Intellisys - Project Report

2nd August 2021

Progresses

We have been trying to debug the code, to find why the performance of the network does not seem to improve as much as we would expect and why the final value of the loss (L1 with mean) does not correspond to the plots of the outputs.
We have been unit testing and checking all the outputs and in between steps. Full prints can be seen in the git repo. Specifically we tried to :

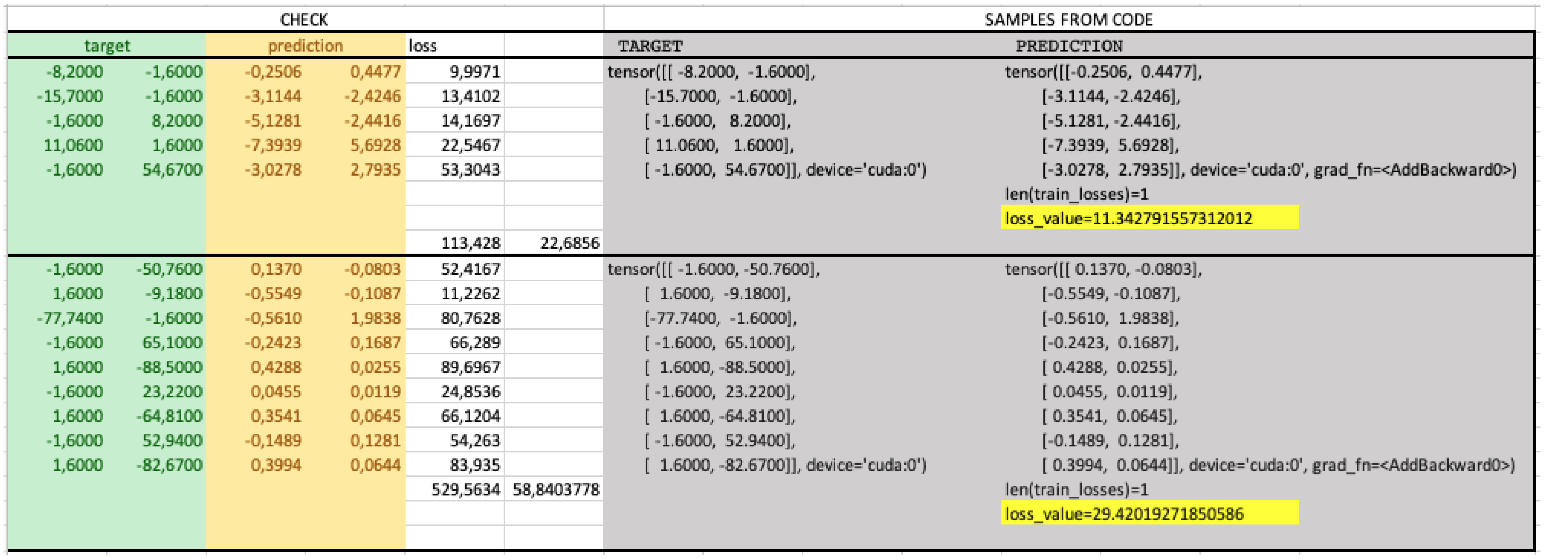
- take the delta position instead of the absolute final position as target, with small train datasets didn't return considerable improvement, but need to be tested for bigger datasets.
- normalize the input with min max normalization and standardization: currently training.
- check the size of the input and the target to be sure that this is not causing any issues, using batch size 1 and printing out. the input can be of different size, eg each input has different number of cars.
- check the loss L1 function from pytorch, comparing with handmade L1 loss function. Given an input matrix with n rows (vehicles in one graph) and m (columns features) the calculation of the loss for each vehicle is correct, the sum is correct, the mean is calculated by diving the sum by n*m instead than only by the number of vehicles. so the number we see in the loss is half of what it is in fact. This shouldn't have an impact on the training, but doubling the loss is more fitting the current pictures. See Fig 1.
- check plot function, which is correct. Added zoom in to see better what happens locally. see Fig 2
- check the input data, counting how many input rows/graphs contain only vehicles that are moving and how many contains only vehicles that are still. The number of cases in which the vehicles are all still is much higher than the number of cases in which the vehicles are all moving (which is in fact only a small fraction of the total, depending on the dataframe between 0.02% and 0.10%). See for example two of the big datasets with 15000 and 30000 rows of inputs Fig 3. In the plots, we selected only frames in which all the vehicles where moving, which as said is a very little fraction of the dataset (this could explain the bigger difference between the min loss and the actual loss in these plots).
- train with balanced the input data: concatenating all the datasets and taking the same amount on rows where all the cars are all moving and where some cars are moving and some are standing still
- try to create a even more simple dataset with only 5 vehicles over 250000 steps and training with it (currently on going)
- try with different input columns (eg including or excluding the intention column doesnt seem to have an impact)
- check on which frames seem to be the best ones for the predictions and which ones are the worst. See Fig 4, 5.

