EPFL

Deep Learning for Autonomous Vehicules

Project Milestone 1

Victoria Nguyen, 385211
Valentin Perret, 327718
Code Git

April 2024

1 Introduction

The aim of this project is to be able to predict the trajectory of an agent, given its near past, and those of the agents in the scene. We use the database provided to train the model: the aim of this first Milestone is to implement temporal attention, which focuses on important moments in the agent's past, and social attention, which emphasizes the link between different agents. This report will summarize the choices we have made to implement and improve the model, according to our chosen metric.

Our repo (including our README file): GitHub. The repo is private, only our TA is able to access it right now.

2 Implementation of Attention

2.1 Without attention

In order to get a starting score, we ran an initial training session with the code provided, i.e. without an attention layer. It performed a minADE6 of around 4m on the validation set. On Kaggle, this gives us an error of around 6m on the easy test set. **Our goal is now to improve the training and the final score.**

2.2 With attention

We had 2 key functions to implement. They allow us to apply attention to each agent and/or each time step. The provided ptr.py file implements a transformer-based architecture : the modele is called Autobot. We completed the two functions (see the code), in order to have two differents attentions layers :

- Temporal attention : Here, we only focus on the first dimension of every embedded agent. Temporal layer is applied to each timestep.
- Social attention : Here, we apply a social attention layer for each agent. In both cases, we had to reshape the dimension to fit the expected input shape.

When comparing our implementation with the original one, as discussed with our TA, we decided to use the original implementation.

3 Our changes to the Unitraj framework

3.1 Our ideas to improve the model

To tune our model we ran different models with different parameters. Several parameters were quickly identified as crucial: **the learning rate**, **the complexity of the model** and **the scheduler**.

3.1.1 Hyperparameters Tuning

With the basic model, we were already having a minADE6 of 4.38 on Kaeggle. In comparison with a more complex model, i.e. 8 layers for the encoder and 512 for the hidden size, but a wrongly-initialized scheduler and learning rate, led us to a score higher than 10. This showed us the importance of the right learning rate.

At first, we thought increasing the complexity of the model will give us better performances. We had several attempts by varying the learning rate at an encoder layer number of 8 and a hidden size of 512. However the minimum score did no go under 4 : it reached a plateau. You can find an example of results we had a Fig. 1. For example with the Figure as the validation reached a plateau we thought about changing the scheduler, or play with the regularizer as this showed for us some overfitting. In the end, none of those solution helped.

Overall we tried to play with different parameters, the dropout, the batch size, or the regularizer. None seem to be able to go beyond the value of 4. We also noticed that by just slightly increasing the complexity, that the model was not able to be as precise anymore. Therefore we tried the simple model again and just reduced the hidden size and the number of attention heads. This showed us that actually we needed a simpler model. We tuned a bit the parameters and rapidly went below the minimum score.

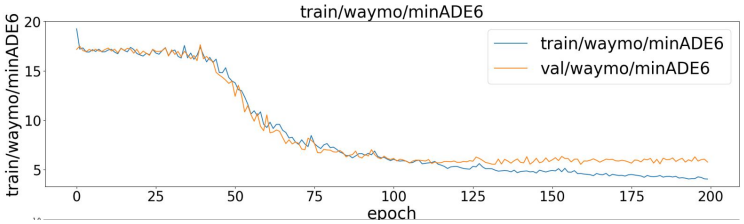


Figure 1: Figures showing the evolution of the MinADE6 criterion, during the training phase for an architecture of eight encoder layers, a hidden size of 512, a scheduler of [20, 40, 80, 110, 160], a train batch size of 256 and 200 epochs. The non metionned parameters are the same as the initial one.

3.1.2 Importance of the scheduler.

Once we had a good model, the difficulty came with the plateau that was reached after a certain number of epochs.

Initializing the learning rate is complex: that's why we also looked at how to modify the scheduler. We followed the Pytorch documentation. We tested 3 schedulers, depending on the loss evolution at each epoch:

- **MultiStep LR** : it reduces the lr at the end of each epoch specified in the parameter
- **ReduceLROnPlateau** : As explained above, our training often remained stagnant: the gradient values were very low, so we had to wait a large number of epochs before seeing any change. So, to avoid having to specify the epochs when the lr should be changed, we tried the ReduceLROnPlateau scheduler: the lr decreases as soon as the metric is stagnant.
- **CosineAnnealingLR** : use a cosine annealing schedule as explained in the documentation.

The CosineAnnealingLR gave us the best result.

