TSFS12 Hand-in 5:
Learning for Autonomous Vehicles

JOHN DOE
E-mail: email@example.com
LiU-ID: johndoe

SAMUEL ÅKESSON
E-mail: email@example.com
LiU-ID: sermuns

2025-11-24

1. Introduction

2. Results for 4.

Exercise 4.2

$V(s)$ is a function mapping a state s to a number representing how "good" that state is. In this context, being closer to the goal node is "good".

Exercise 4.3

This function encapsulates the agent's possibility of making mistakes. It returns three possible states, with their respective rewards and probabilities. Only one of these is desired, the other two represent missteps in the perpendicular direction.

Computing the next value for $V(s)$ means finding the a that maximizes the reward, despite its errors.

Exercise 4.5

The given solution is (3,0), (2,0), (2,1), (2,2), (2,3), (2,4), (3,4). Since our agent is "clumsy" it could be more worth walking further away from the cliff, but in this case where we make our intended action 99% of the time, it seems not to be penalizing to walk so close to the cliff.

The plot of $V(s)$ matches our description in Exercise 4.2– states closer to the goal have lower penalty than those further.

Exercise 4.6

Setting `P_move_action` to 1 makes the agent infallible. The value function is very obviously outlining the solution now.

Decreasing `P_move_action` just a little below 0.99, makes the agent reconsider the solution and choose a lengthier one to avoid tripping into the cliff. At 0.8, even the starting action is to try to walk left, since it is too costly to fall into the cliff to the right.

We predicted that a more clumsy agent would pick a longer detour to avoid the cliff, which is what happened as `P_move_action` increased.

Exercise 4.7

γ=1 makes the agent completely future-aware, considering the full weight of all the planned actions. Moving γ closer to 0 makes the agent take actions in the heat of the moment. For an optimal solution, you would want γ as close to 1 as possible, but decreasing it makes the computation more lightweight.

In our tests decreasing γ reduced the number of iterations needed to converge on a solution.

Exercise 4.8

Making `R_sink` more negative makes, even the agent with `P_move_action`=0.99 take a detour to avoid walking next to the cliffs. Already at `R_sink`=-40, the detour happens.

Making `R_sink` more positive makes the agent consider wading through the cliffs to get to the goal node. At `R_sink`=-0.5, the solution involves walking through the cliffs.

Exercise 4.10

The Q-function maps pairs of (state, action) to a "value"/"quality". Constructing the optimal policy function could be done by finding the maximum valued action for every state and assigning it.

Exercise 4.11

The Q-learning and the value iteration converges to the same optimal policy when using the same parameters.

Exercise 4.12

When varying `P_move_action` for both methods they behaved very similar. For `P_move_action` at 0.9 value iteration proved to be somewhat more risky than Q-learning. At 0.8 value iteration takes the safe route while Q-learning still takes some risks. However when lowering even more to 0.6 both methods take the safest route to avoid the cliffs.

Value iteration seems to play it more safe than Q-learning.

The states outside the solution path do not seem to have a clear correlation to `P_move_action`

Exercise 4.13

Having epsilon decrease over time, given by

$$f(x) = 1 - \frac{\text{itr\_nbr}}{\text{nbr\_iters}} \tag{1}$$

makes the final iteration finish with a higher reward sum. Using constant epsilon=0.5 makes the final iteration finish with around -4, but with this linearly decaying epsilon, the final iteration finishes with around 0. This is shown in Figure 1.

Making epsilon vary in a more advanced pattern, given by

$$g(x) = \frac{1}{2}\left(1 + \cos\left(2\pi \cdot \frac{\text{itr\_nbr}}{\text{nbr\_iters} \cdot \frac{2}{3}}\right)\right) \tag{2}$$

the reward sum is still around 0 after the iterations. This is shown in Figure 2.

