Deep Learning for Autonomous Vehicules

Project Milestone 3

Victoria Nguyen, 385211
Valentin Perret, 327718
Code Git

April 2024

1 Introduction

The aim of milestone 3 is to integrate a second model, in addition to autobots, into the unitraj framework. This report describes our integration in terms of code, and our results. The code is provided in the github repo, on the Milestone 3 branch.

2 Integrating a new model to Unitraj : mmTransformer

We have chosen to implement Multimodal Motion Prediction with Stacked Transformers (abbreviated MMTransformer). The article is available at the following link. Their code is available at the following git repo.

2.1 Architecture

The model we chose is about stacked transformers in which the past trajectories, the road information, and the social interaction are aggregated with several transformer encoder-decoder module. Initially, a first transformer will encode the information for each individual vehicle, as well as the map's spatial information. Then, another transformer will encode the interaction between each agent. A decoder will then produce K trajectory proposals. Each proposal will have a score.

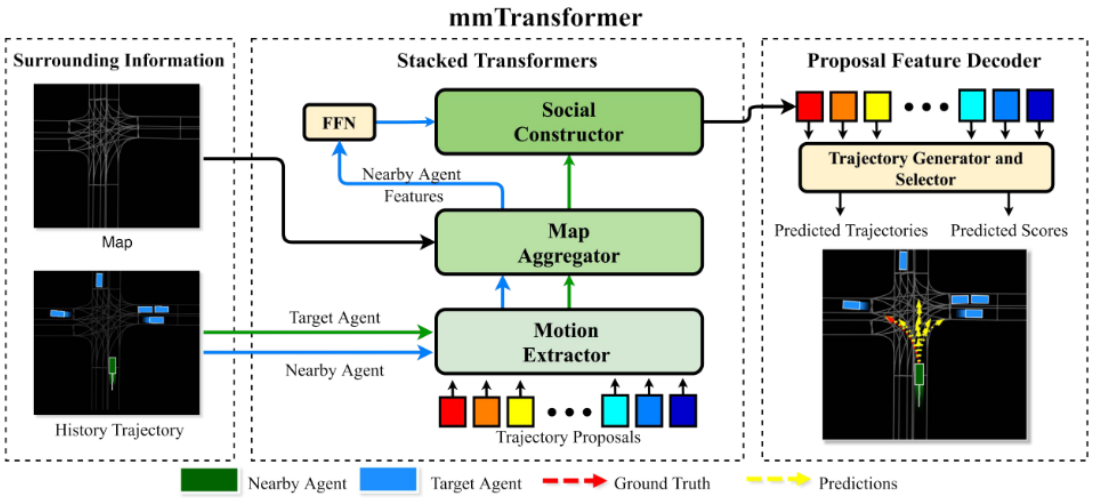

Figure 1: Architecture of the model provided in the paper.

2.2 Dataset and results provided

According to the official paper, the model was tested on Argoverse data, and achieved a score of 0.8435 in minADE.

Methods	minADE	minFDE	MR(%)
NN [6]	1.7129	3.2870	53.69
LSTM ED [6]	2.34	5.44	-
TNT [35] (4th)	0.9358	1.5384	13.28
LGCN [22] (7th)	0.8679	1.3640	16.34
LA [27] (21th)	0.9436	1.5486	21.79
WIMF [19] (8th)	0.8995	1.4220	16.69
mmTrans. (5th)	**0.8435**	**1.3383**	15.42
mmTrans.+RTS (1st)	0.8704	1.3688	**13.00**

Figure 2: Metrics comparison provided

2.3 Integration to Unitraj

2.3.1 Input arguments

To integrate the model into Unitraj, we had to modify the datsets class and the models class. The challenge was to ensure the right dimensions into the stacked transformers. The model takes into account four arguments:

- `HIST: [batch size, max_agent_num, 19, 4]`, the last dimensions corresponds to x,y, the timestep and the mask.
- `POS: [batch size, max_agent_num, 2]`, it represents the final position of each agent, with respect to the final position of the target agent.
- `LANE: [batch size, max_lane_num, 10, 5]`, the last 5 dimensions represent four dimensions of the lanes and the mask of the lanes.
- `VALID_LEN: [batch size, 2]`, represents per batch the number of valid agents and valid lanes.

For the equivalent, we used 'obj_trajs', 'obj_trajs_masks', 'map_polylines', 'map_polylines_mask', 'obj_trajs_last_pos' and 'track_index_to_predict'. We also had to define traj_valid_len (['VALID_LEN'][:,0]) : we assumed that the number of agents in our batch represents this number, and thus created a tensor at the size of the batch size, and containing the value of the nunmber of agents. Max_agent_num is also defined as the as the max number of valid of agent in the batch, here we suppose that the batch contains the same number of agents for every batch index. We made the same manipulation for lane_valid_len (['VALID_LEN'][:,1]) and max_lane_num.

