EPFL

Deep Learning for Autonomous Vehicules

Project Milestone 2

Victoria Nguyen, 385211
Valentin Perret, 327718
Code Git

April 2024

1 Introduction

The purpose of this report is to demonstrate all the modifications made to our basic model implemented in milestone 1. We have been able to experiment many different approaches: illustrations and code fragments are shown to illustrate our purpose.

2 Data augmentation

Initially, at the end of Milestone 1, we noticed an imbalanced dataset : an important part of the error comes from this imbalance and perhaps from the lack of robustness of our model for rare trajectories, or those that slightly differ from those seen in training. To this end, the first approach we implemented was data augmentation to improve the generality of the network and its robustness.

To do this, we've modified the BaseDataset class. We have defined our transformation classes, illustrated below. We then modified the getitem function, and applied the on-the-fly transformation for each epoch of the training. We chose to apply each transformation to the entire dataset. We tried differents types of transformations :

Horizontal Flip

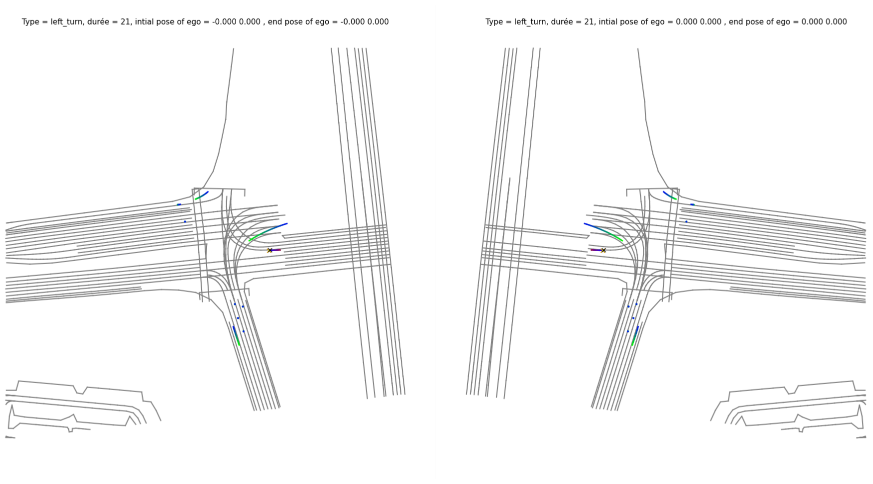
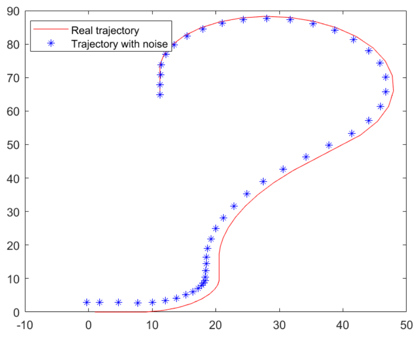
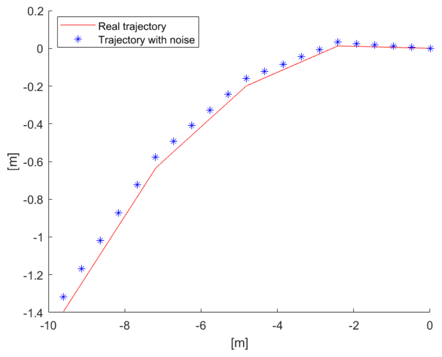

Type = left_turn, durée = 21, intial pose of ego = -0.000 0.000 , end pose of ego = -0.000 0.000
Type = left_turn, durée = 21, intial pose of ego = 0.000 0.000 , end pose of ego = 0.000 0.000

Figure 1: Horizontal flip BEFORE/AFTER

Constant noise on trajectory of vehicle
After implementing noise on the starting position, we implemented constant noise on the whole trajectory. We wanted to create a new trajectory of our own agent, beeing parallel to the real trajectory. For this we needed to make sure even in the curves that the noise would be correctly applied. The goal is to apply the same noise with respect to the agent's relative orientation/position. When the time steps increase this noise magnitude reduces to join the original trajectory. This is done to keep the final output the same as for the original trajectory. In Fig. 2, there are examples of how this is applied. We added a Gaussian noise of 10 cm on the initial position, thus getting reduced through every time steps. As a comparison this noise implementation is similar to having a right wheel's trajectory and then computing the left wheel's trajectory, but the left wheel starts to join the right's wheel position.



(a) Trajectory example



(b) vue de dos

Figure 2: Example of the noise applied on the trajectory, the left figure shows how it works on different type of curves, the right figure shows on example of the trajectory of an agent during training: after and before noise.