Issues/Questions

1. What could we have missed, review of print outs?
2. Regarding the presentation and report content: suggestion on how to present the project if we dont manage to get a network to give us accurate enough predictions.
3. Formals regarding the presentation and the report:
   (a) How many minutes/slides? how many pages?
   (b) Are there templates we need to use?
   (c) Structure proposal: Problem description, problem motivation, method(s), experiments, results, future work
4. confirmation of the date for presentation, delivery of report and QA.

Tables and Figures

Figure 1: Check on L1 loss



CHECK					
target		prediction		loss	
-8,2000	-1,6000	-0,2506	0,4477	9,9971	
-15,7000	-1,6000	-3,1144	-2,4246	13,4102	
-1,6000	8,2000	-5,1281	-2,4416	14,1697	
11,0600	1,6000	-7,3939	5,6928	22,5467	
-1,6000	54,6700	-3,0278	2,7935	53,3043	
				113,428	22,6856
-1,6000	-50,7600	0,1370	-0,0803	52,4167	
1,6000	-9,1800	-0,5549	-0,1087	11,2262	
-77,7400	-1,6000	-0,5610	1,9838	80,7628	
-1,6000	65,1000	-0,2423	0,1687	66,289	
1,6000	-88,5000	0,4288	0,0255	89,6967	
-1,6000	23,2200	0,0455	0,0119	24,8536	
1,6000	-64,8100	0,3541	0,0645	66,1204	
-1,6000	52,9400	-0,1489	0,1281	54,263	
1,6000	-82,6700	0,3994	0,0644	83,935	
				529,5634	58,8403778

SAMPLES FROM CODE

TARGET	PREDICTION
tensor([[-8.2000, -1.6000], [-15.7000, -1.6000], [-1.6000, 8.2000], [11.0600, 1.6000], [-1.6000, 54.6700]], device='cuda:0')	tensor([[-0.2506, 0.4477], [-3.1144, -2.4246], [-5.1281, -2.4416], [-7.3939, 5.6928], [-3.0278, 2.7935]], device='cuda:0', grad_fn=<AddBackward0>) len(train_losses)=1 loss_value=11.342791557312012
tensor([[-1.6000, -50.7600], [1.6000, -9.1800], [-77.7400, -1.6000], [-1.6000, 65.1000], [1.6000, -88.5000], [-1.6000, 23.2200], [1.6000, -64.8100], [-1.6000, 52.9400], [1.6000, -82.6700]], device='cuda:0')	tensor([[0.1370, -0.0803], [-0.5549, -0.1087], [-0.5610, 1.9838], [-0.2423, 0.1687], [0.4288, 0.0255], [0.0455, 0.0119], [0.3541, 0.0645], [-0.1489, 0.1281], [0.3994, 0.0644]], device='cuda:0', grad_fn=<AddBackward0>) len(train_losses)=1 loss_value=29.42019271850586

