Homework #6 Report

[EECN30168] Self-Driving Cars 2022

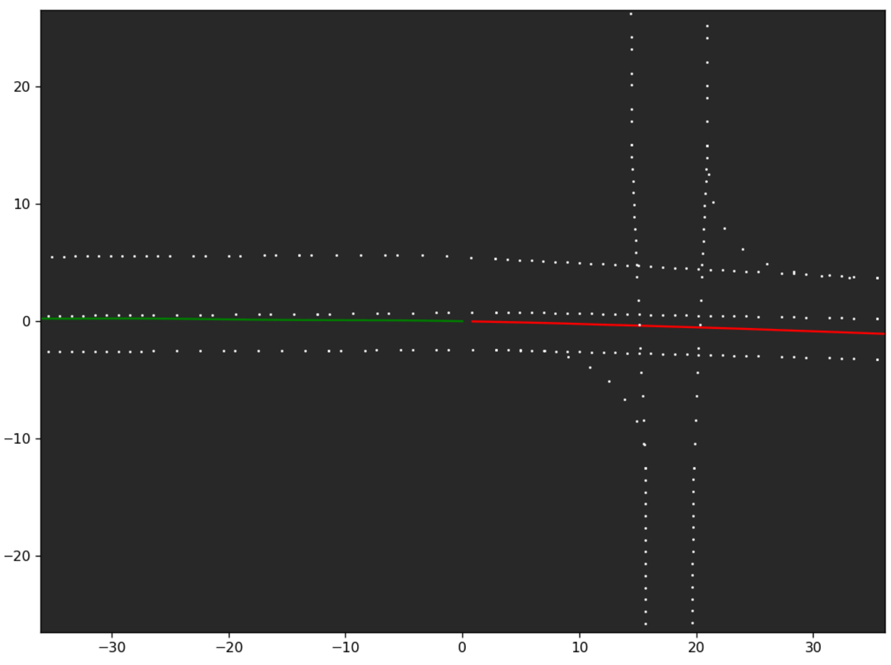
Student ID: 311605004 **Name:** 劉子齊 **Date: 2022.12.07**

1. Introduction

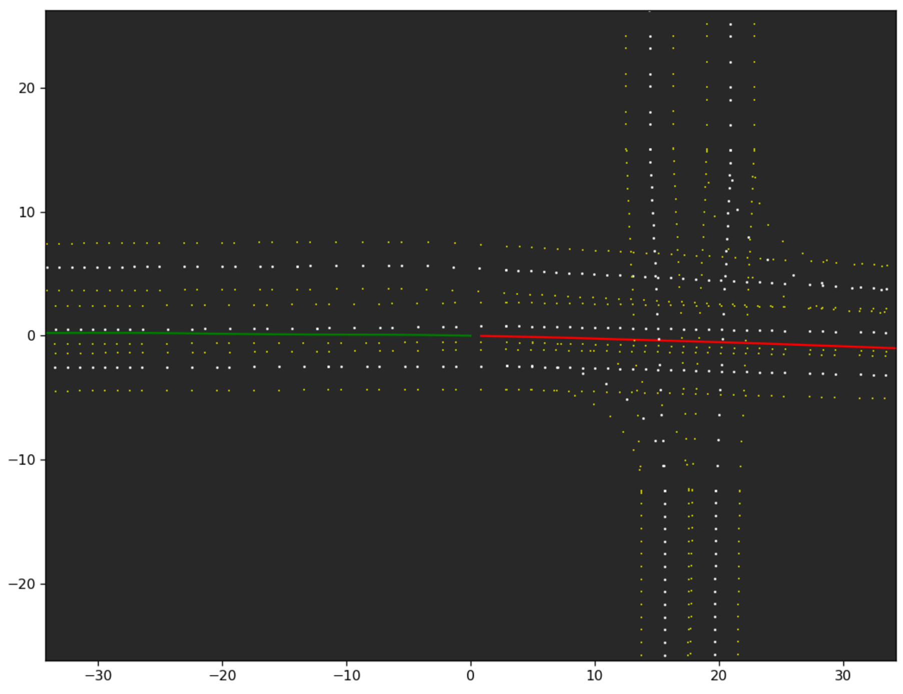
In this homework, there are two goals for us to accomplish. The first goal is to get familiar to the motion dataset and to visualize the historical trajectory and road line of the object to be predicted through Matplotlib. The second is the implementation of motion prediction model which we have to observe the history of the past five seconds and predict the trajectory for the next six seconds.

