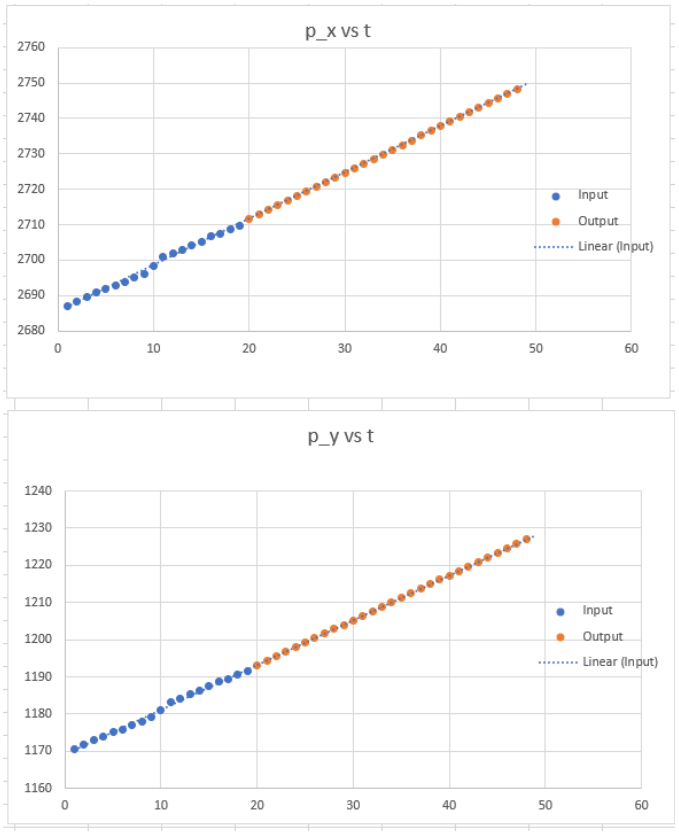


Figure 8: Linear Regression

Due to the long time series that needs to be provided, we use teacher forcing here to prevent continuous predictions with increasing deviations during training. The probability of Teacher forcing is set to 0.5.

This is a example of using teacher forcing in RNN. The predicted $t + 1$ result is replaced by actual result by 50% probability, in order to prevent the prediction deviation from getting bigger and bigger, and speed up the training process at the same time. However, later we discovered teacher forcing caused a overfitting problem, therefore in our final model we did not use it.

4 Experiment Design

4.1 Problem A

The training process and the test process are carried out on the personal computer and the UCSD datahub at the same time. Personal computers use GTX 1060 GPU for tensor acceleration, while UCSD datahub uses GTX 1080Ti GPU for tensor acceleration.

The provided 205942 training data is converted into a training set and a validation set at a ratio of approximately four to one. Finally, use the 3200 validtion data as the test set to verify the model results.

Use smaller data sets on personal computers, 20,000 training data and 5,000 validation data. In UCSD datahub, all data is used, namely 160,000 training data and 45942 validtion data.

Use Adam optimizer to train the model. After many attempts, a learning rate of 0.001 was used on a personal computer, and a learning rate of 0.0001 was used on the UCSD datahub. Other parameters are provided by pytorch by default, and the weight decay is 0. The learning rate is determined by observing the loss curve. In a personal computer, a learning rate of 0.001 can make the train loss close to convergence after 20 epochs, while in the UCSD datahub, if the learning rate of 0.001 is still used,