3.1.3 Weight initialization

The plate on Fig 1 suggests that the weights have been incorrectly initialized: nothing happens at the start of training. We've therefore tried to modify this aspect slightly. The original code only initialized the weights according to a normal distribution. The bias was fixed constant. So we simply initialized all parameters, weights and biases, according to a normal distribution :

```
init_ = lambda m: init(m, nn.init.xavier_normal_, lambda x: nn.init.xavier_normal_(x, 0), np.sqrt(2))
```

3.2 Final parameters and result

Table 1 shows the hyperparameters for our best model. The training and validation curve for the minADE6 criterion can be found on Fig 2. Finally, visualizations are provided on Fig 3.

Hyperparameter	Original value	Value chosen
Nb epoch	300	250
Learning rate	0.00075	0.00075
Scheduler	MultiStepLR	CosineAnnealingLR
Dropout	0.1	0.1
Number of encoder layers	2	2
Number of decoder layers	2	2
Hidden size	128	64
Attention heads	16	16

Table 1: Summary of the hyper-parameter's value chosen.

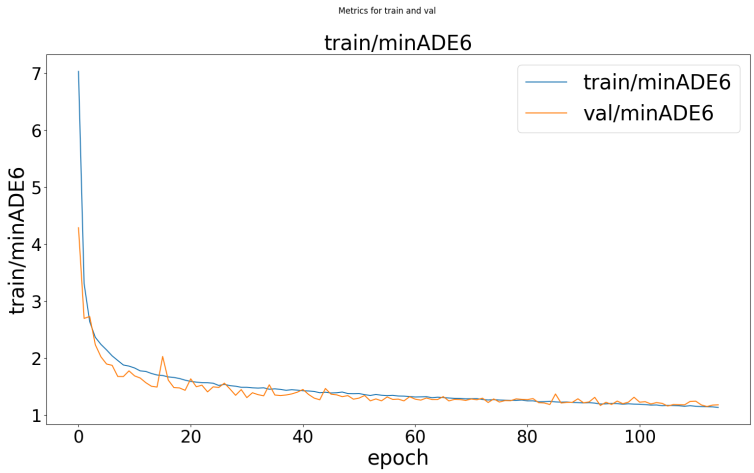
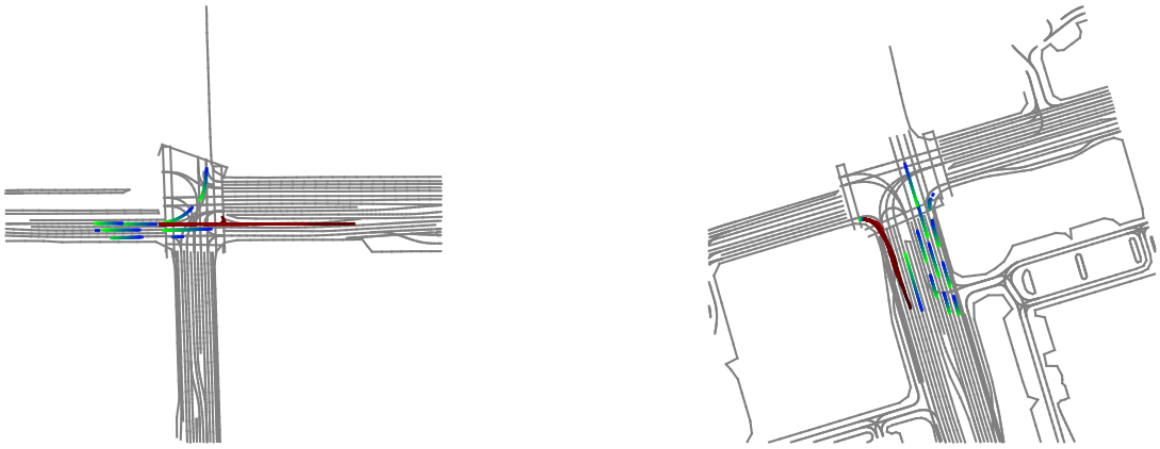


Figure 2: Evolution of our main metric during the training process



(a) Trajectory 1 (b) Trajectory 2

Figure 3: Some visualizations of predictions with our best model

4 Challenges faced

- First, we had to learn to use Scitas, and it proved to tricky. We were always pushed back in the queue, and it wasn't until 5 days before the deadline that we finally had a code running, with the help of our assigned TA.

- As said previously with our intuition we thought about making the network a little bit more complex since the task is ambitious. However, as discussed with our TA, and according to our differents submissions attempts, we realized that making the model less complex (ie less layers), can improve our performance, which is not intuitive at first sight.

5 Conclusion

Overall this milestone developed the basics needed for the rest of the project, and has build some foundations for later in the project. We were able to reach a minimum score of 1.04939 on the minADE6 criterion on Kaeggle, which is below the minimum score of 1.074 required.