2. Visualization of the Dataset

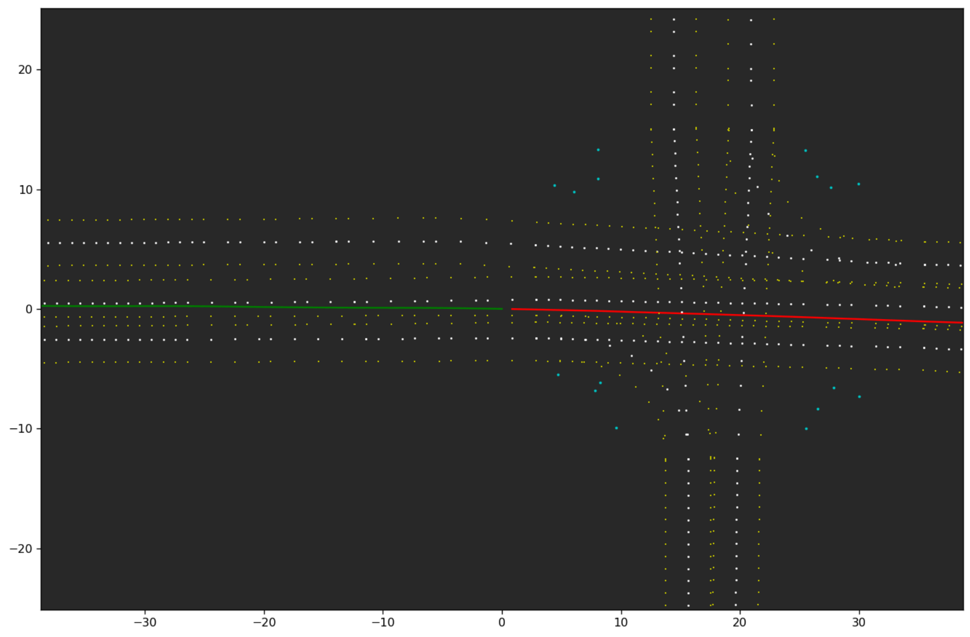
The following is the visualization result of the centerline of the road, the history trajectory, and the future trajectory. The red line in the figure is the future trajectory. The green line in the figure is the past trajectory. Finally, the white dots in the figure below form the lane centerlines.



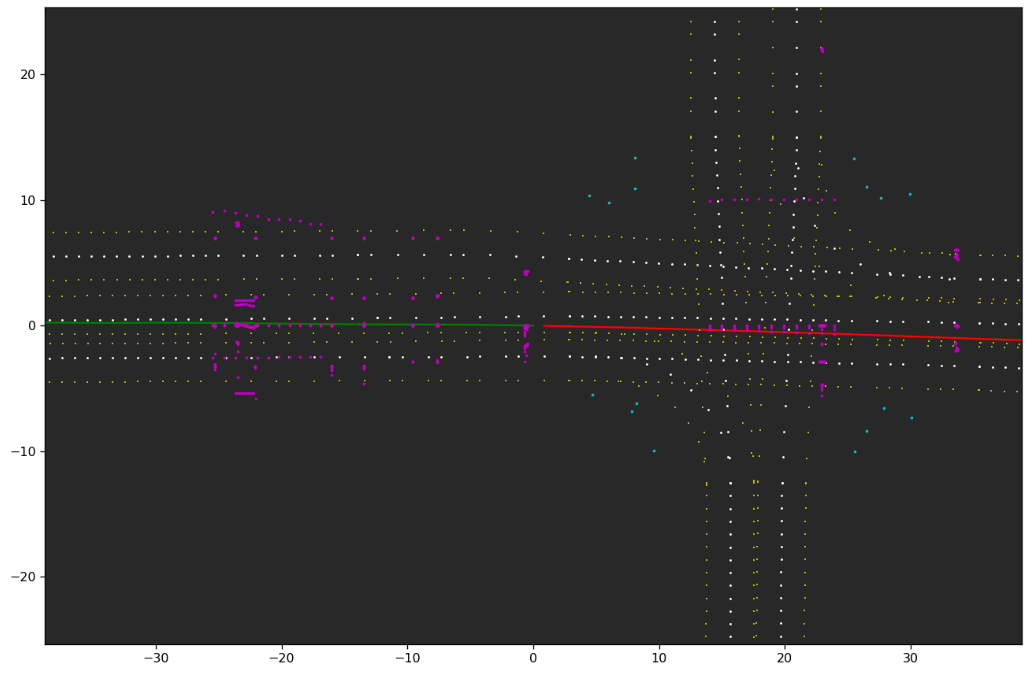
Following the previous figure, the following figure add on the visualization result of the lane polygon, which is consisted by the yellow points in the figure below.