Figure 2: Full plot, zoom plot

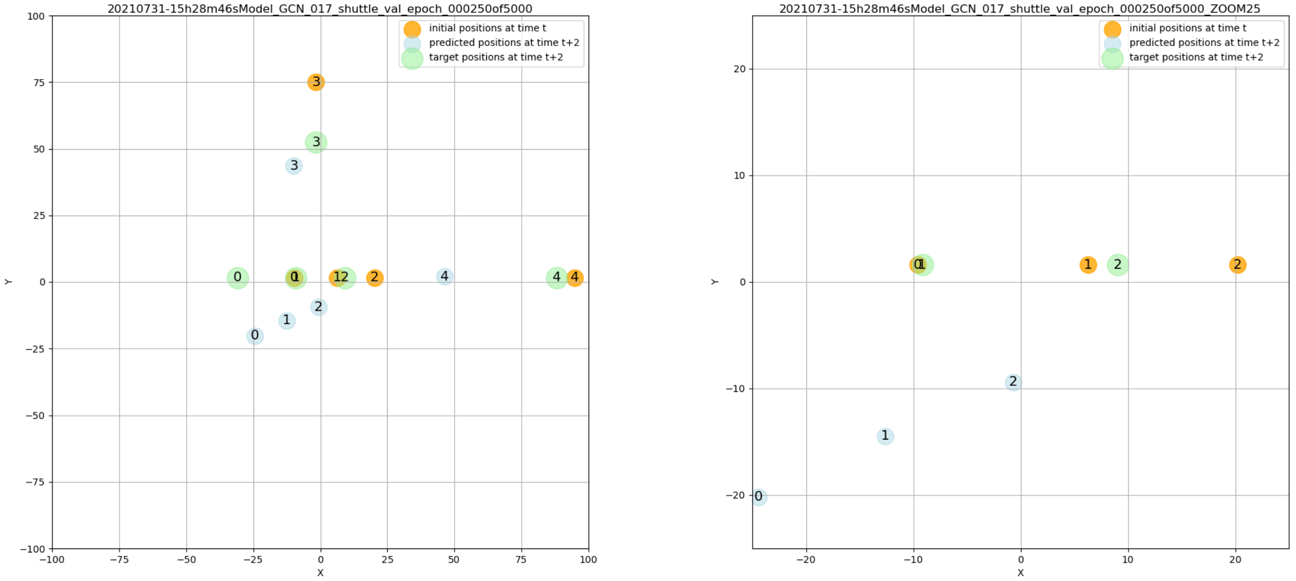
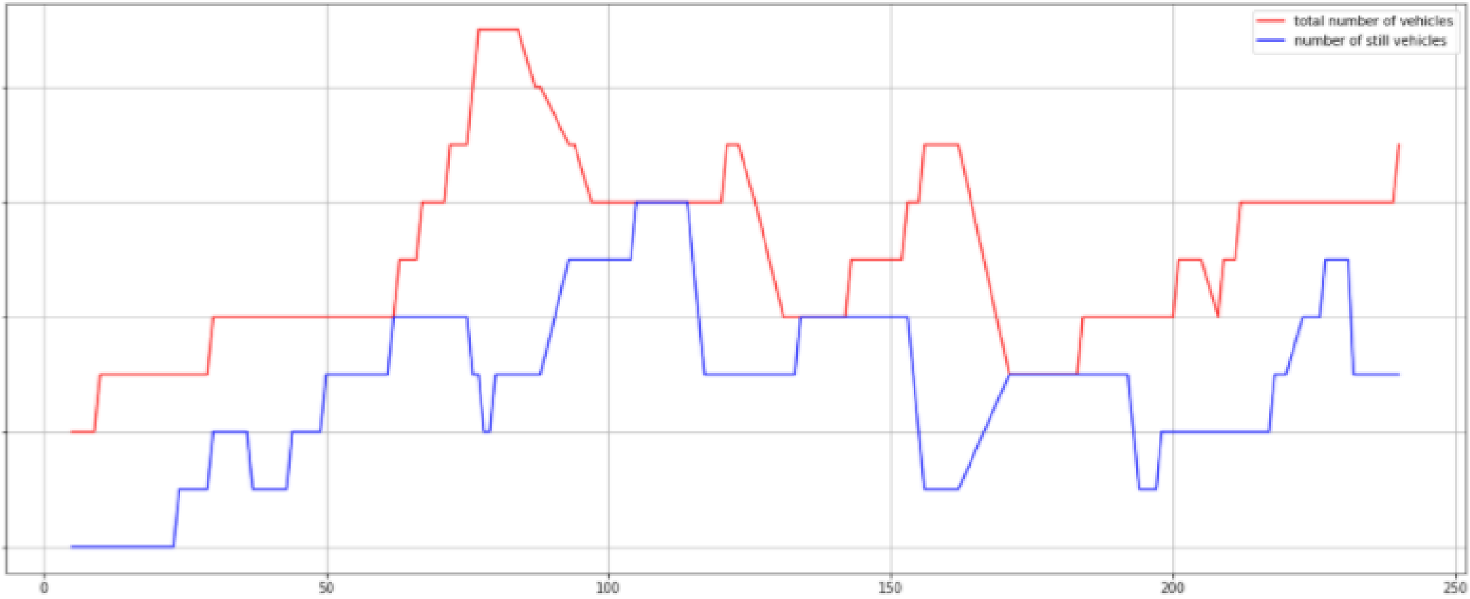