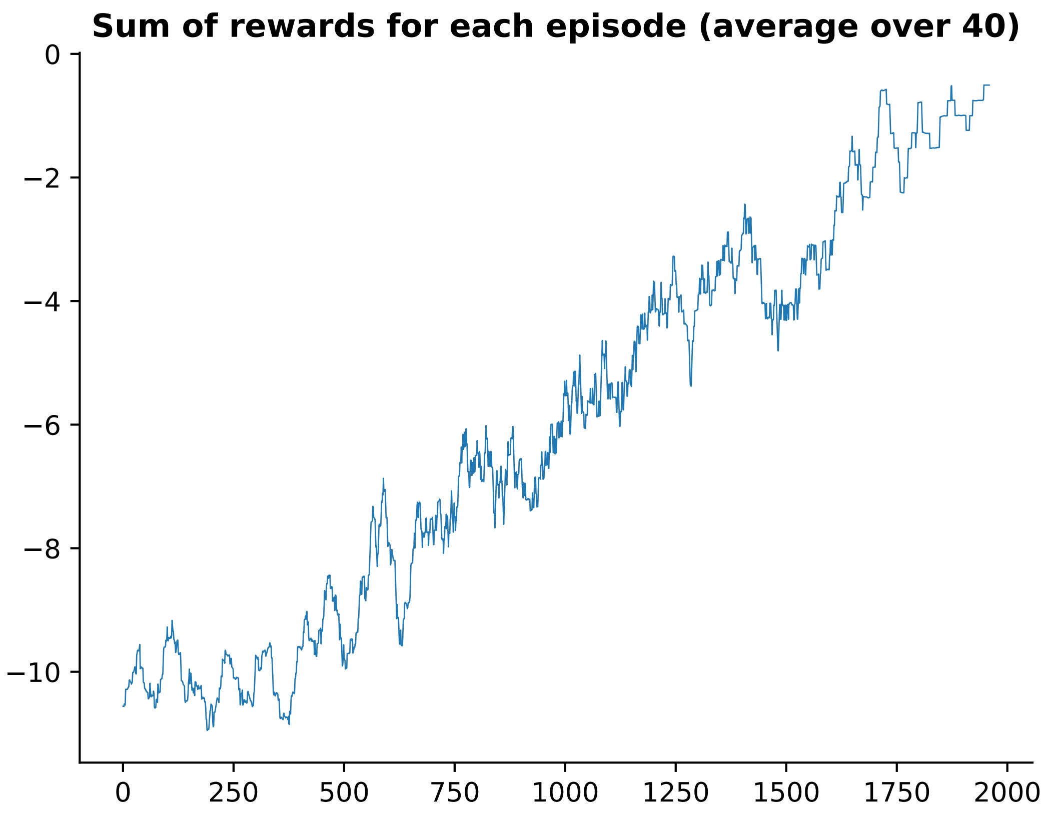

Figure 1: Sum of rewards as epsilon=$f(x)$ (1)