Random Lane Occlusion

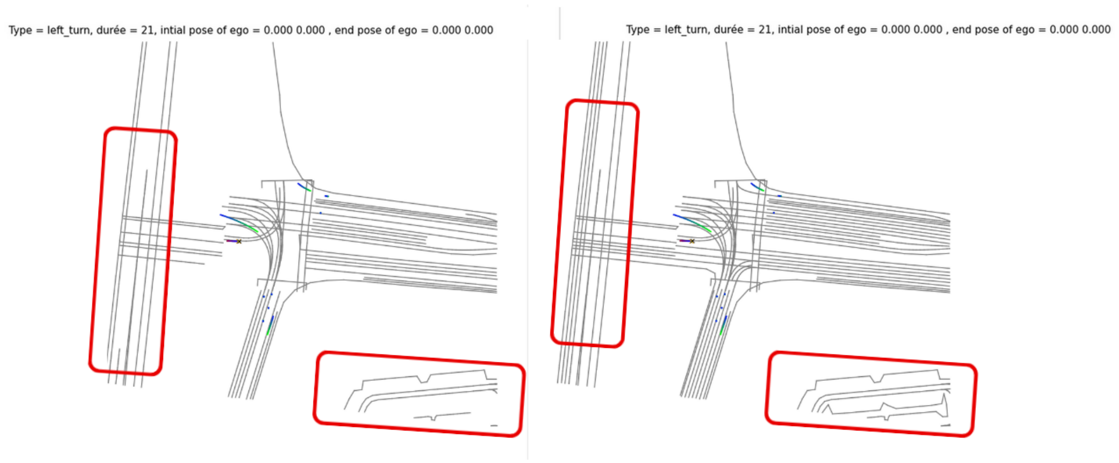

Type = left_turn, durée = 21, intial pose of ego = 0.000 0.000 , end pose of ego = 0.000 0.000
Type = left_turn, durée = 21, intial pose of ego = 0.000 0.000 , end pose of ego = 0.000 0.000

Figure 3: Lane Occlusion BEFORE/AFTER

Noise on agents every position We also tried to implement noise on every agent's position. The physical intuition behind this is that the center's position may be unprecise by a certain amount as not well detected, therefore we decided to add small gaussian noise of 5 cm.

Masking We also tried to improve the model by implementing masking of different time steps of the agents. We masked the past trajectory of the agents. With a probability p the position of an agent X would be masked. We tried different probabilities, but didn't notice considerable change. In the below example, the probability of an agent's past position to be masked is set at $p = 0.2$. The value is chosen through a uniform distribution between 0 and 1. This data modification was directly implemented in the ptr.py file in the forward method.

3 Trajectory distribution

When we look at the situations where the error in the test base is high, we see that these are usually the same types of scenario. There are a total of 8 possible trajectories in the test database, grouping together every possible movement. As it turns out, the train base does not cover all these cases uniformly. As a result, errors are mainly caused by the few trajectories in the train database. Below 4is a histogram representing the number of occurrences of each case in the test database. We can clearly see the dominance of one class, and a minority of rare cases.

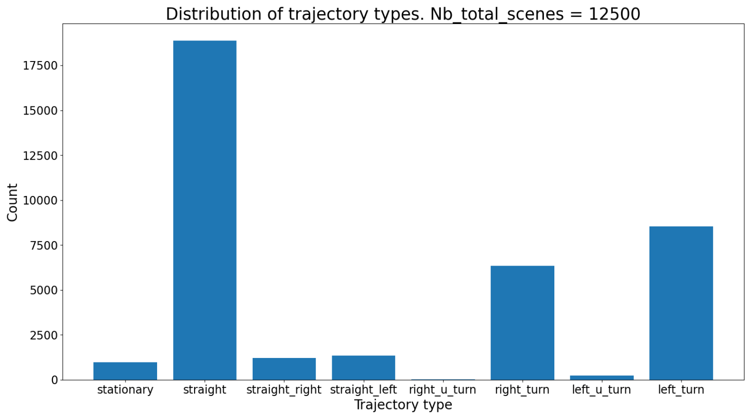


Figure 4: Trajectory distribution

To compensate and reduce the error caused by the lack of rare cases during training, we therefore tried to influence the loss: for each batch, we weighted the loss by the type of trajectory. In this way, rare trajectories will have a greater weighting in the loss: this impact will better penalize the network and therefore have an impact on learning. To calculate the weights, we used the frequency of each trajectory. However, we didn't notice any significant difference, even with the implementation of these weights.

4 Trick and Tips

In addition to the changes introduced in milestone 1, we've also made some minor changes.

- On the train curves, we noticed that the validation metrics sometimes remained constant for a long time. So we changed all the activation functions, using instead the LeakYRelu function, which avoids zero gradients.

- To avoid layer dominance, we have placed a layer normalization between each encoder layer. This resulted in better training, see Fig 5

- We also added some residual connections between each layer, in order to avoid the vanishing gradient. see Fig 7 : we notice a slight improvement on the training curves.

