CSE 151B Project Final Report

Futian Zhang, Manxin Zhang, Ruojia Tao
f6zhang@ucsd.edu, maz029@ucsd.edu,rut013@ucsd.edu
Group name: Two Three Three Three
Github Link: https://github.com/f6zhang/CSE-151B-Final-Project

1 Task Description and Background

1.1 Problem A: Task Description

While autonomous vehicles are still in the stage of developing, it will likely change the way we commute in the future. The goal of autonomous vehicles is to be part of the transportation system and to be able to drive safely along with self-driving vehicles. Thus, the first step to achieve this goal is to predict the motion of self-driving vehicle. Since this involves a complex situation such as the turning, changing lane, and so on, the behavior of the self-driving vehicle also depends on the vehicles near by. For example, if the vehicle on the left wants to change lane, the driver may slow down or speed up to make room for the other vehicle. In our project, we utilizes different methods in deep learning to help predict the future movement of the target vehicle.

The task is to predict the movement of self-driving vehicles 3 second in the future, given data about the tracking object's velocity and position in the first two second. The observation includes velocity and position, and some information of the lane. The data-set comes from two different places: Pittsburgh and Miami.

If we can solve this task and make a good prediction of a self-driving cars, we can set up a system to detect other cars' behaviors and provide suggestion for the drivers by setting up a auxiliary system. For example, if the car is going to hit other cars, we can alert the driver to prevent this kind of accident happens. The auxiliary system can help increase drivers' security.

1.2 Problem B: Research

We did some research on this project and the one inspire us the most is the paper from *Argoverse: 3D Tracking and Forecasting with Rich Maps* (1). This is the official dataset description. They provide us some insights about how we should process the data, such as consider the object that near our target most. They also purpose a few baseline models, such as LSTM.

In addition to the dataset paper, we also read other papers that attempt to solve the problem with neural network. In the *Non-local Social Pooling for Vehicle Trajectory Prediction*(2), they introduce a encoder-decoder LSTM model with non-local social pooling. The non-local social pooling is between encoder and decoder to process the interactions between vehicles that are close to each other. This model considers both spatial and temporal information.

This article, *Modeling Vehicle Interactions via Modified LSTM Models for Trajectory Prediction* (3), solves the problem with a different LSTM model. They use three LSTM. The first is for predict the future trajectory, the second is for the interaction between target vehicle and another surrounding vehicle, and the third is to combine all the surrounding information to make final prediction. They make the unvarnished model by add a shortcut connection between the layers, which inspire us on how to make a model to solve vanishing gradient problem.

Preprint. Under review.

1.3 Problem C: Input and Output

In this prediction, the input includes the velocities and positions of vehicles. The given training files works like dictionary with 11 keys:
city: The city where the scene exist
p_in: the position of vehicles in the first two seconds with 10 Hz, total 19 timestamps
v_in: the velocity of vehicles in the first two seconds
p_out: the vehicles position in the next three seconds with 10 Hz, total 30 timestamps, the information we should predict
v_out: the vehicles velocity in the next three seconds, the information we should predict
track_id: the id for each vehicle, can be used to found the vehicle
scene_idx: the id of the specific scene, can be used to found the scene
agent_id: the track id for the target vehicle, which indicate the target we want to predict
car_mask: boolean value to indicate if it is a real car
lane: indicate the center-line nodes
lane_norm: the direction of each lane node

We are given the Training data $S = \{p\_in_i, v\_in_i, p\_out_i, v\_out_i, track\_id_i, track\_id_i, scene\_idx_i, agent\_id_i, car\_mask_i, lane_i, lane\_norm_i\}_{i=1}^{N}$, and we set up our model as f(x|w,b), while x represents $\{p\_in, v\_in, track\_id, track\_id, scene\_idx, agent\_id,\ car\_mask, lane, lane\_norm\}$ and y is $\{p\_out, v\_out\}$. However, our actual input is different from the given training data, we did a lot of modification and feature engineering on the given data set, to make sure the data used to train the model is concise and informative. We want to minimized the lost function L(y,f(x|w,b)) as our predictive goal: $argmin_{w,j} \sum_{i=1}^{N} L(y_i, f(x_i|w, b))$

I think the model can also be used to predict the different stage of the driving, for example, will the driver driving straightly? Will the driver making a U turn? Will the driver is accelerating? Will the driver changing the lanes? When we predict the next 3 second movement of the driver, we can also predict the meaning of the movement. We now only predict the motivation of the car, we can also use the direction and speed accelerate of our output to indicted the situation of the car.

2 Exploratory Data analysis

2.1 Problem A: Exploratory analysis

The given train data has a size of 205942. Each data sample includes the information of a road scene. The test data has a size of 3200. Our output includes the 60 columns of dimensions 30 (seconds)* 2 (directions [x,y]) of the target vehicle.

The dimensions and meaning of the inputs in each file:
city: 1, one city name
p_in: 60*19*2, the positions of 60 vehicles in the first two seconds with 10 Hz, total 19 timestamps
v_in: 60*19*2, the velocity of 60 vehicles in the first two seconds with 10 Hz, total 19 timestamps
track_id: 60*30*1, the id for each vehicle for 60 vehicles, there will be 30 values but all of them are the same
scene_idx: 1, the id of the specific scene
agent_id: 1, the track id for the target vehicle
car_mask: 60*1, boolean value to indicate if it is a real car for 60 vehicles
lane: n*3 (the last one column is 0) position of each lane
lane_norm: n*3 (the last one column is 0), direction of each lane
(n represents number of lanes in the scene)

The dimensions of the outputs in each file:
p_out: 60*30*2 the position of 60 vehicles in the next two seconds with total 30 timestamps
v_out: 60*30*2 the velocity of 60 vehicles in the next two seconds with total 30 timestamps

Here is a visualization of data sample 200001 (Figure 1):

According to the graph, we see that it is a representation of road scene. This scene includes 711 roads, which is the red lines, and forms a highway. The scene also contains 7 vehicles. The graph

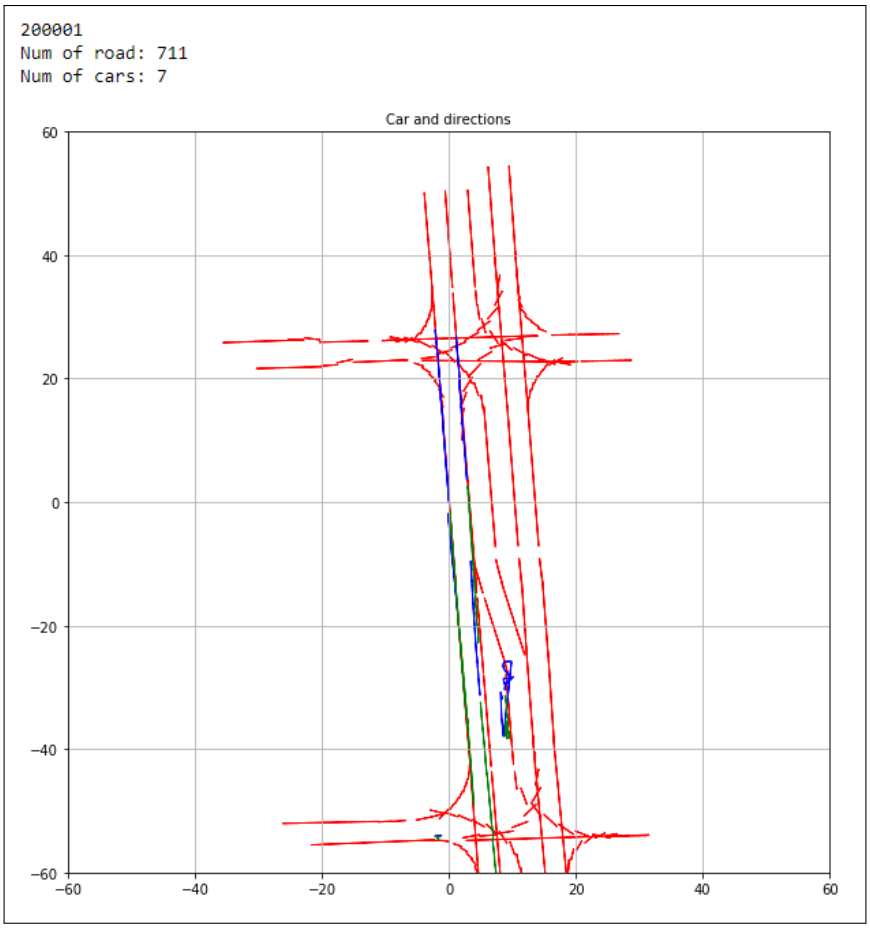


Figure 1: Distribution of Input Absolute Positions of All Agents

shows the movement of each vehicle in the road. The blue line is p_in and green line is p_out. The target vehicle, which is at the center of the graph, moves from top to the bottom. There are also other vehicles that moves in different velocity in this graph. There are also more information not shown by the graph, such as the city these scene belongs to, and id of the vehicles. The whole data sample build up a scene.