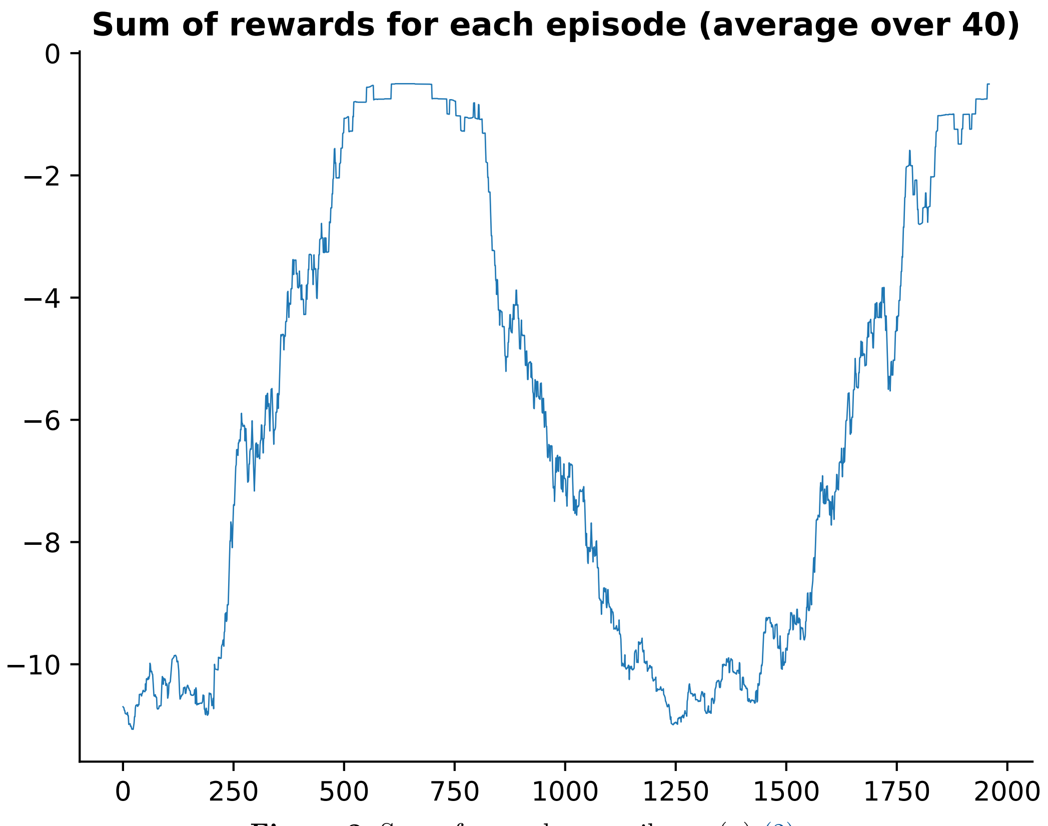

Figure 2: Sum of rewards as epsilon=$g(x)$ (2)

Exercise 4.14

Q-learning is a model-free method that can work in an unknown environment and learn the characteristics by interacting with it, which means the updates are incremental and depend on exploration. Whereas value iteration converges in fewer iterations because it is a model-based method that requires knowledge of the environment to compute the results.

3. Results for notebook

1.4

- 279 batches, and therefore $279 \cdot 32 = 8928$ scenarios.
- 885 agents in first batch.
- 20 and 20.
- 4 types

-
  ```
  tensor([[ 3.4497e-02, -3.0300e-03],
          [ 1.1784e-01, -9.8290e-03],
          [ 2.8071e-01, -1.2697e-02],
          [ 5.5216e-01, -1.0583e-02],
          [ 9.2878e-01, -1.8613e-03],
          [ 1.3911e+00,  1.4750e-02],
          [ 1.9319e+00,  4.6724e-02],
          [ 2.5487e+00,  9.8399e-02],
          [ 3.2096e+00,  1.7322e-01],
          [ 3.9486e+00,  2.7497e-01],
          [ 4.7648e+00,  4.5830e-01],
          [ 5.5812e+00,  6.8636e-01],
          [ 6.4334e+00,  9.8501e-01],
          [ 7.2944e+00,  1.3522e+00],
          [ 8.1671e+00,  1.7691e+00],
          [ 9.0386e+00,  2.2472e+00],
          [ 9.9020e+00,  2.7851e+00],
          [ 1.0745e+01,  3.3773e+00],
          [ 1.1556e+01,  4.0142e+00],
          [ 1.2329e+01,  4.6693e+00],
          [ 1.3094e+01,  5.3208e+00],
          [ 1.3848e+01,  5.9749e+00],
          [ 1.4614e+01,  6.6231e+00],
          [ 1.5401e+01,  7.2951e+00],
          [ 1.6227e+01,  7.9856e+00]])
  ```
- 34 vehicles in batch 20.