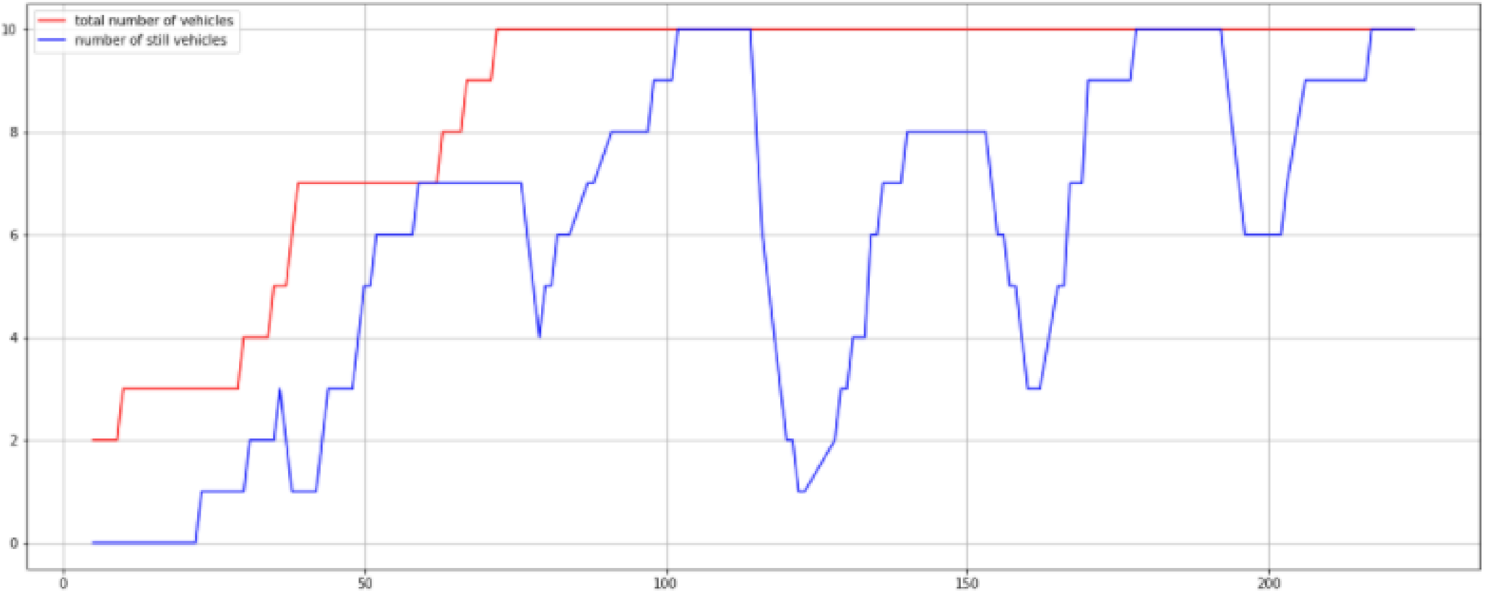
Figure 3: dataset balance

```
20210710-20h38m27s_timesteps14930_ec3500_em7000
```