2.2 Problem B: Data Distribution

We have attached histograms of positions and velocities of all agents to the Appendix (Figure 13 - 23). We charted the input/output absolute positions/velocities of all agents and the target agents. We also graphed the input/output relative positions of the target agents with respect to their initial position. As we can see, the absolute position concentrates on two lanes. The relative position of the target vehicle and surrounding vehicles mostly on eight directions. Among the eight directions, top-left and bottom right occurs the least. Surrounding vehicles can be on left/right of the eight directions because the road contains more than one lane. But the area around the target vehicle has no car because drivers need to maintain safe distance.

2.3 Problem C: Data Processing

We do use some feature engineering. We modify the input and we take from the given scene into the closest six vehicles to the target car, because these six vehicles is most influential six cars towards the

target vehicles. The six vehicles are shown in Figure 2. We are helping the algorithm to pick the most influential variables in the scene.

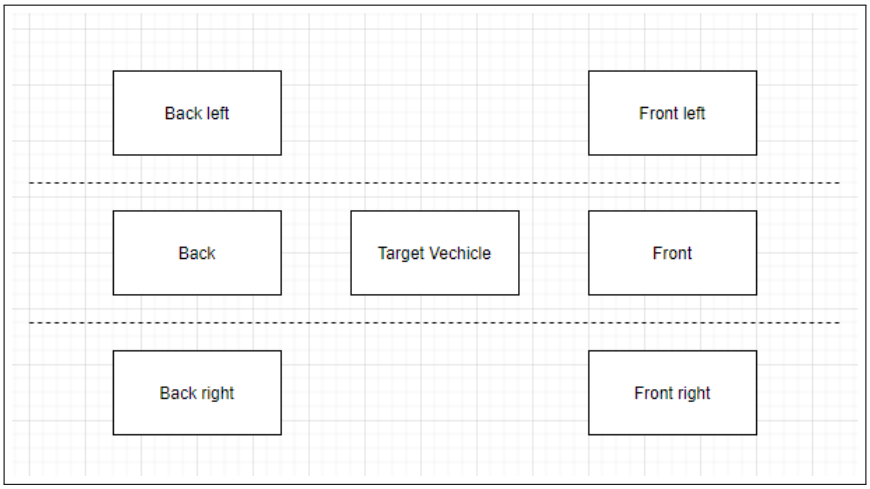


Figure 2: Figure of Surrounding vehicles

We also do the normalization of our data. For the p_in and v_in we perform some normalization on it. We normalized all positional values so that the last p_in of the target vehicle equals to (0,0). For the directions, we also normalized all values such that the last v_in of the target vehicle is at the direction of (1,0). The purpose we made these decision is because that we want to help our model to understand the position and direction. If we do this kind of normalization, our model do not need to consider the absolute values of position and the direction of velocity of the target vehicle. The only thing the model need to do is consider the change of position and the direction.

For the lane information, we use only the lane that is closest to the target car. That decision is based on our experiments. We experiment a lot of kinds of input and we finally founds out that only include the lane under the target car will provide us the best information. We have experimented with different methods of including more lanes, but every time, some scenes would include the wrong lanes.

3 Deep Learning Model

3.1 Problem A: Deep learning pipeline

We purpose two different kinds of deep learning models in this report. One is linear models and the other is a encoder-decoder model. We have different data processing for each kind of model.

Data processing for linear model:

For the linear model, we end up using 104 input data and 60 output data. The 104 input data composed of p_in and v_in of 19 timesteps of the target vehicle (76 values), p_in and v_in of last timestep of the six surrounding vehicle (24 values) and the lane and lane_norm that is closest to the target vehicle (4 values). The output data is p_out of 30 timestpes of the target vehicles concatenated together. We tone the input values so that this combination would yield the best result. The input data is not too much such that the model would be biased and the input data is not too less such that it doesn't provide enough information.

Data processing for Encoder-Decoder model:

For encoder-decoder model, we use 19 * 28 input data and 30 * 2 output data. The input data has 19 timesteps. For each timestep, the input data is composed of p_in and v_in of the target vehicle and 6 surrounding vehicles. The output data has 30 timesteps. For each timestep, the output data is composed of p_out of the target vehicle.

Loss function:

We use MSELoss function that is provided by PyTorch. We also tried L1Loss function and sometimes it works better because it reduces the influence of outliers. However, we decide to use MSELoss because it is more consistent than L1Loss.

Our decisions:

We choose to use linear model because it is easy to implement, so we want to start with a linear model. Also, linear model is flexible in its input and output. We can use any combinations of input values. Therefore, we can tone the best input data for the model.

We choose to use encoder-decoder model because it is good to handle future prediction and time sequence problem. We can easily process the input and output data to fit in the model. Since we are provided with 19 timesteps of input data and need to predict 30 timesteps of output data, we can use 19 sequences of LSTM as the encoder and 30 sequences LSTM as the decoder.

3.2 Problem B: Deep learning model

We will introduce 5 different models. 4 linear models and 1 encoder-decoder model.

1. Single layer linear model: Most simple model. Only use one fully connected linear layer from input to output. 104 inputs and 60 outputs. No activation and no normalization. Shown in figure below (Figure 3).

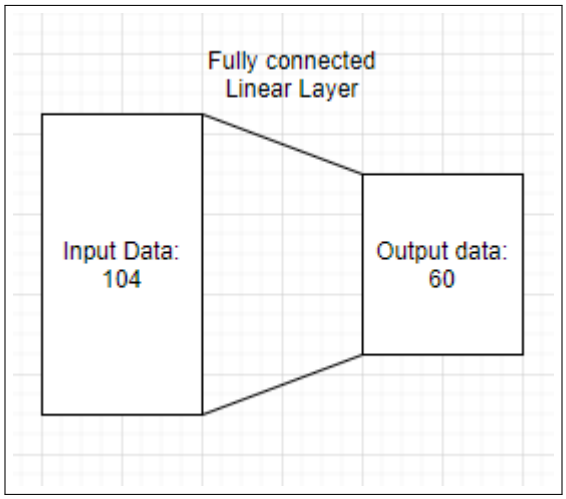


Figure 3: Figure of Single layer linear model

2. Multi-layer linear model: Multi layer model. 104 inputs first fully connected to a hidden layer with 1024 nodes, then in a loop for 10 times, 1024 nodes concatenate with a copy of 104 inputs, then a total of 1128 nodes fully connected with a layer with 1024 outputs. After the loop, 1128 nodes fully connected with 60 outputs. Sigmoid activation and layer normalization. Shown in figure below (Figure 4).

3. Multi-layer linear model with reuse layers: The same as above, 104 inputs are first fully connected to a hidden layer with 1024 nodes and 10 loops of 1128 nodes fully connected to 1024 nodes, and lastly connect to a layer of 60 outputs. But in the loop, we use the same layer for 10 times. This tests to be more efficient than the previous models. We think that it is due to the fact that the gradient would vanish while back-propagation, therefore, using the same layer would be beneficial. We use Sigmoid activation and layer normalization. Shown in figure below (Figure 5).

4. Repeat-layer linear model: This model has 12 layers of 2152 data fully connect to 2048 data. Similar to the above, we use 104 inputs data. For the first layer, we concatenate the 104 input data with 2048 zeros, together we have 2152 values. This layer is fully connected to a layer of 2048 nodes, then concatenate with 104 input data, and is fully connected to the next layer of 2048 nodes.