With the lane polygon drawn, now we move onto the crosswalks, which is shown in the following figure by the cyan points. Here we can find the dataset uses the endpoints at both ends of the crosswalk to represent the entire crosswalk instead of drawing the entire crosswalk directly.



Finally, we move onto drawing the trajectory of the surrounding objects, which is shown in the figure below by the magenta points. In the figure below, we can find some surrounding objects detected on the lane that might be a car, and some objects detected on the crosswalk that might be some pedestrians.



3. Implementation of the Motion Prediction Model

After finishing the visualization of the dataset properly, now we move onto the implementation of the motion prediction model. As the process, we have to observe for 5 seconds and make use of our model to predict the trajectory for the next 6 seconds.

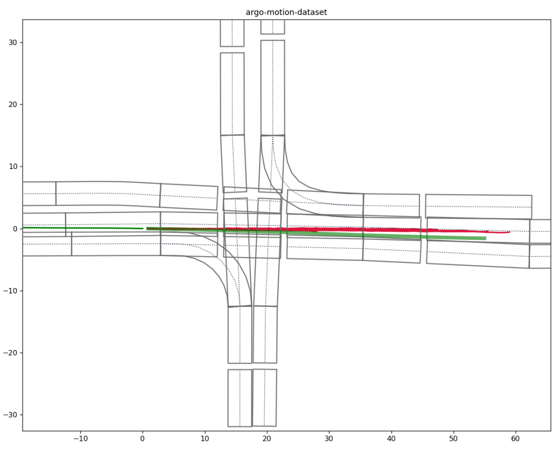
For the design of the prediction model, I basically based on the architecture provided. Initially, the size of the hidden layer was set to 128 * 128 initially. After testing the initial settings out, I consider the hidden layer could actually be bigger to contain more information. Hence, I changed the size of the hidden layer to 256 * 256, which is four times bigger than the original one, and I'll take it as my baseline in this project.

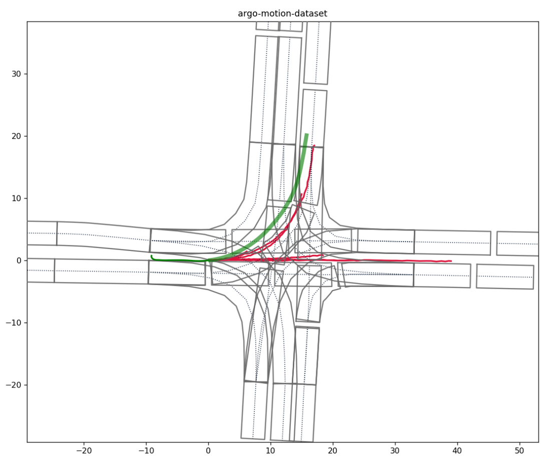
Afterwards, I started to try to make my docker able to train my model through the GPU, but not the CPU to make the training process to be less time consuming. However, I had some difficulties making use of my GPU through the docker provided, even if I followed several references on the internet, there's always errors popping out telling me I didn't set my docker correctly.

After spending plenty of time trying to figure out the GPU problem of the docker, I started to think about if the GPU is necessary in this project. I found that if I simply make use of my CPU, it will take me about an hour to train my model for 100 epochs, which is actually tolerable on the time it cost, and the result is not bad. As a result, the device I made use of in the experiments in this project are all CPU.

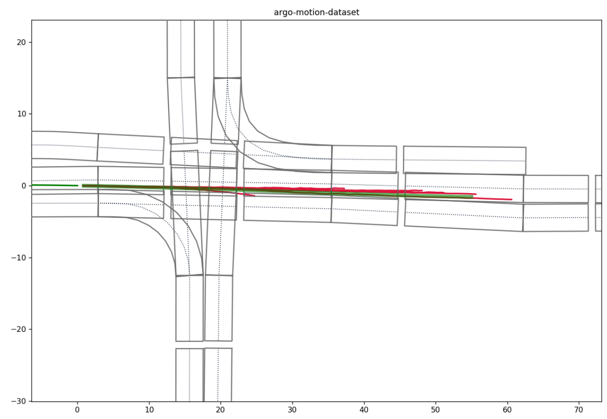
In this project, I mainly modified the calculation of the total loss, trying to find the correlation between the Average Displacement Error (ADE), the Final Displacement Error (FDE) and the performance of the model. Besides, I set the total epoch to 100, and learning rate to 0.0001 for all the experiments. The follow are the different calculations I applied on the total loss and their corresponding captures of their result:

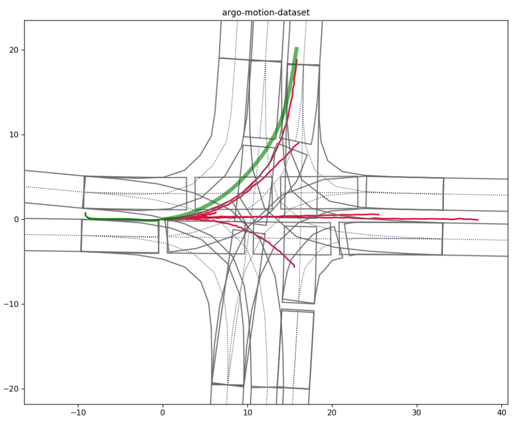
- Total Loss = ade + fde + CONF_ALPHA * maxmargin



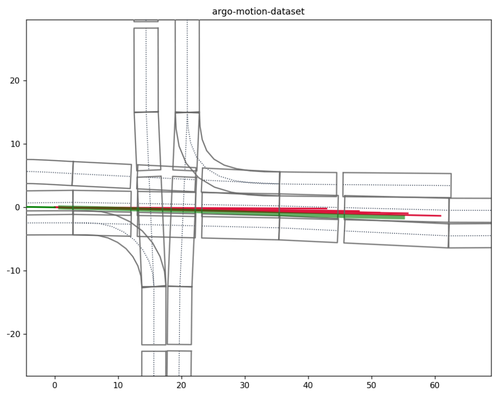


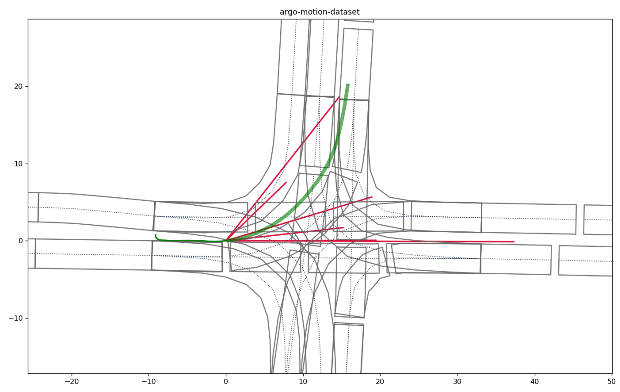
- Total Loss = ade



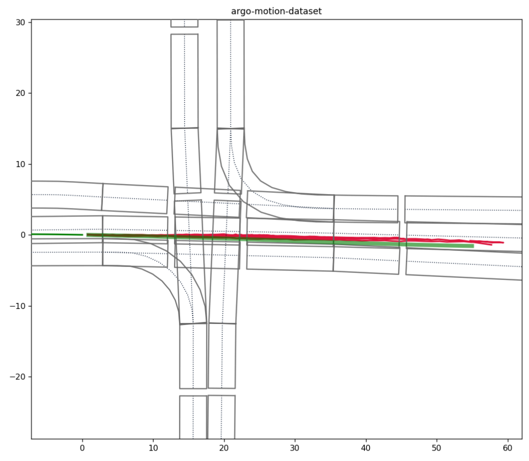


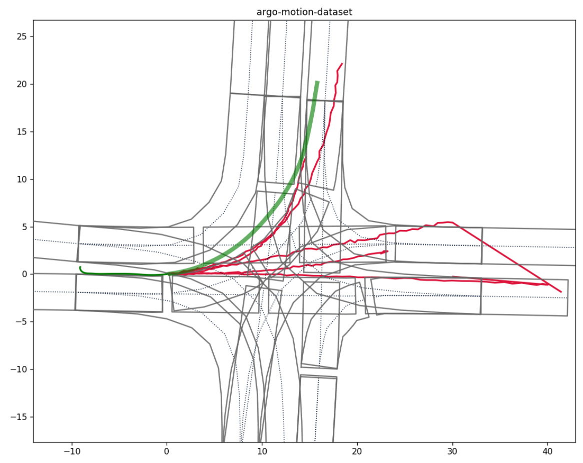
- Total Loss = fde



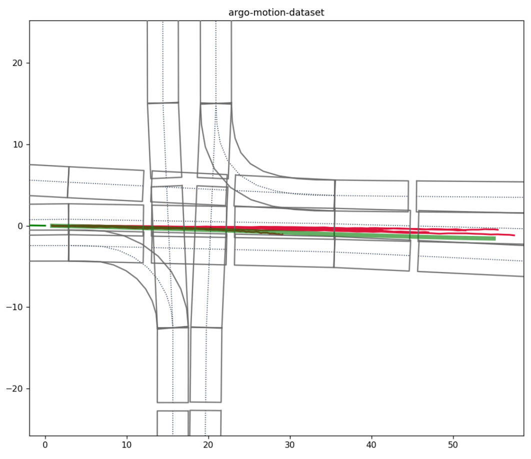