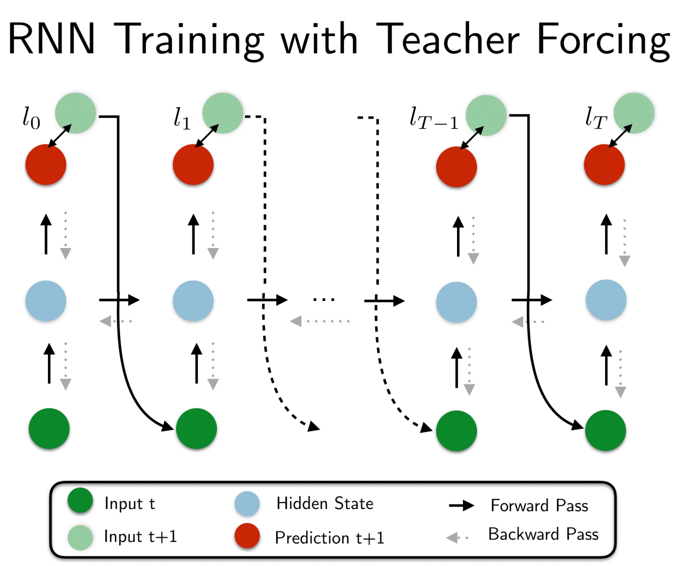


Figure 9: Teacher Forcing on RNN

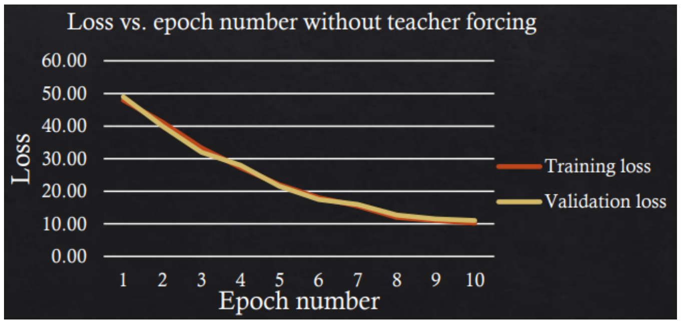


Figure 10: loss vs. epoch

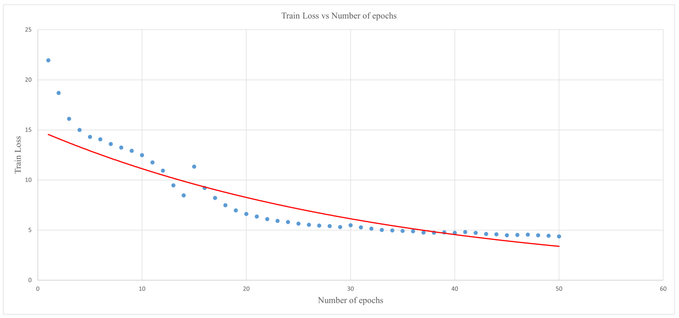
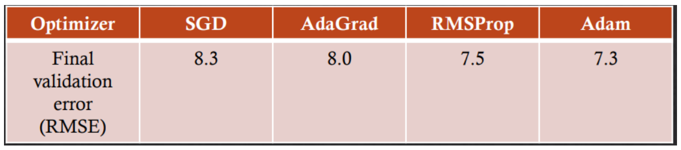


Figure 11: Train Loss



Optimizer	SGD	AdaGrad	RMSProp	Adam
Final validation error (RMSE)	8.3	8.0	7.5	7.3

Figure 12: Final error with different optimizer

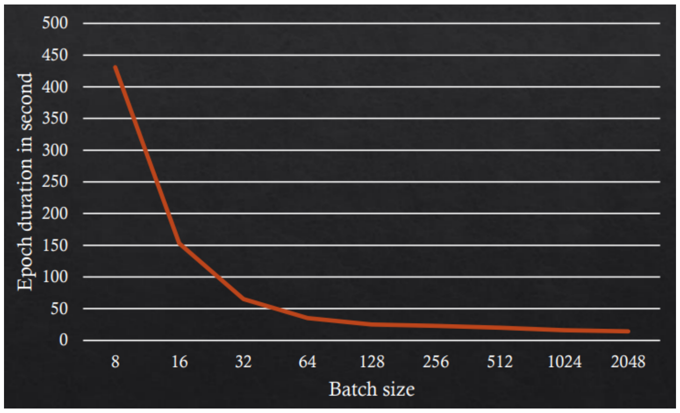


Figure 13: Average one epoch duration vs. Batch size

the train loss will drop rapidly after two to three epochs, which indicates that the learning rate is too high. High, which is also the result of the increase in the amount of data in a single epoch. Therefore, a smaller learning rate is selected on the UCSD datahub to control the train loss to converge after 20 epochs.

In order to make multistep (30 step) prediction for each target agent, we get the result by repeatedly using the output of the decoder. Every time the hidden state and cell state are input into the decoder, the output and updated hidden state and cell state can be obtained. Repeat 30 times to get the result forecast for the next 3 seconds (30 steps).

We used 50 epochs and batch size was 256. An epoch on a personal computer takes about 20s, and an epoch on a UCSD datahub takes about 7 minutes.

The number of epochs is determined by repeated attempts to determine whether the train loss can converge or fall sufficiently low within these epochs. The batch size is a larger value selected in consideration of the memory of the graphics card to speed up training.

5 Experiment Results

5.1 Problem A

Throughout the competition we used the linear model described before, 4-layer LSTM and 8 layer LSTM. We found out that the 4-layer LSTM gave the best RMSE. The 4-layer LSTM has 3,949,826 trainable parameters, 8 layer has 8,160,514 trainable parameters. However we found more trainable parameters does not mean better performance since overfitting can happen on deeper models. The training time for 4-layer LSTM is about 7 min per epoch on my local computer, and 8-layer LSTM is slower about 10 min per epoch. We tried to use teacher forcing to speed up the training process however it causes overfitting problems.

5.2 Problem B

Figure 14 is the training/validation loss (MSE) value over training steps exponentially decaying.

Here are visualizations of two training samples' ground truth and predictions. It's shown in figure 11. We noticed that the sample on the left has a better prediction result, while the sample on the right has a worse prediction result. The main reason is that the model has not been able to determine that the vehicle in the sample on the right has almost stopped moving.

Our final ranking on the leader board is **26** and our RMSE is **2.79800**.