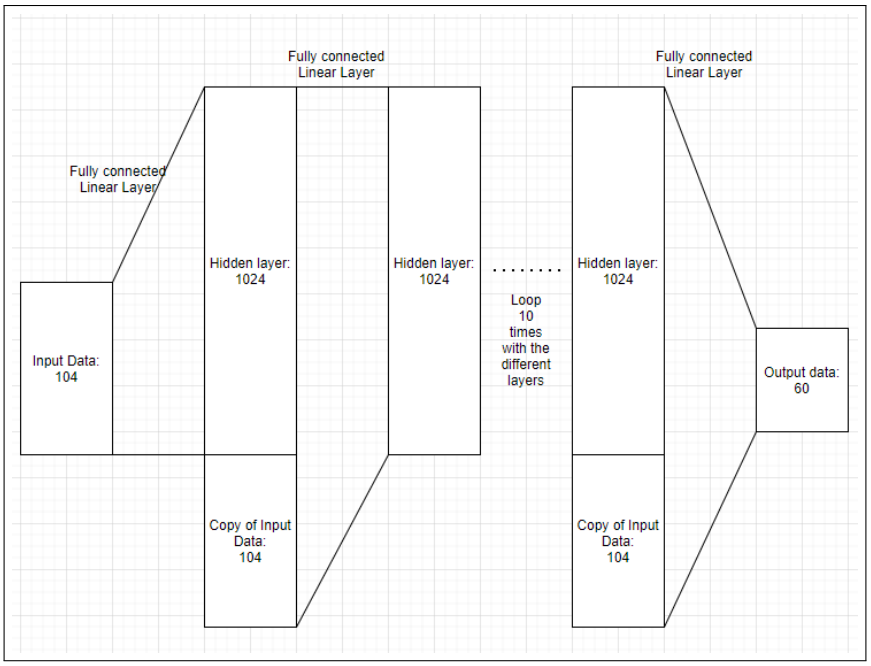


Figure 4: Figure of Multi layer linear model

We repeat this for 12 times, and the last layer is fully connected to 60 nodes, which is our output. This model is different from the above model because the 1st, 4th, 7th, 10th layer shares the same weight, the 2nd, 5th, 8th, and 11th layer shares the same weight, and the 3th, 6th, 9th, 12th layer shares the same weight. In other words, we repeatedly use 3 different linear layers to make a 12 layers linear model. We use ReLU for first and second repeated-used layers and Sigmoid for the third repeated-used layer. Because Sigmoid is tested to be the best for this problem. However, if we all use Sigmoid, we have the problem of vanishing gradient. Therefore, we only use Sigmoid for the third layer and use ReLU for the first and second layer. We use layer normalization. Shown in figure below (Figure 6).

5. Encoder-decoder model: This model uses the framework of encoder-decoder model. All LSTM has 2 layers and a hidden size of 100. For encoder, 19 timesteps of input data are used as the input of 19 sequences of encoder LSTM with initial hidden values of zeros. We only keep the last hidden value and it is used as the initial hidden values of decoder LSTM. For decoder, we use auto-regressive model, which means each LSTM uses the output of last LSTM as input. The first LSTM uses zeros as input because we normalize the last timestep of input position to (0,0). Its output is connected to a linear layer of 2 nodes, which would be the output of that timestep. At the end, we would generate 2 values of 30 timesteps. No activation and normalization layers. Shown in figure below (Figure 7).

For regularization techniques, we tried layer normalization, batch normalization and dropout. For linear model, layer normalization performs the best. Batch normalization yields a bad score and dropout would decrease the speed of training. For encoder-decoder, we don't use any normalization because normalization would remove some information that is contained in the layer.

4 Experiment Design

4.1 Problem A: Training and Testing design

We use the GPU provided by kaggle for all of the program which is NVIDIA TESLA P100 GPU, and the others are by the default setting. We also uses a NVIDIA rtx 1080 on a local machine.

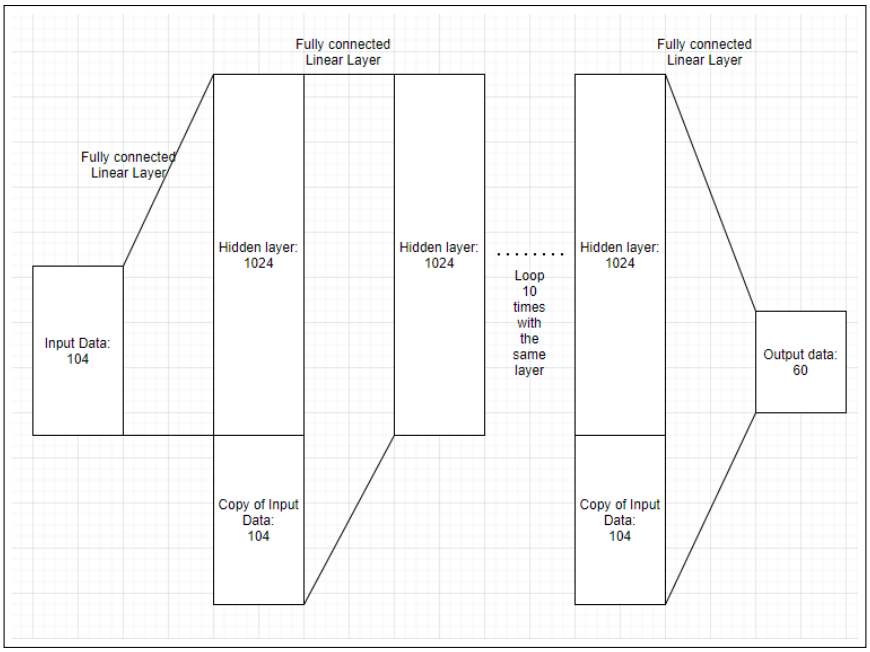


Figure 5: Figure of Multi layer linear model

We split the dataset into about 90-10 train-validation. We use 180000 data for training and 25942 data for validation.

We use Adam and our learning rate is 0.0005, and the other inputs are by default. We tune our optimization function by using the same dataset and running each optimization function and parameters to compare their results. For example, we tried SGD, Rprop, ASGD, Adam, RMSprop. It turns out that Adam is the most consistent optimization function that has the lowest test error while converges fast. SGD converges slow. RMSprop has low training error but high test error. ASGD and Rprop had a higher test error than Adam. We change the learning rate, learning rate decay, momentum and other parameters to see which pair of parameters would yield the best result. It turns out that Adam with learning rate 0.0005 and other default parameters yields the best validation error.

For the multi-step prediction, we simply output 60 data which is 30 pairs of (x,y) for linear model. For encoder-decoder, the decoder has 30 sequences, each sequence output one timestep of output data.

We don't use a fixed number of epochs. We would run our model for 50 epochs and record the model and the validation error after each epoch. Then we would use the model with the lowest validation error to generate the test output. We use a batch size of 250. For linear model, it takes 20 sec to train each epoch on Kaggle and 30 sec to train each epoch on our local machine. For encoder-decoder model, it takes 30 sec to train on Kaggle and 1 min to train on our local machine.

We use kaggle to train our model because it is more powerful than our computers. We also use rtx 1080 on our local machine because the Kaggle has a time limit to use their GPU. We choose to train over 180000 data and test over 25942 data because we think that more than 20000 data is enough for validation. We use Adam for optimization because we have ran tests and found that Adam is the most consistent optimization function that has the lowest test error while converges fast. We don't use a fixed number of epochs because we could overfit if we run too many epochs. Our method would get the model with the lowest validation error, which would also yield the lowest test error in theory. We use 250 batch because it is big enough to have accurate gradient estimate and run fast, while small enough to have low generalization error and we can have enough memory to train the model.

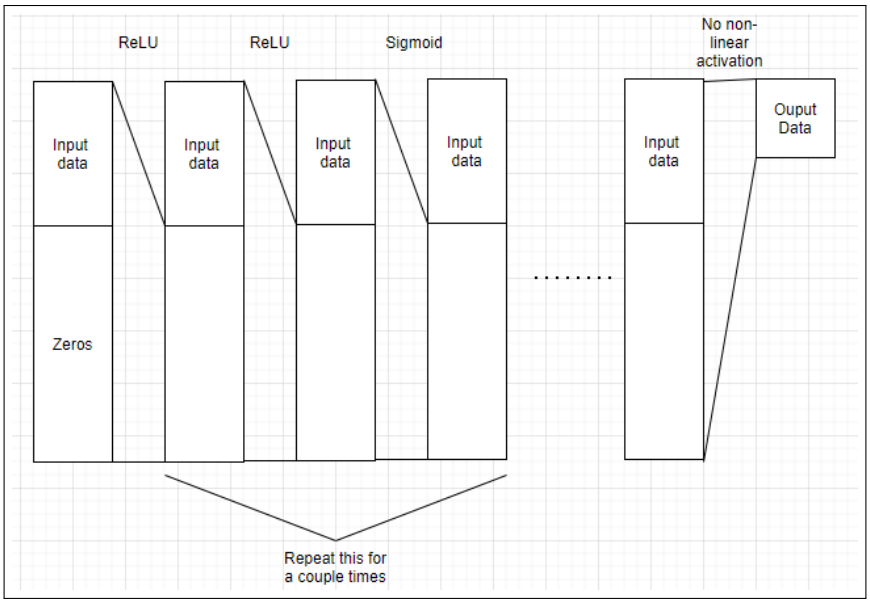


Figure 6: Figure of Multi layer linear model

5 Experiment Results

5.1 Problem A: Representative models

We will report our best linear model and the encoder-decoder model. The RMSE error of these models are shown in the below table.