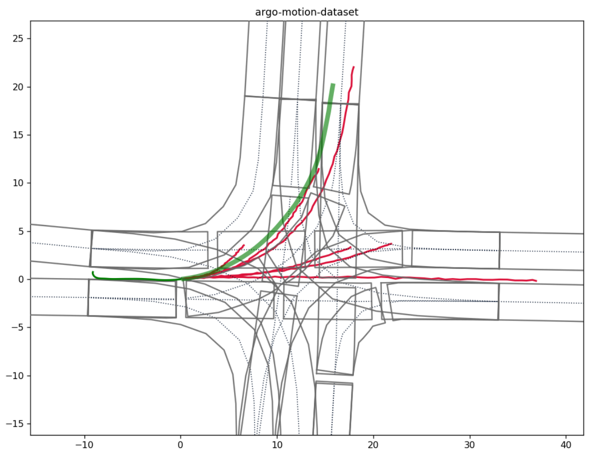
- Total Loss = ade + fde



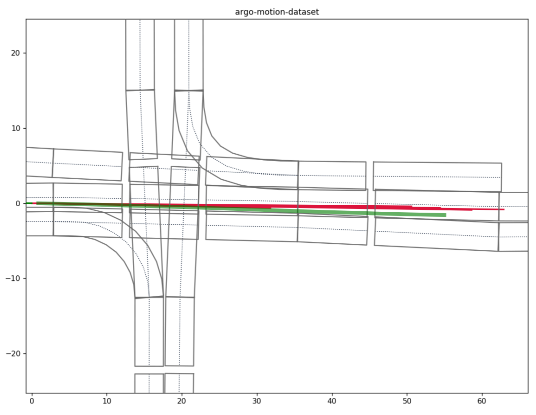


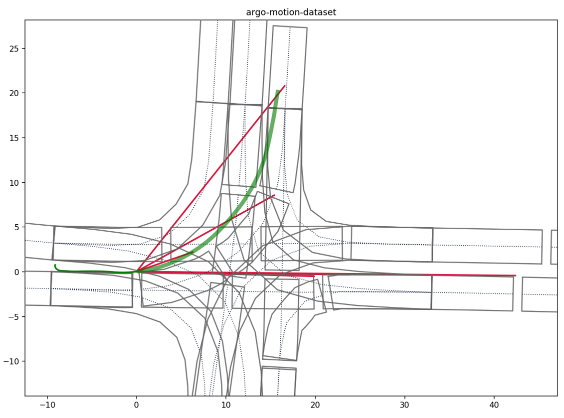
- Total Loss = ade + CONF_ALPHA ∗ maxmargin



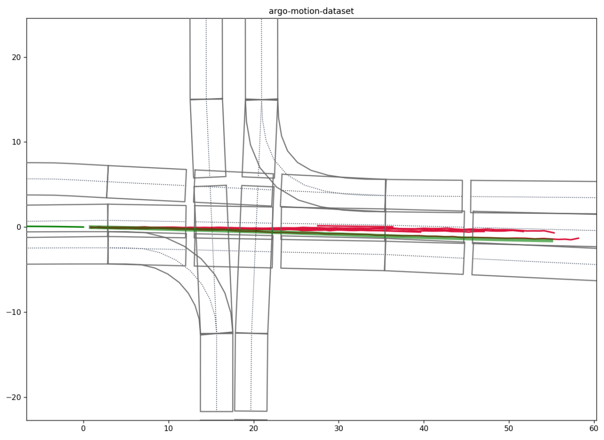


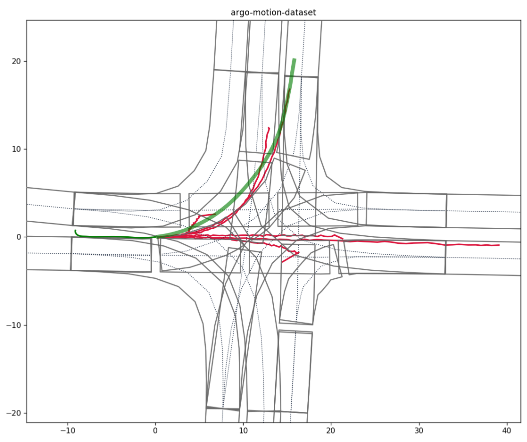
- Total Loss = fde + CONF_ALPHA ∗ maxmargin