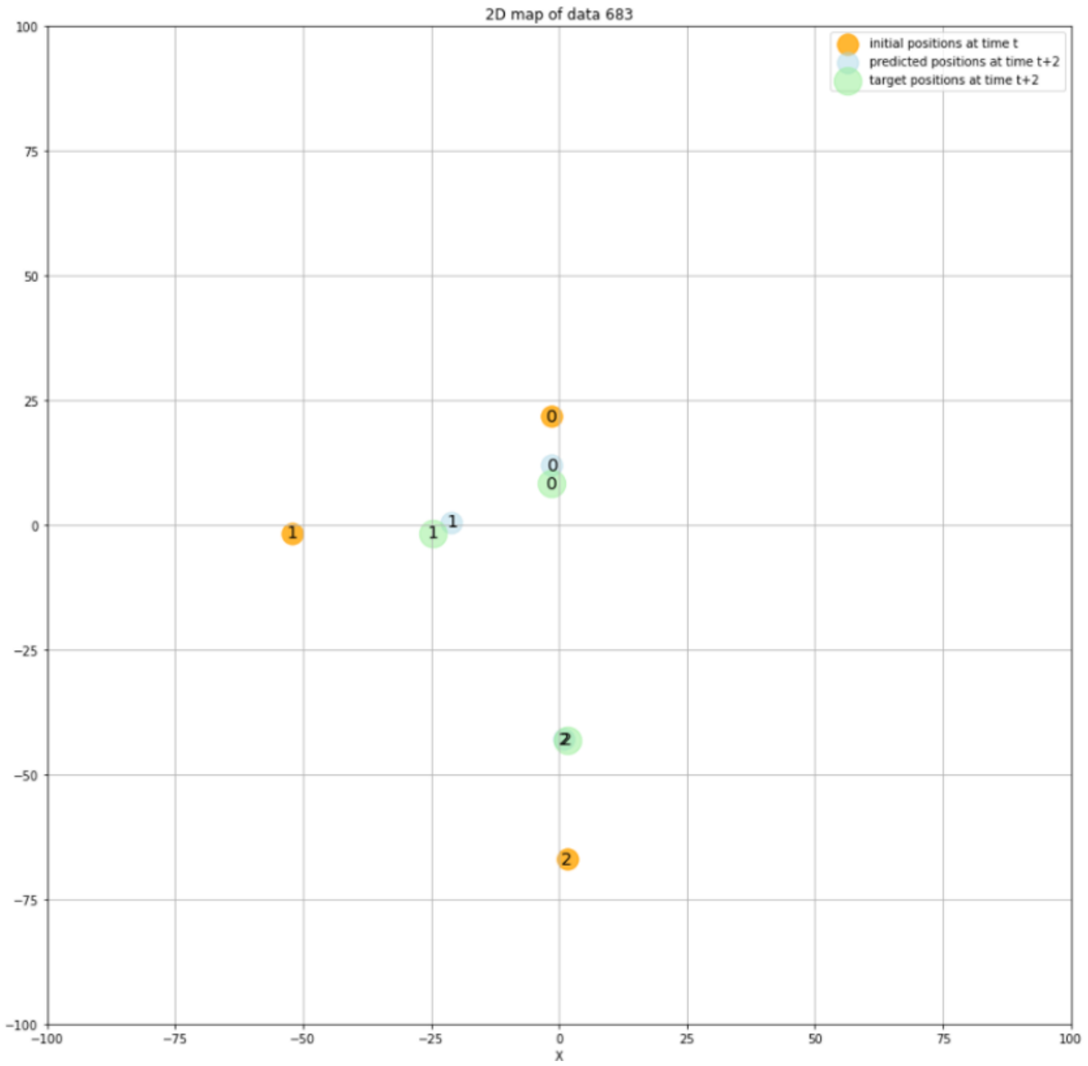
```
in how many rows all the vehicles are moving?
0.07301317467063323
on the totality of input rows, how many contain vehicles that are moving ?
0.4433098348192688
```