Table 1: Test result for different model

Algorithm	Train Error	Validation Error	Test Error
Repeat-layer linear model	2.18476	2.28733	1.99828
Encoder-decoder model	2.23225	2.33789	2.22255

As we can see, repeat-layer linear model performs better than encoder-decoder model in validation and test error. Also, the test error seems to be lower than validation error, which means that the test dataset could contain less errors than the validation and training dataset.

For repeat-layer linear model, we roughly need 20 mins to train. For encoder-decoder model, we roughly need 40 mins to train. We reduce the training speed by increasing the batch size, use less hidden nodes, and use GPU instead of CPU to train.

For repeat-layer linear model, we have 13361308 parameters. For encoder-decoder model, we have 255402 parameters.

5.2 Problem B: Visualization of result

Training and testing RMSE over epochs: Shown in figure below (Figure 8). Note that first plot is training RMSE and the second plot is validation RMSE.

Random samples prediction: Red line indicates the road, blue line represents past position, green line represents ground truth, orange line represents our prediction We visualized some random samples to see if we do a good prediction in different traffics scene:

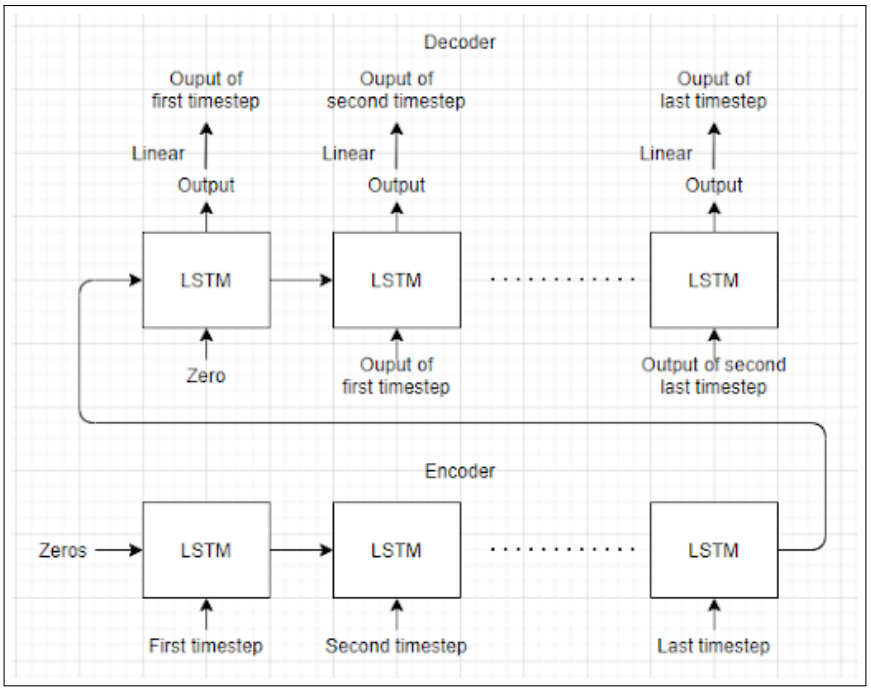


Figure 7: Figure of Multi layer linear model

1. Well predicted while driving in a relatively straight line: Shown in figure below (Figure 9). Our model works well in the scene that the vehicle drives in a relatively straight line.

2. Error while turning. Shown in figure below (Figure 10). Our model does not works well in predict the vehicle turning.

Current ranking: #1 on the leaderboard before the top 10 is locked. Test private RMSE: 2.04794. Test public RMSE: 1.99828.

6 Discussion and Future Work

Pre-processing the data:

One of the things we have learned through this competition is that pre-processing the input data is very important. We carefully select our inputs and normalize our data for our model to predict well. We think that the most efficient strategy is that we only include the information that we need. For example, we only include the last timestep of surrounding vehicles because if we include all timesteps, the model would become biased while training and not performing well. Also, we need to preprocess the data for our model so that our model don't have to understand the data first. For example, we normalize the position and direction so that our model don't have to understand the relative position and direction.

Strategies and techniques:

The techniques we found most helpful include data visualization and hyper-parameter tuning. Data visualization helps us understand the dataset better and identify possible errors in the dataset. As shown in Figure of Vehicle Trajectory (Figure 12), we are able to spot missing errors in the data from the inconsistency of the lines representing the positions of the vehicles. This helps us deciding our model and pre-process the data. Hyper-parameter tuning has also shown to be very useful in improving our scores. Trying out different models got us to a score as best as around 2.3 at the beginning, then with tuning, we were able to bring the score from 2.3 to 1.99 on the linear model as indicated in the Figure of Scores vs Submission (Figure 13). We tried different combinations of

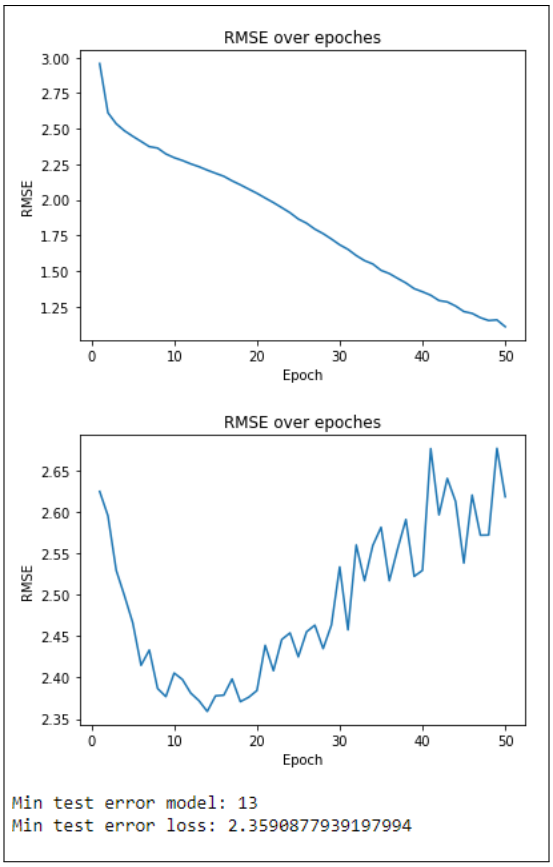


Figure 8: Figure of Training and Validation RMSE over epoch

model hyper-parameters, loss function parameters, and optimization function parameters. Those are all important for improving our score.

Bottlenect:

The biggest bottlenect is our model design. We considered many models before deciding on our final repeat-layer linear models. It takes us a lot of time to decide use a linear model and to design the structure of our model. Our score improves significantly after we finish designing the model and the rest is just toning hyper-parameters.

Tips:

For deep learning beginners, we would suggest trying out different models first to see which one(s) work better. A simpler model could work better on some problems. Then stick with the model, take a portion of the data size, and fine-tune the hyper-parameters to decrease errors. Record the scores after each epoch and see the trend to decide on the best numbers to use.

Exploration in the future:

If we have more resources, we would like to explore a lot of things.

Firts, we want to better pre-process our data. The real-world data is usually messier due to various reasons. While it makes sense to us that a data point of zeros might be an error among a set of non-zero data points representing the speed, the algorithm might not know that since it doesn't have prior knowledge like us humans do. Thus our future goal is to "clean up" so that the input data is as clean as it could get to feed into the algorithm.

Second, we would like to try out more models such as Graph, a better LSTM. There are many models for this problem in the public research papers. We could improve our score by using a different model.

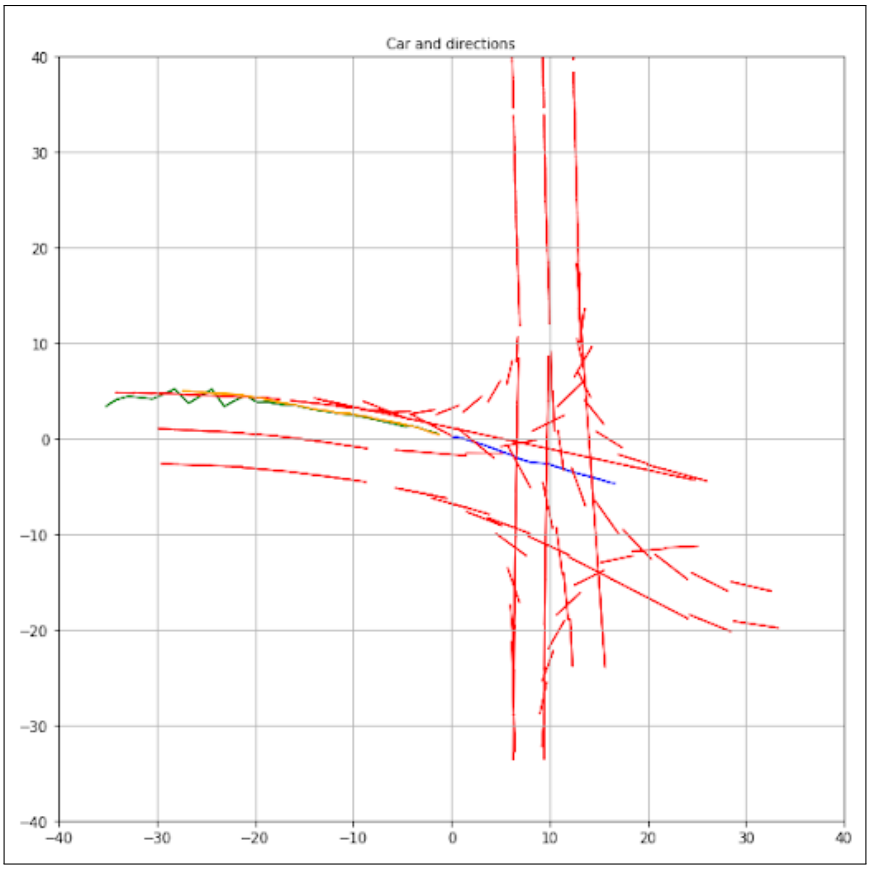


Figure 9: Figure of vehicles driving in a relatively straight line

Also, we did not use LSTM for our final model because LSTM would be disrupted by the errors that present in the dataset. If we can figure out a LSTM model that isn't influenced by the errors, we could improve our score.