2.3

Running on the school computers.

```
%%timeit -n 10 -r 5
with torch.no_grad():  # Turn off gradients computation
    _, pred, _, _ = model_nomap(data)
```

7.55 s ± 75.8 ms per loop (mean ± std. dev. of 5 runs, 10 loops each)

```
%%timeit  -n 10 -r 5
with torch.no_grad():  # Turn off gradients computation
    _, pred, _, _ = model_map(data)
```

8.65 s ± 23.1 ms per loop (mean ± std. dev. of 5 runs, 10 loops each)

```
%%timeit -n 10 -r 5
with torch.no_grad(): # Turn off gradients computation
    _, pred, _, _ = model_gatmap(data)
```

10.5 s ± 93.7 ms per loop (mean ± std. dev. of 5 runs, 10 loops each)

```
%%timeit -n 1000 -r 5
cv_pred = constant_velocity(data, 25, STEP_SIZE)
```

26.4 ms ± 101 µs per loop (mean ± std. dev. of 5 runs, 1,000 loops each)

2.4

The prediction model gatmap, shown in Figure 3, generates the best prediction for the paths. For agent 12 and 15 the prediction follows the targeted path closely. However for agent 20 the prediction deviates from the targeted path because of the right turn it takes. The observed path is before the turn and therefore the prediction is to continue straight forward.

The constant-velocity model, shown in Figure 4, should work fine on straight roads, and work terribly in curved roads, especially in intersections where cars can turn ±90°.

Figure 5 and Figure 6 show the prediction models map and nomap which have very similar predictions but the map model is slightly better at following the targeted path.