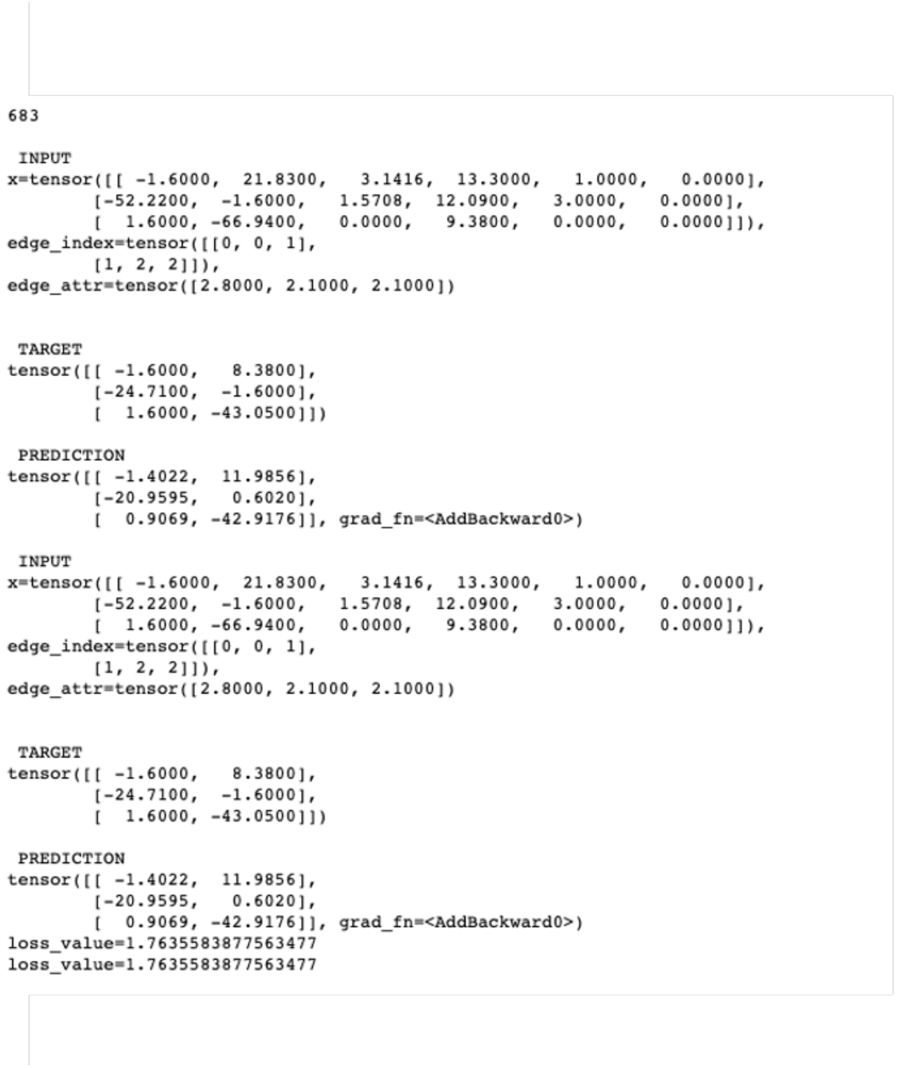
```
20210711-17h59m44s_timesteps30000_ec3500_em7000
```



```
in how many rows all the vehicles are moving?
0.026305253042921205
on the totality of input rows, how many contain vehicles that are moving ?
0.2929934263267597
```