Third, we would like to explore datasets from more cities. We have found that the two cities in the dataset are very different from each other when we trained them separately. We suspect the reason being that people in different cities having different driving habits due to the variation in weather, geography, culture, city layout, you name the rest. Therefore, we would like to take that into consideration when designing our model in the future.

References

[1] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Sławomir Bak, Andrew Hartnett, De Wang1, Peter Carr, Simon Lucey, Deva Ramanan, and James Hays *Argoverse: 3D Tracking and Forecasting with Rich Maps*. Argo AI, Carnegie Mellon University, Georgia Institute of Technology, 2019. `https://openaccess.thecvf.com/content_CVPR_2019/papers/Chang_Argoverse_3D_Tracking_and_Forecasting_With_Rich_Maps_CVPR_2019_paper.pdf`

[2] Messaoud, Kaouther and Yahiaoui, Itheri and Verroust, Anne and Nashashibi, Fawzi *Non-local Social Pooling for Vehicle Trajectory Prediction*. Intelligent Vehicles Symposium (IV), Paris, France, 2019. ff10.1109/IVS.2019.8813829ff. ffhal-02160409. `https://hal.inria.fr/hal-02160409/file/IV_Kamessao.pdf`

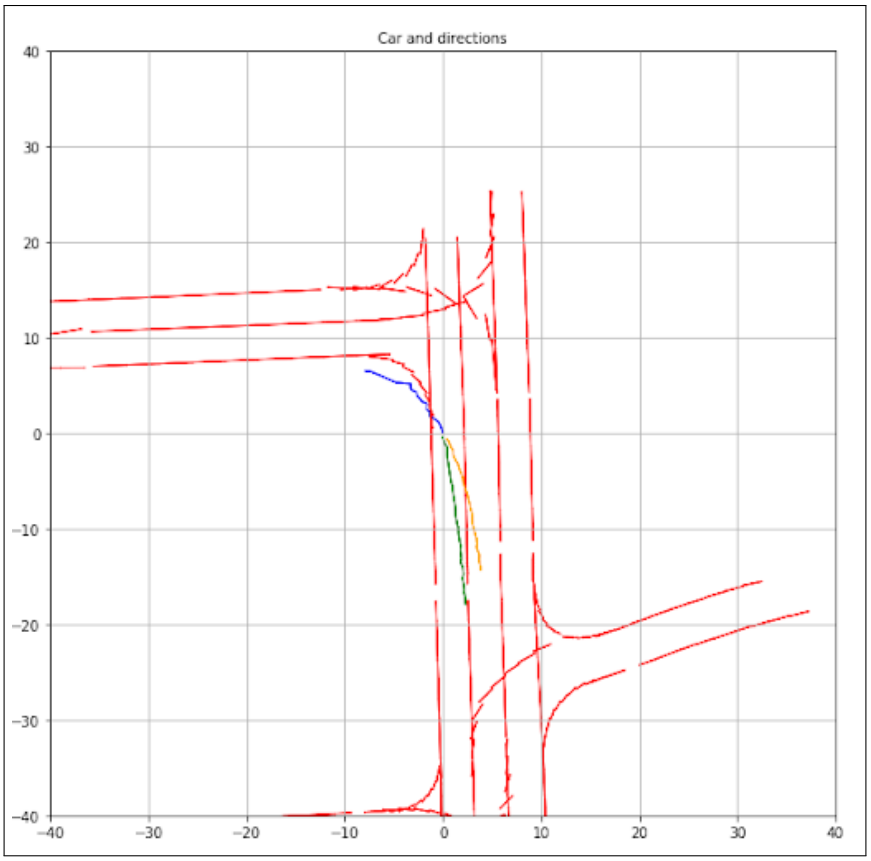


Figure 10: Figure of vehicles make a turn

[3] Dai, Shengzhe and Li, Li and Li, Zhiheng *Modeling Vehicle Interactions via Modified LSTM Models for Trajectory Prediction*. IEEE, vol. 7, 2019. doi: 10.1109/ACCESS.2019.2907000. `https://ieeexplore.ieee.org/document/8672889`

Appendix:

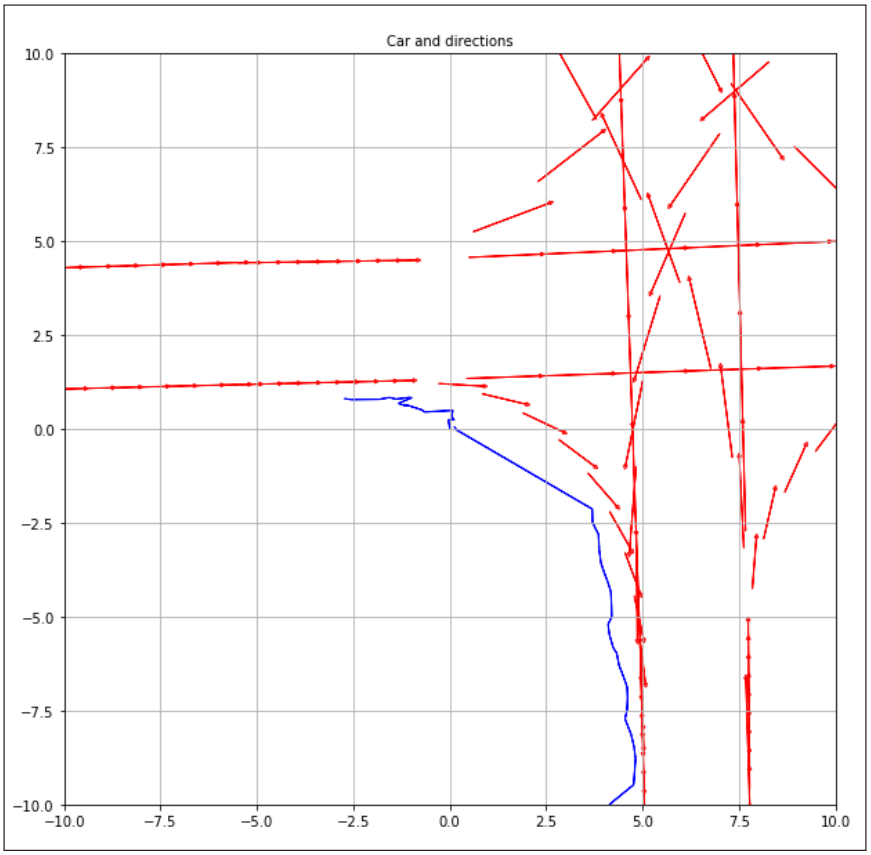


Figure 11: Figure of Vehicle Trajectory

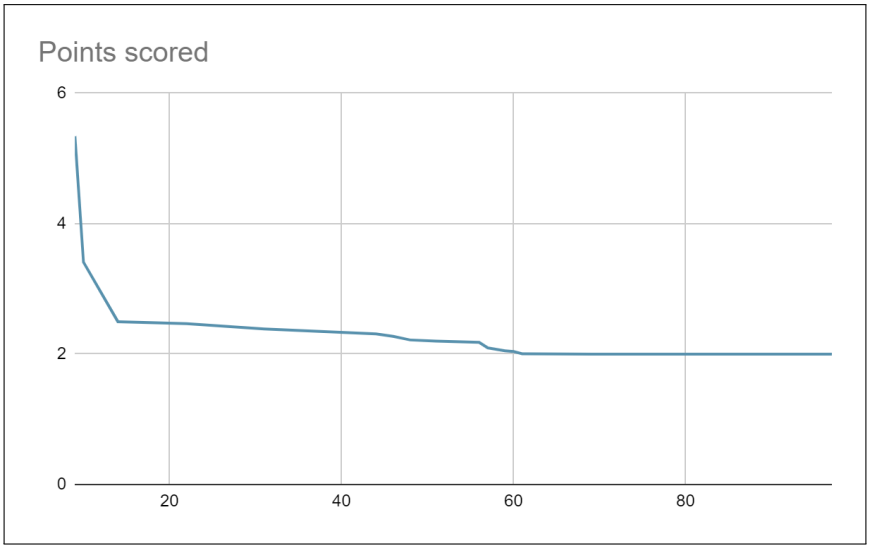


Figure 12: Figure of Scores vs Submissions

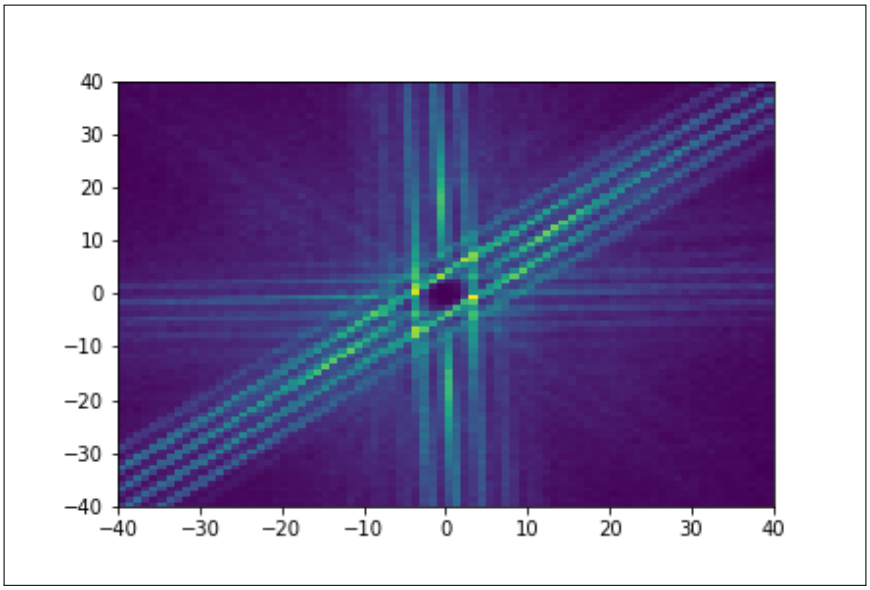


Figure 13: Distribution of Input Relative Positions of All Agents

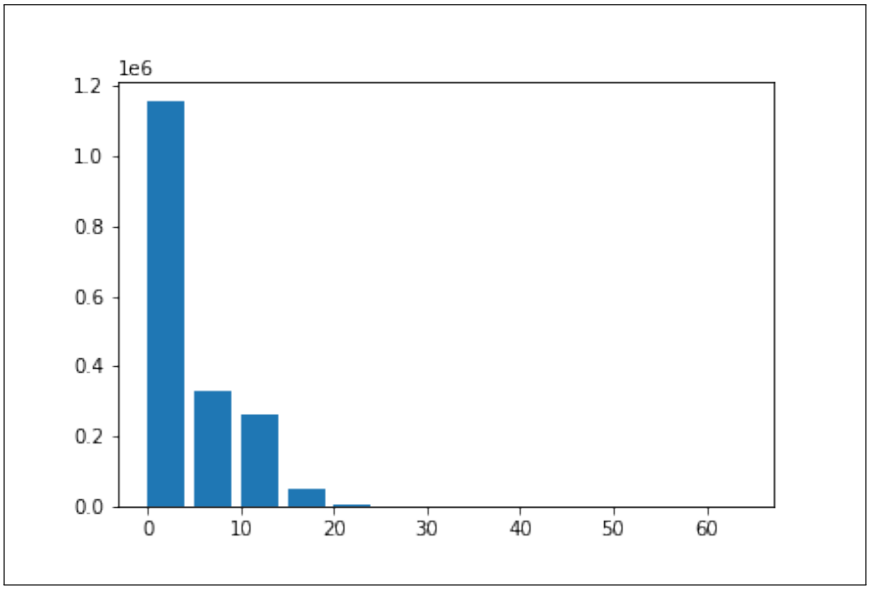


Figure 14: Distribution of Input Velocities of All Agents

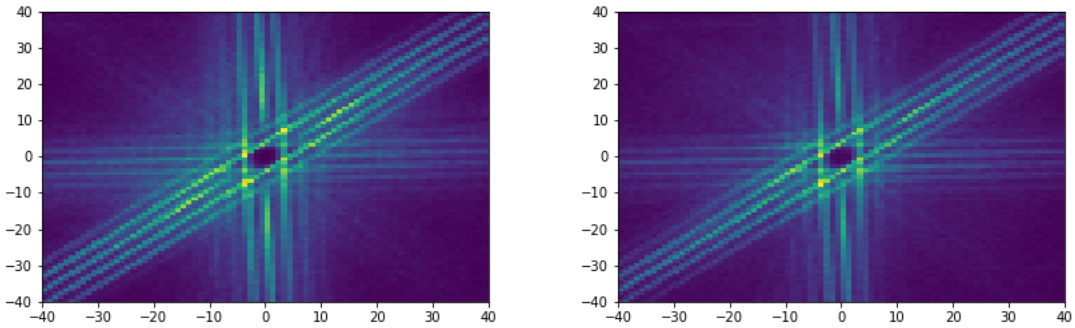


Figure 15: Distribution of Output Relative Positions of All Agents At Timestamp 0 and 29