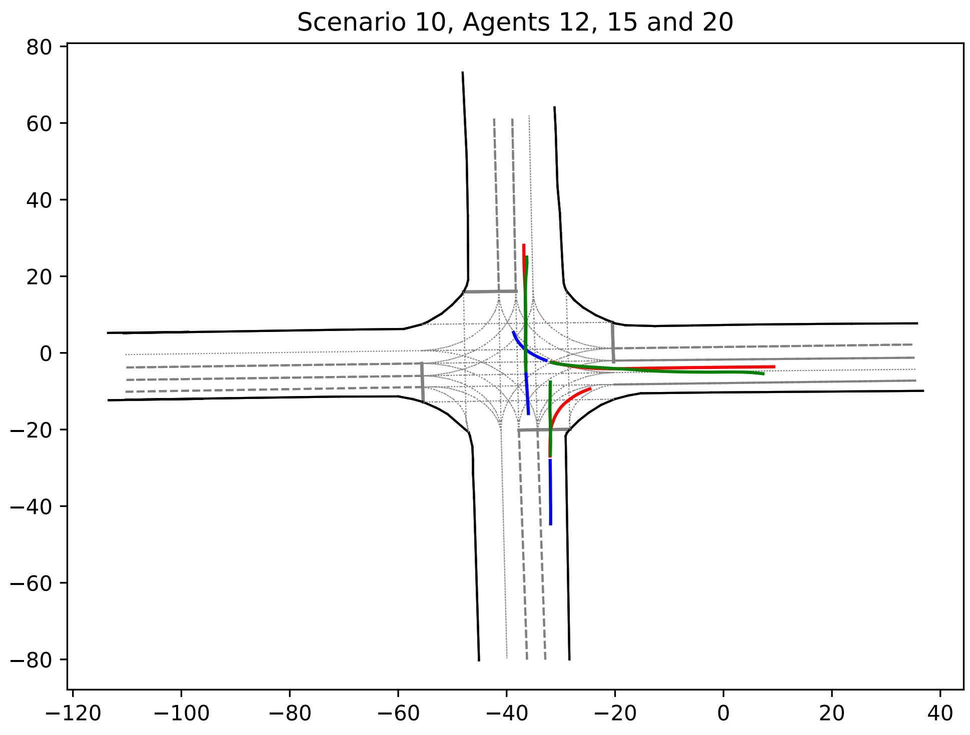

Figure 3: Predictions using the model_gatmap

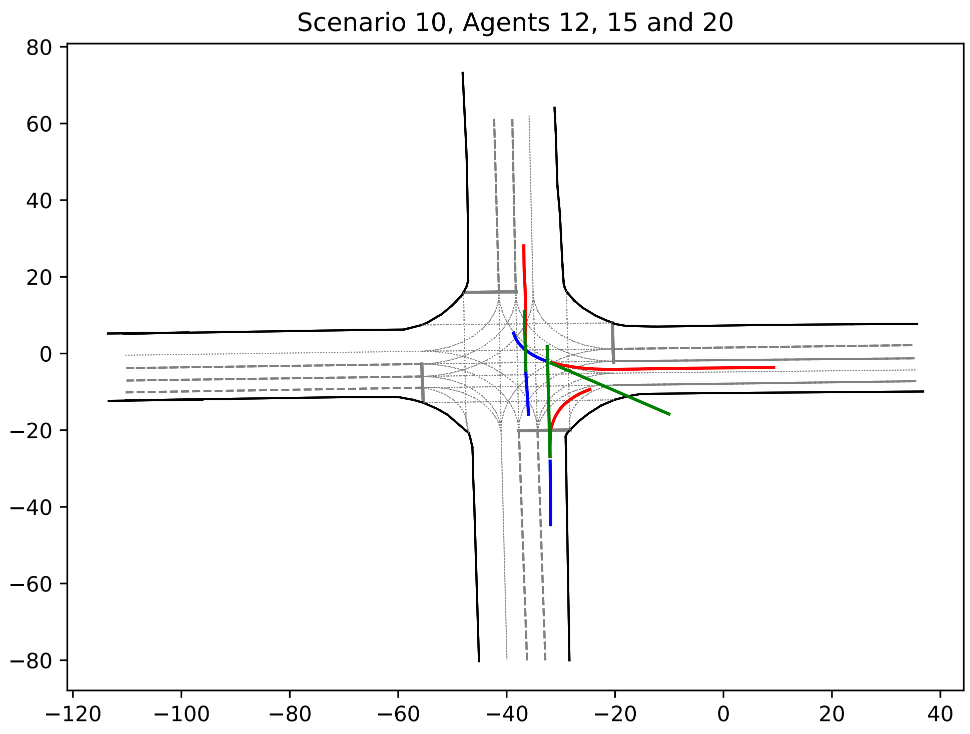

Figure 4: Predictions using the constant_velocity

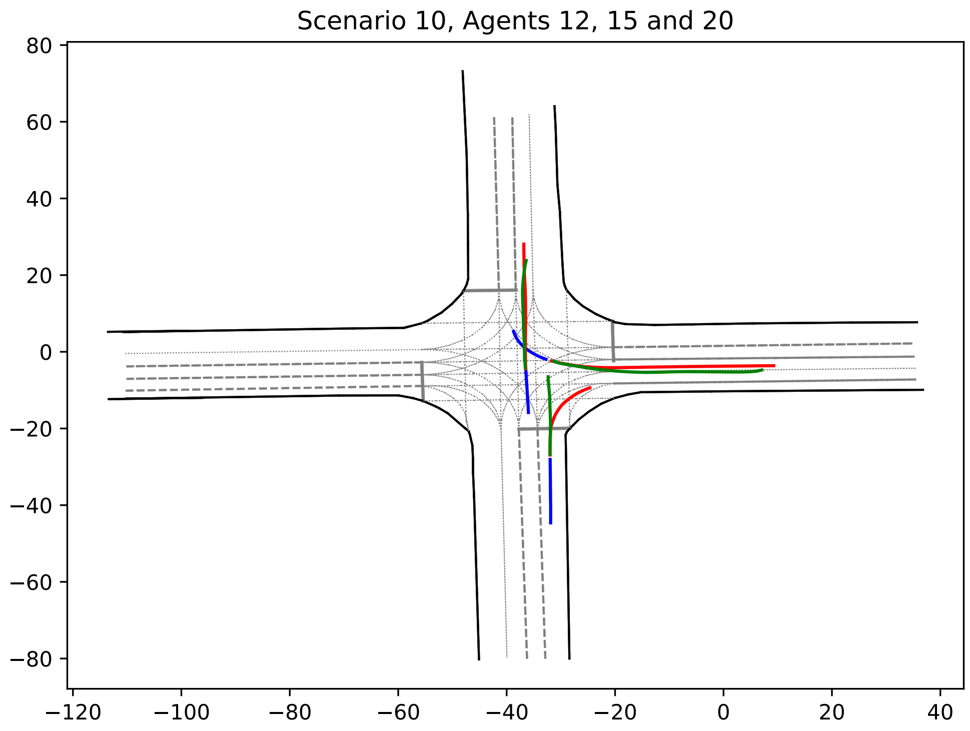

Figure 5: Predictions using the model_map

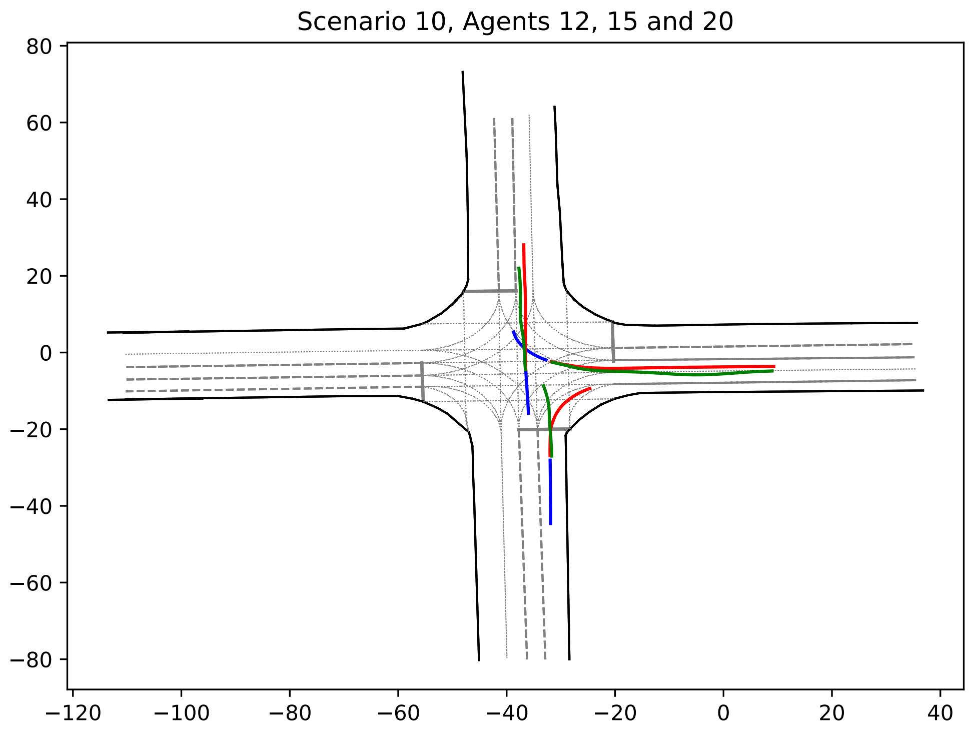

Figure 6: Predictions using the model_nomap

The models model_nomap, model_map and model_gatmap have 217 090, 301 123 and 336 386 total parameters, respectively.

model_nomap takes around 50 ms to predict, which is possible to compute in realtime.

model_map takes around 150 ms, which might not be acceptable for realtime systems.

model_gatmap takes around 0.5 s, which is probably unacceptable in this scenario.

constant_velocity is near instant to compute, but probably has unacceptable results.