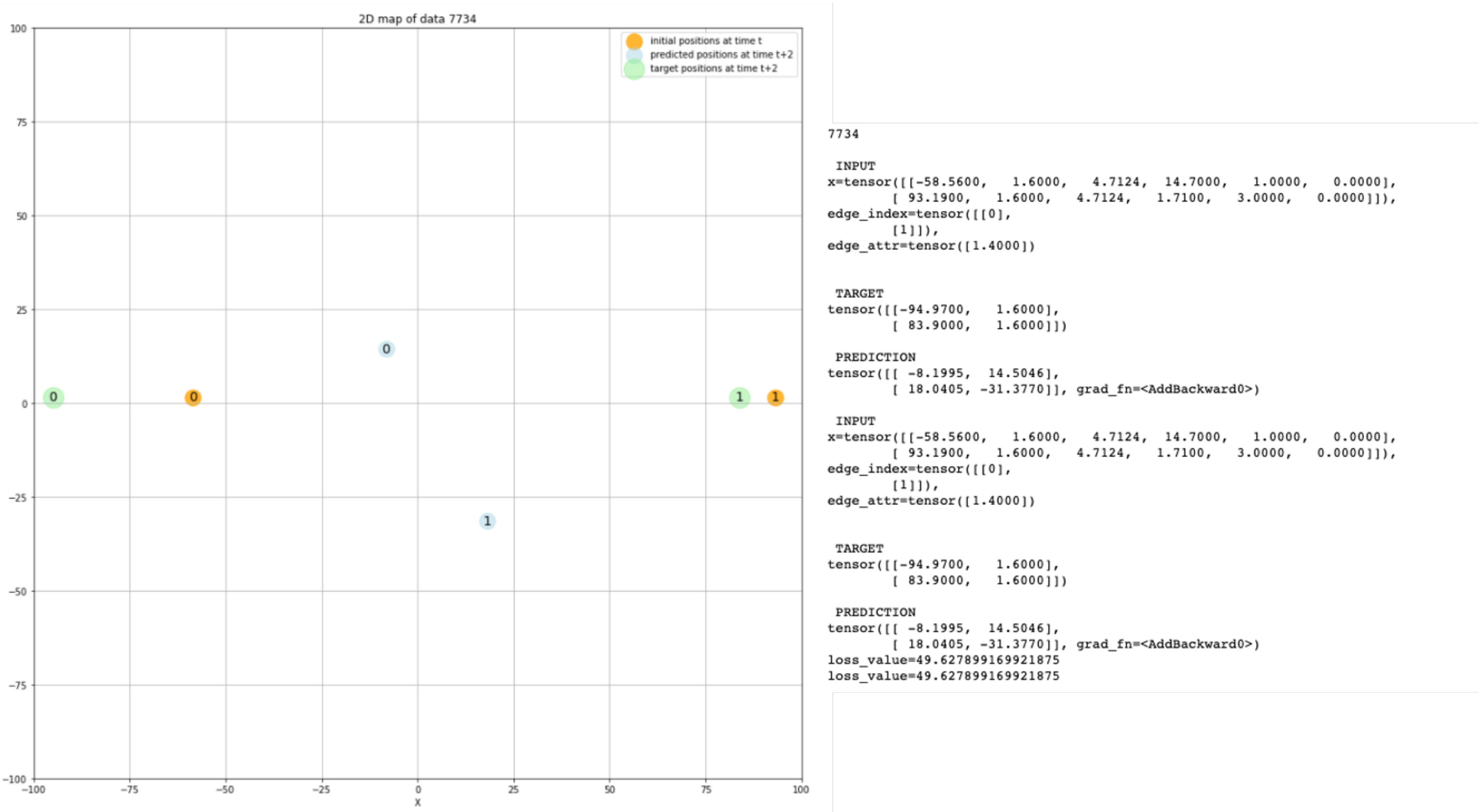
Figure 4: example of good prediction





```
683

 INPUT
x=tensor([[ -1.6000,  21.8300,   3.1416,  13.3000,   1.0000,   0.0000],
        [-52.2200,  -1.6000,   1.5708,  12.0900,   3.0000,   0.0000],
        [  1.6000, -66.9400,   0.0000,   9.3800,   0.0000,   0.0000]]),
edge_index=tensor([[0, 0, 1],
        [1, 2, 2]]),
edge_attr=tensor([2.8000, 2.1000, 2.1000])


 TARGET
tensor([[ -1.6000,   8.3800],
        [-24.7100,  -1.6000],
        [  1.6000, -43.0500]])

 PREDICTION
tensor([[ -1.4022,  11.9856],
        [-20.9595,   0.6020],
        [  0.9069, -42.9176]], grad_fn=<AddBackward0>)

 INPUT
x=tensor([[ -1.6000,  21.8300,   3.1416,  13.3000,   1.0000,   0.0000],
        [-52.2200,  -1.6000,   1.5708,  12.0900,   3.0000,   0.0000],
        [  1.6000, -66.9400,   0.0000,   9.3800,   0.0000,   0.0000]]),
edge_index=tensor([[0, 0, 1],
        [1, 2, 2]]),
edge_attr=tensor([2.8000, 2.1000, 2.1000])


 TARGET
tensor([[ -1.6000,   8.3800],
        [-24.7100,  -1.6000],
        [  1.6000, -43.0500]])

 PREDICTION
tensor([[ -1.4022,  11.9856],
        [-20.9595,   0.6020],
        [  0.9069, -42.9176]], grad_fn=<AddBackward0>)
loss_value=1.7635583877563477
loss_value=1.7635583877563477
```

Figure 5: example of bad prediction


2D map of data 7734
initial positions at time t
predicted positions at time t+2
target positions at time t+2
100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100
X
0
0
0
1
1
1

```
7734

 INPUT
x=tensor([[-58.5600,   1.6000,   4.7124,  14.7000,   1.0000,   0.0000],
        [ 93.1900,   1.6000,   4.7124,   1.7100,   3.0000,   0.0000]]),
edge_index=tensor([[0],
        [1]]),
edge_attr=tensor([1.4000])


 TARGET
tensor([[-94.9700,   1.6000],
        [ 83.9000,   1.6000]])

 PREDICTION
tensor([[ -8.1995,  14.5046],
        [ 18.0405, -31.3770]], grad_fn=<AddBackward0>)

 INPUT
x=tensor([[-58.5600,   1.6000,   4.7124,  14.7000,   1.0000,   0.0000],
        [ 93.1900,   1.6000,   4.7124,   1.7100,   3.0000,   0.0000]]),
edge_index=tensor([[0],
        [1]]),
edge_attr=tensor([1.4000])


 TARGET
tensor([[-94.9700,   1.6000],
        [ 83.9000,   1.6000]])

 PREDICTION
tensor([[ -8.1995,  14.5046],
        [ 18.0405, -31.3770]], grad_fn=<AddBackward0>)
loss_value=49.627899169921875
loss_value=49.627899169921875
```