- Total Loss = 0.5 ∗ ade + 0.5 ∗ fde + CONF_ALPHA ∗ maxmargin





Through the figures above, we can find that the baseline can already predict simple scenes, such as going straight and making turns. However, when we encountered sudden decelerate, our model will make some prediction errors.

I think this result can be attributed to the simple MLP architecture applied in our model. Such a simple architecture is not likely to have any major problems when dealing with some simple marching or turning. When dealing with some simple surrounding noise, it will not be affected to some extent. However, when the distance is extended, or the information on the map is reviewed more, some errors will occur, such as performing a turn where there is no fork in the road, and so on.

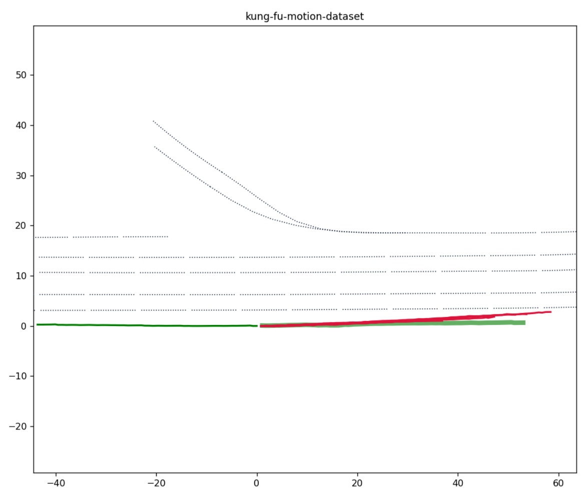
If we don't take max margin into account today, we can find that no matter how we add ADE or FDE, the final calculated trajectory is easy to be sprayed out. In the case of taking max margin into consideration, if only ADE is calculated, the final result will basically not be much different from the baseline.

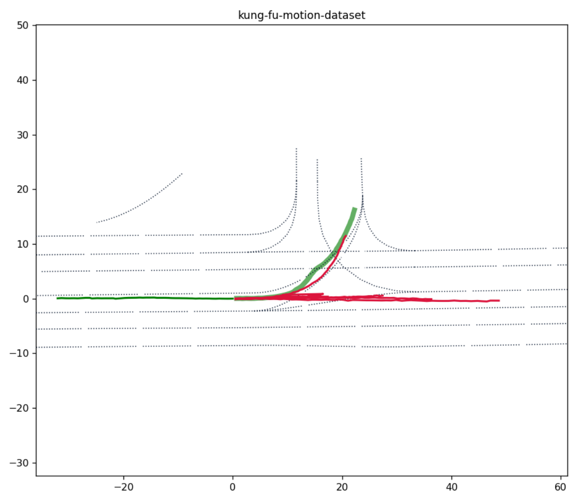
If only FDE is calculated, it can be found that almost all the predicted trajectories are straight lines, and there are almost no curves, but the final end point is almost in line with the end point of the car. I speculate that was attributed to our simple model, which was affected by a relatively small number of turning scenarios, causing it to want to take the majority move, that is, go straight.

As a result, I found that when max margin is taken into consideration, the path predicted by the trained model is the most reasonable and the best performance in the case of half of ADE and FDE, which Total Loss = 0.5 * ade + 0.5 * fde + CONF_ALPHA * maxmargin.

4. Prediction on the Kung Fu Dataset

The following figures are the prediction results of the best model I derived from the Argoverse 2 dataset when applied on the Kung Fu dataset.





From the results in the figures above, we can find that in the data of Guangfu Road, when we are going straight and decelerating, the motion can be reasonably predicted, at least the error value is still small. But if we encounter a turning situation, the predicted result will sprayed out first easily and then being caught back afterwards.

In my opinion, the reason for this situation can also be attributed to my over-simple model architecture, and it may also be due to the fact that the situations of making turn are a minority in the data set of Guangfu Road.