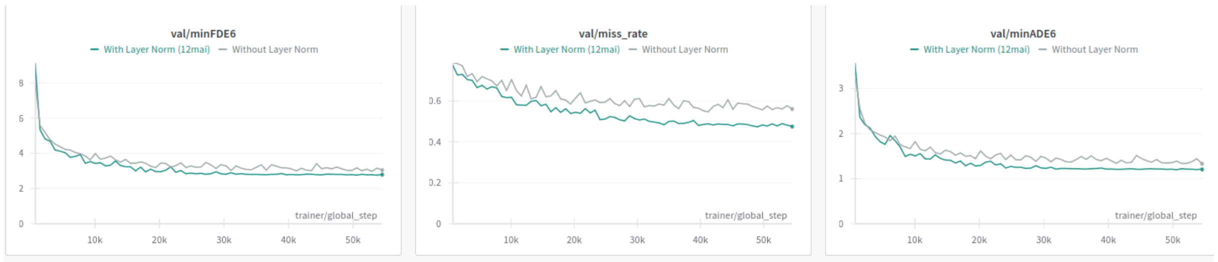


Figure 5: Effect on Layer Normalisation on 3 metrics for the validation set. The green dark curve is with layer normalization, the grey one is without.

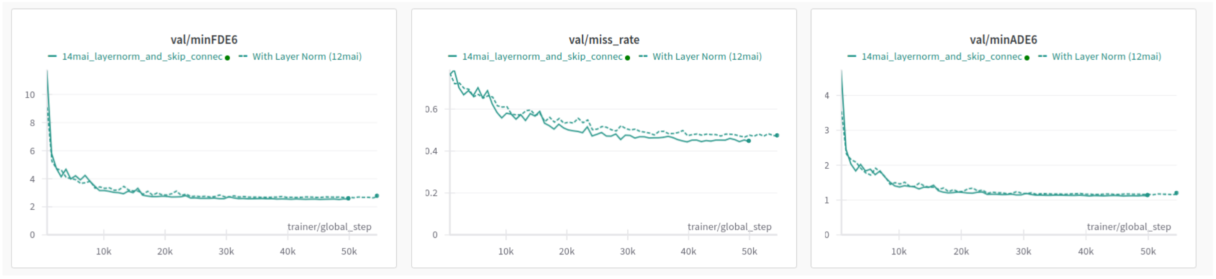


Figure 6: Effect of the residual connection on 3 metrics for the validation set. The full curve is with residuals, the other one is without.

5 Difficulties encountered

In opposition to the first Milestone we did not have any difficulties to reach the minimum score needed. Even with the previous milestone network we where able to reach a minimum score of 0.99 on the medium Dataset. However following this we had difficulties to improve the score. Even by implementing the previous points we did not see a noticeble jump in performance.

During training we also noted a certain amount of overfitting. The train score would go up to 0.7, however the validation score would stay at 1.5 and not go below. We faced this phenomenon even with the Data Augmentation, the masking, or the balanced loss fucntion, and we tried to increase the regularizer of the Adamw optimizer as a solution but did not see a better effect.

6 Results : training curves and Kaggle submission

With a 48-hour train, our best result on Kaggle was 0.98105 as minADE6 score.

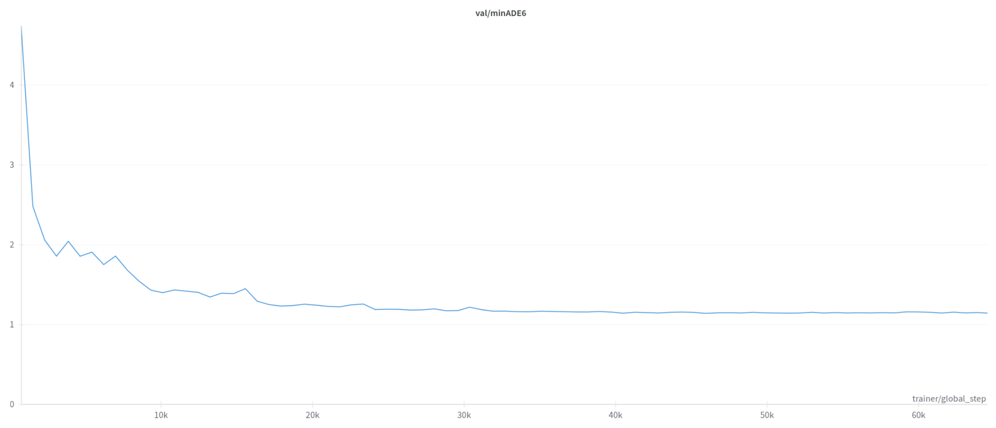


Figure 7: Loss of our training