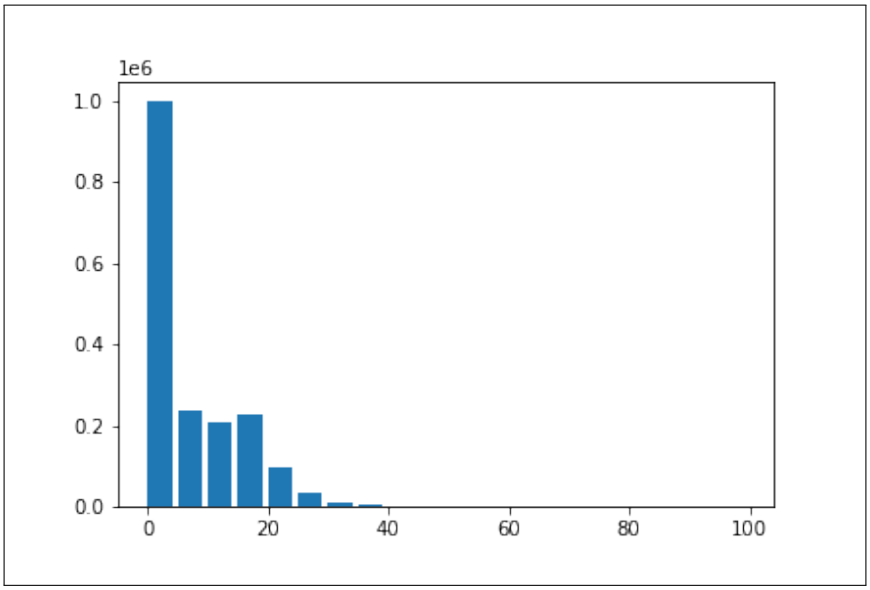


Figure 16: Distribution of Output Velocities of All Agents

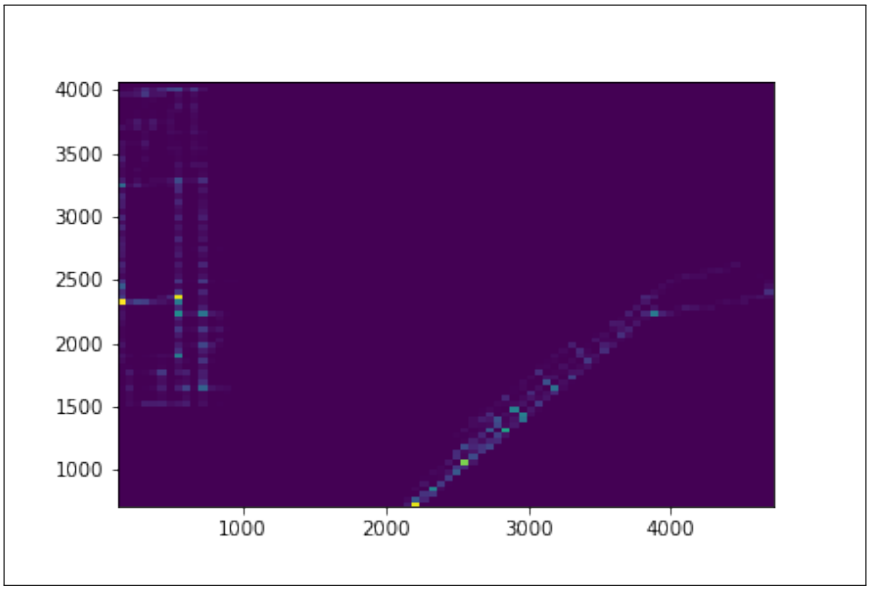


Figure 17: Distribution of Input Absolute Positions of Target Agents

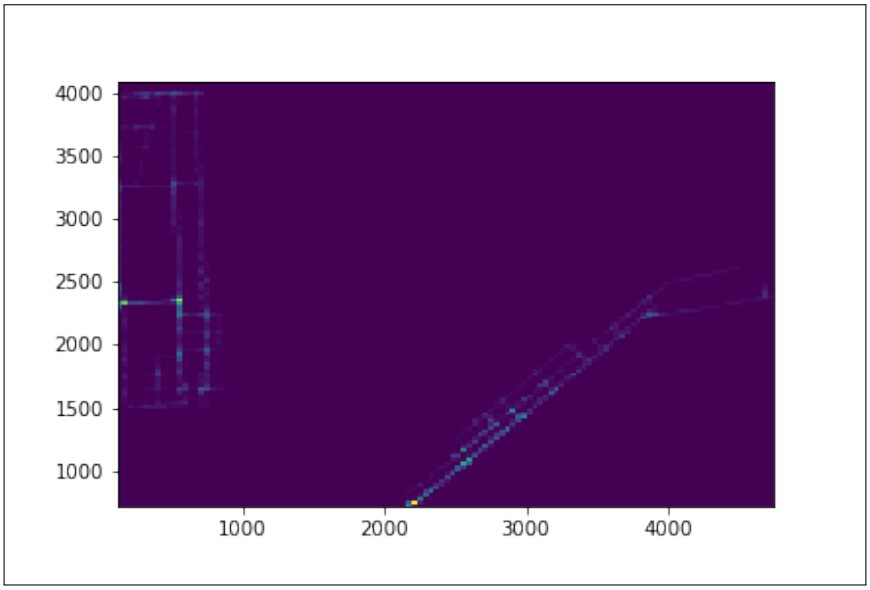


Figure 18: Distribution of Output Absolute Positions of Target Agent

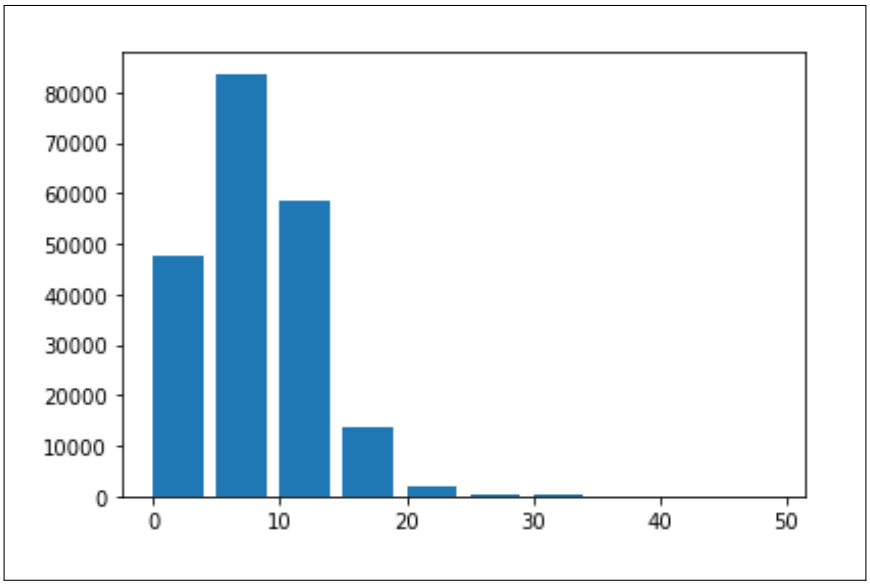


Figure 19: Distribution of Input Velocities of Target Agents