2.3.2 Output arguments

In the first place, and given the short deadline, we decided to re-use the loss function used on the PTR model. We had to reshape the outputs : the stacked transformer model ouputs the predicted trajectory as a tensor of size [B, num agent, c, T, 2]. As the loss function expects only to get the predicted trajectory for the agent of interest and of size [B,c,T,5], we added 0.1 to all the three supplementary dimensions. It does not affect the computation of the metrics, as only x and y ar used, only the loss seems to be affected. Via the training we saw that the loss was different in order of magnitude due to this change versus the loss of the ptr model.

2.3.3 Tuning of certain points

Hyperparameters of the model We used the same hyper-parameters given in the git by the authors. We only changed two hyperparameters: future_num_frames from 30 to 60 and out_channels: from 60 to 120. This parameter corresponds to the GeneratorWithParallelHeads626 of the final layer. Initially with their hyperparameters the model would only predict the future 30 timesteps. Changing only future_num_frames was not enough and by changing out_channels we solved the issue. We also added our own optimizer : AdamW.

Prediction Generation We also changed the `generation_prediction function()` to `generation_prediction_mmTransformer()`. The model doesn't used the _forward method. We adapted it for our own needs.

3 Integrating new features in the existing model

3.1 Data augmentation

After discussing with the TA, we decided to improve the Data Augmentation especially the flip of the different curves. We modified the `HorizontalFlip` from MS2. We also decided to flip rare trajectories, ie the one with u-turns and turns. We flip along the x-axis.

3.2 Optimizers

- Using Pytorch lightening module, we added the adamW optimiser, using weigh regularisation.
- We also added a scheduler for the learning rate, the same as in the `PTR` model : the CosineAnnealingLR. Link Pytorch

3.3 Change of loss function

As mentioned above, we have initially reused the same loss function as the PTR model. What's more, the paper's open source code didn't provide an implementation of the training mode and therefore of the loss function, so we had to find out how to implement it, in addition to the model. However, this took some time, and Scitas had to wait 48 hours for each run. The results presented here are therefore obtained via loss, not from the paper, but from the PTR model.

The main difference is that the mm Transformer's loss is a multitask loss, allowing you to maintain diversity in the trajectories offered.

4 Results

Because of the time limit in Scitas before each run, we could not train for as long as we want. Thus, we reached a score of 1.62 on minAde6 on Kaggle.

The mmTransformer associated with PTR loss is no better than the PTR model itself, as it can be seen below.

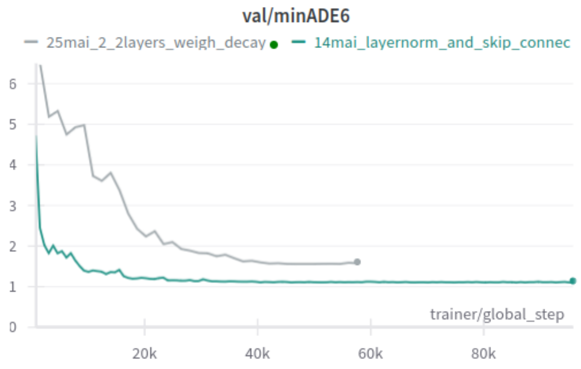

Figure 3: 2 validation curves : blue is PTR model, Grey is mmTransformer

Model	minAde6 [m]	minFDE6 [m]	Miss Rate [%]
PTR (Unitraj Dataset)	1.06	2.5	40
mmTransformer (Agroverse)	0.84	1.33	15.42
mmTransformer (Validation set)	1.59	3.8	62
mmTransformer (Hard set)	1.62	-	-

Table 1: Results obtained with the different models. The Hard set cannot give us the result about the minFDE6 and the missRate. For the ptr model we give the result of the minFDE6 and the missRate on the validation dataset.

5 Challenges

- The code only provides the inference mode, so we had to write and adapt the training.
- We had problems with resizing and matching data, sometimes ambiguous because we lacked documentation.
- The scitas clusters were full, so we weren't able to run as much training as we would like. Therefore we could not do some hyperparameters tunning. We could not make a training as long as for the ptr method, or train with the other loss function. With this we could have really compared the two methods, and see the effects more clearly.

6 Conclusion

This Milestone learnt us how to implement a new model into Unitraj. We were able to see the effect of state of the art neural networks. We obtained a minAde6 score on Kaeggle of 1.62.