3.1

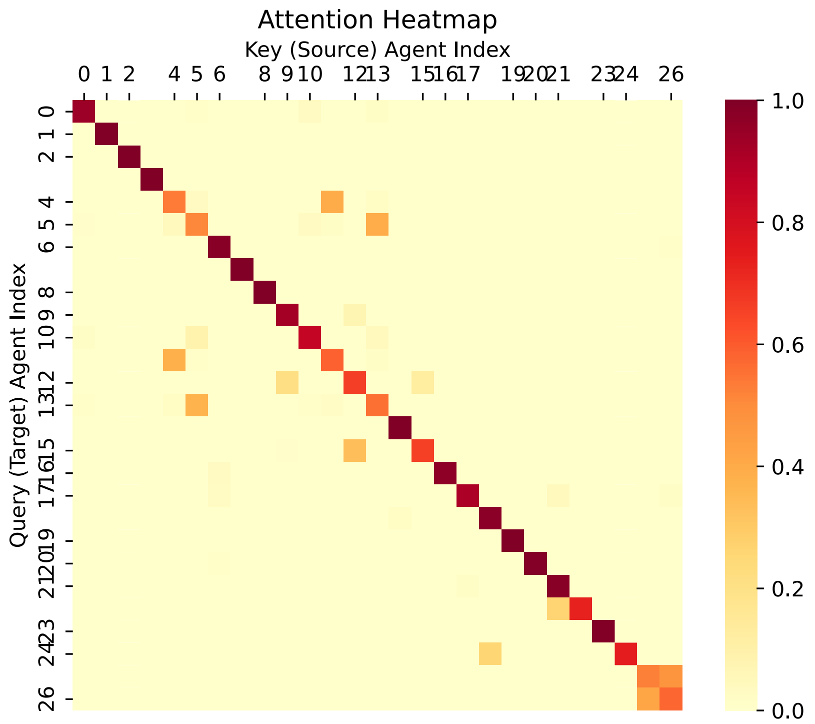

Figure 7: Heatmap for scenario 2.

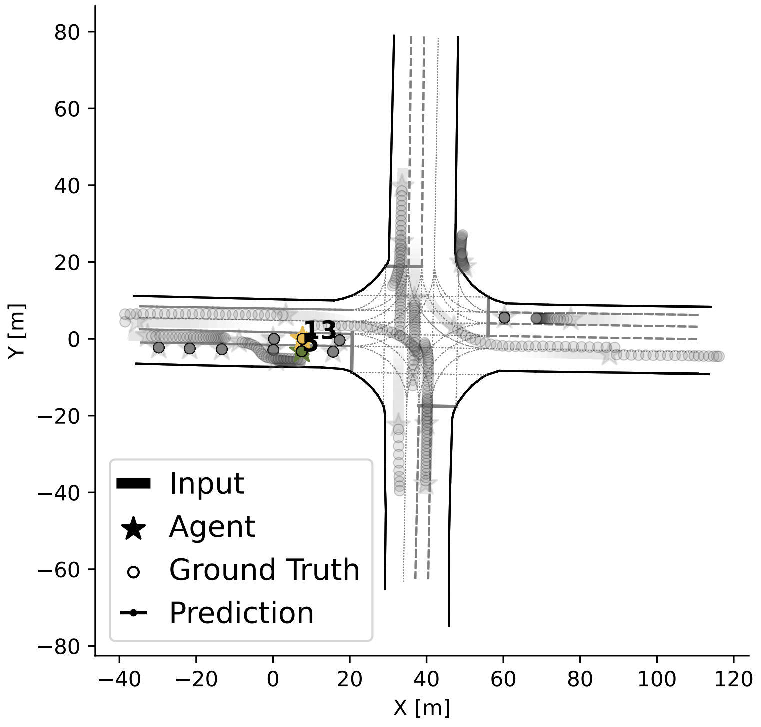

Figure 8: Scenario 2, with agents 5 and 13 highlighted.