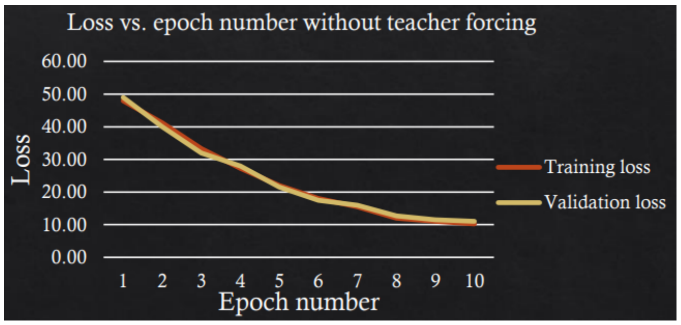


Figure 14: Loss vs. Epoch.PNG

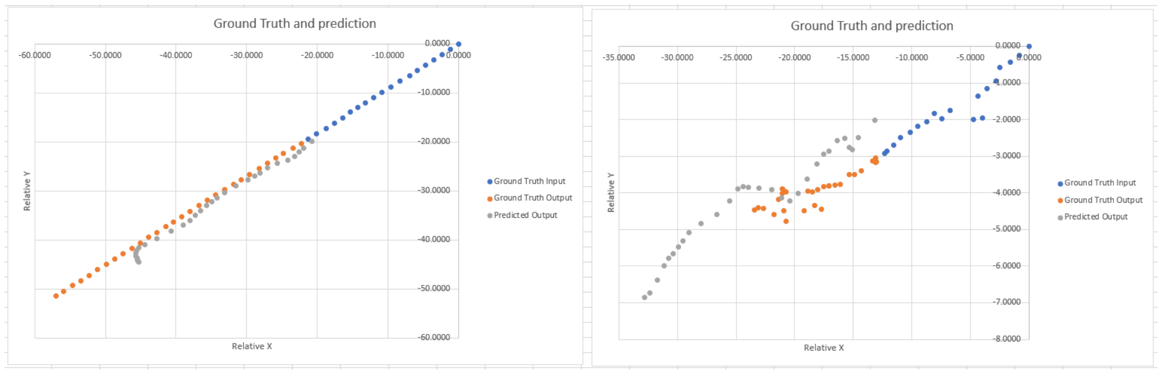


Figure 15: Ground Predict

6 Discussion and Future Work

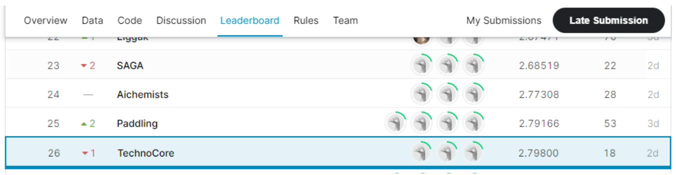
We think the most effective feature engineering strategy is to look deep into the data and identify where the features are most useful, also figuring out where is the data most vulnerable to a bad model is also very helpful

Data visualization definitely helped us understand the nature of the data provided and made our pre-processing efforts more effective. Also a good model is very crucial to improving our score, by reading other researches we found LSTM to be most useful. Finally hyper-parameter tuning is equally important for a good score.

Our biggest bottleneck is finding more effective models and finding good hyper-parameters.

We would advise a deep learning beginner in terms of designing deep learning models to start with simple models and build step by step, also be careful of overfitting problems. Additionally feature engineering and pre-processing is very important and should be done in advance.

We would like to explore some more sophisticated models and understanding how they work if we have extra time and resource.



23	▼2	SAGA	2.68519	22	2d
24	—	Aichemists	2.77308	28	2d
25	▲2	Paddling	2.79166	53	3d
26	▼1	TechnoCore	2.79800	18	2d

Figure 16: Final Kaggle Rank

References

[1] Curow. “Curow/Vehicle-Trajectory-Prediction.” GitHub, `github.com/curow/vehicle-trajectory-prediction`.

[2] “Getting Started with JAX (MLPs, CNNs amp; RNNs).” Rob’s Homepage, 16 Mar. 2020, `roberttlange.github.io/posts/2020/03/blog-post-10/`.

[3] Jagjeet-Singh. “Jagjeet-Singh/Argoverse-Forecasting.” GitHub, github.com/jagjeet-singh/argoverse-forecasting.

[4] “NLP From Scratch: Translation with a Sequence to Sequence Network and Attention¶.” NLP From Scratch: Translation with a Sequence to Sequence Network and Attention - PyTorch Tutorials 1.8.1+cu102 Documentation, `pytorch.org/tutorials/intermediate/seq2seq`$_{t}ranslation_{t}utorial.html$.

[5] Song, H. et al. *Learning to Predict Vehicle Trajectories with Model-based Planning*. ArXiv.abs/2103.04027 (2021): n. pag.

[6] *Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher B. Choy, Philip H. S. Torr, Manmohan Chandraker;*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 336-345