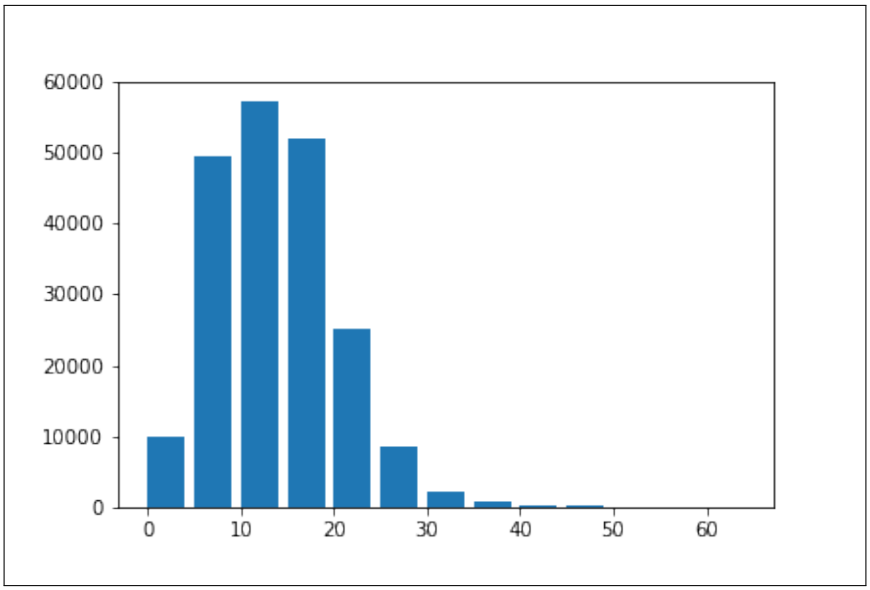


Figure 20: Distribution of Output Velocities of Target Agents

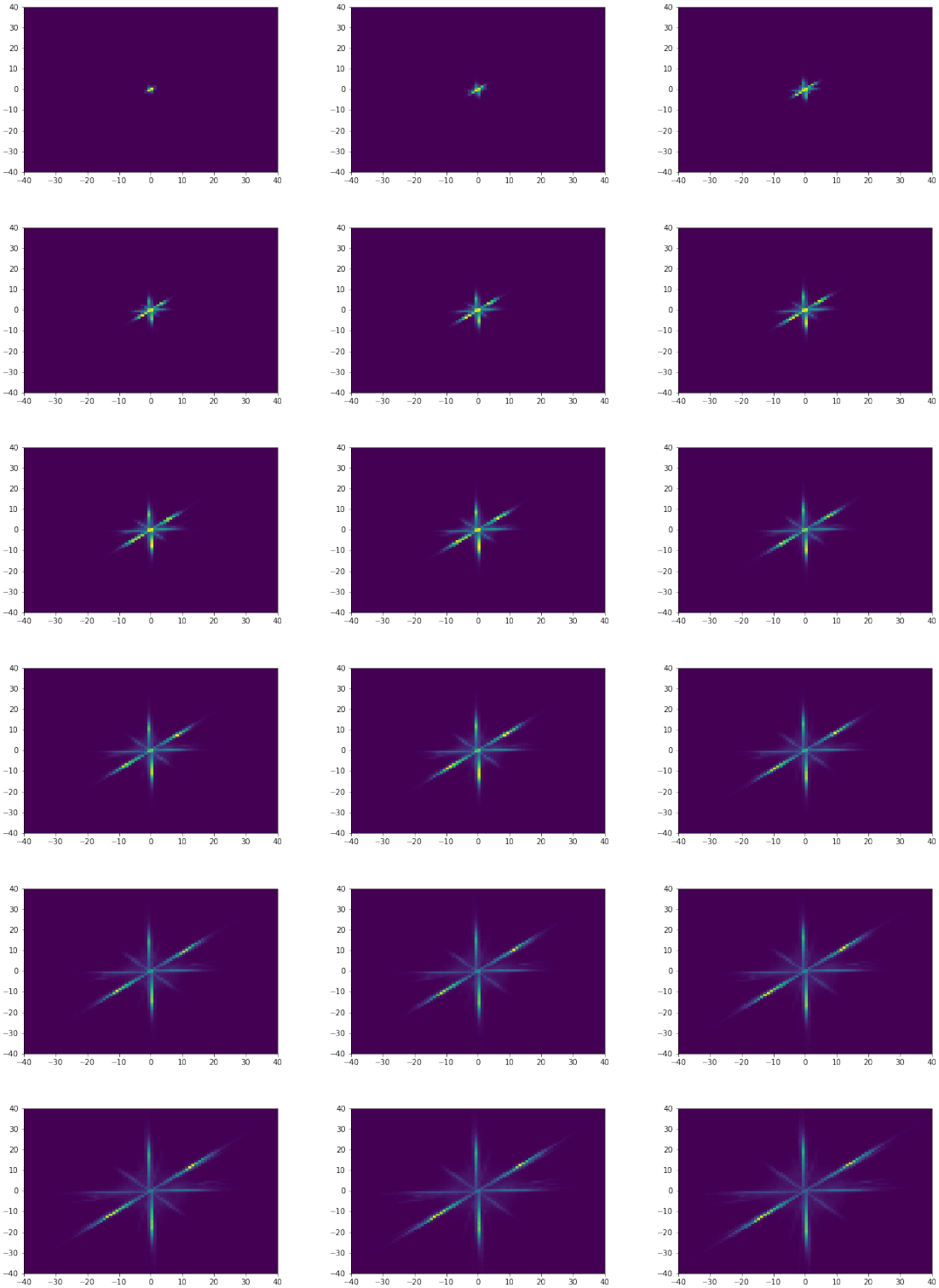
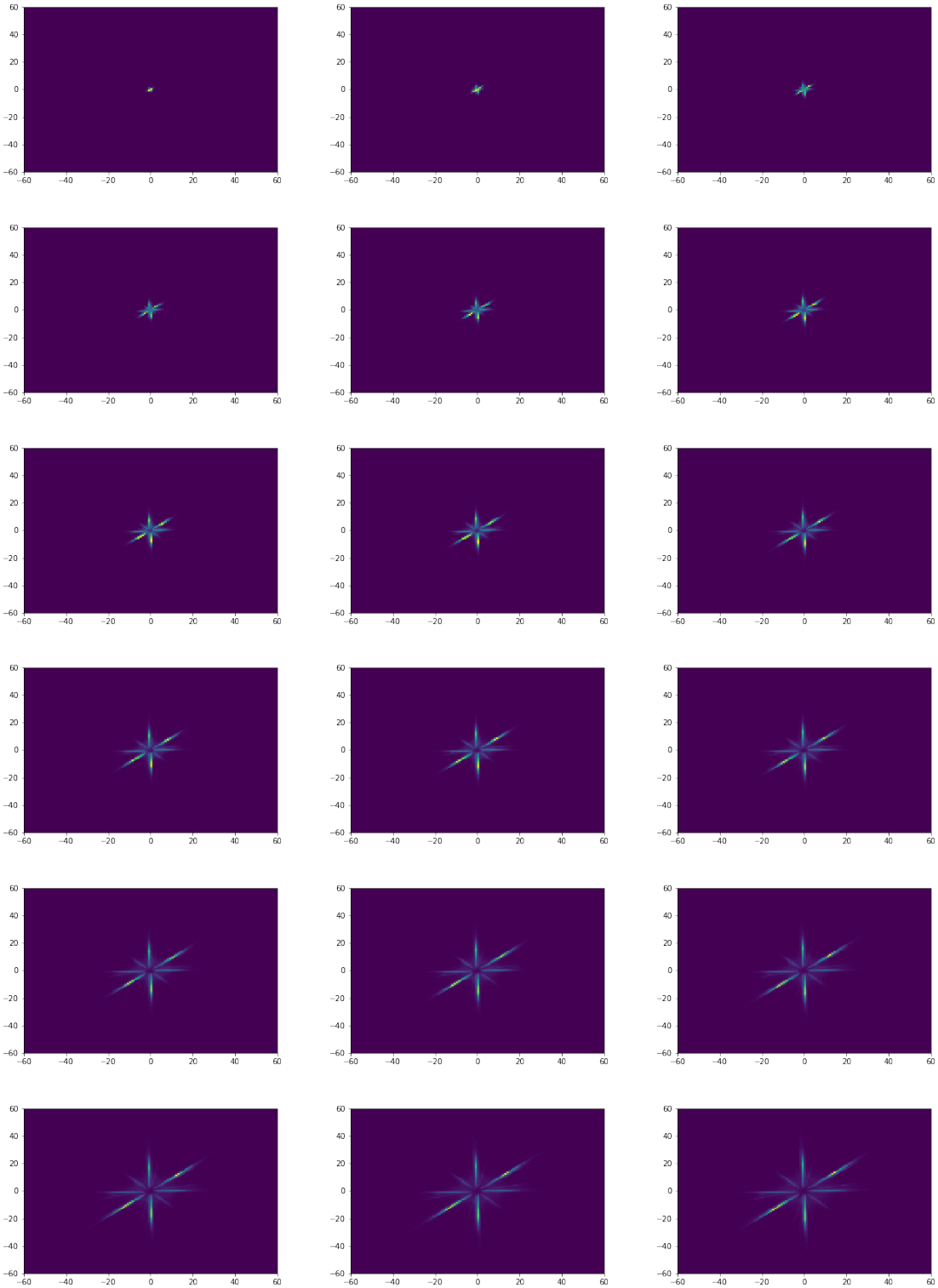


Figure 21: Distribution of Input Relative Position of Target Agent


60
40
20
0
−20
−40
−60
−60 −40 −20 0 20 40 60

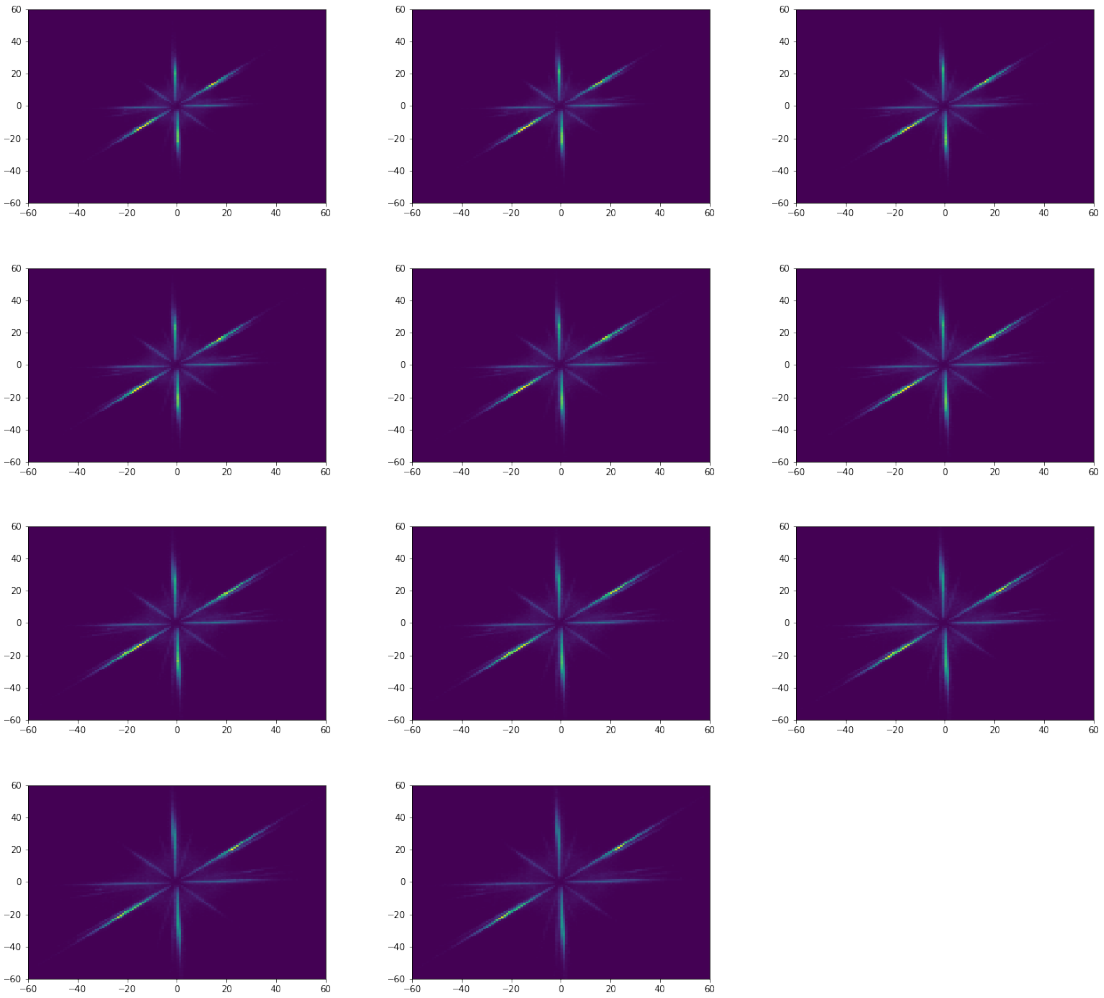


Figure 23: Distribution of Output Relative Position of Target Agent