In scenario 2, agent 5 and 13 both pay attention to each other and are physically close. It makes sense that agents would take into account more the agents that affect its predictions (i.e. can collide with it), and disregard the more distant agents.

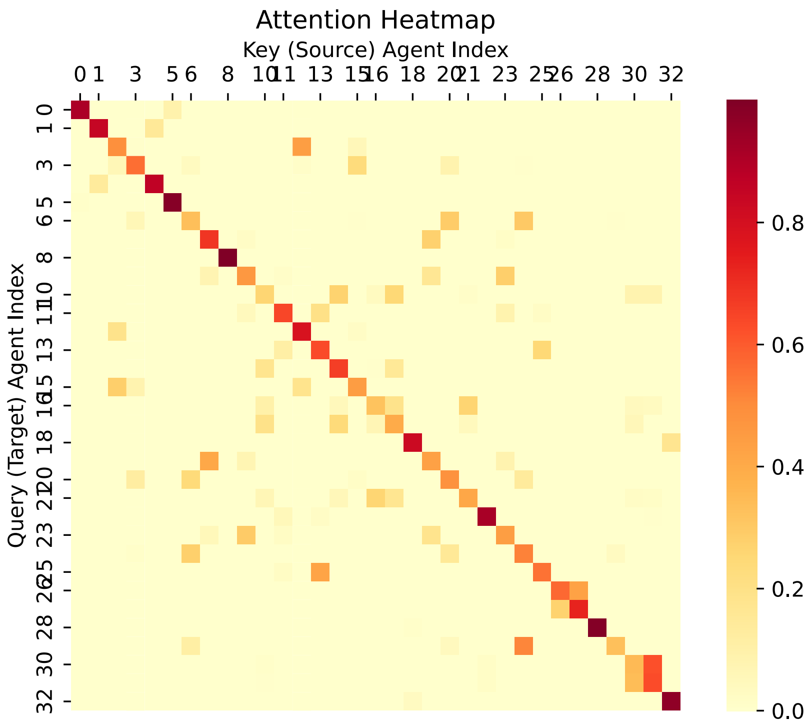

Figure 9: Heatmap for scenario 233.

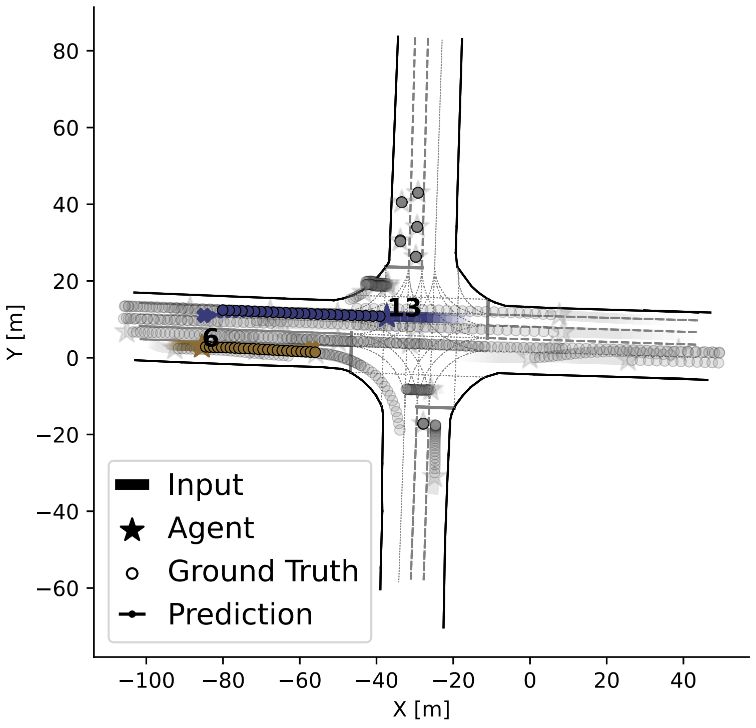

Figure 10: Scenario 2, with agents 6 and 13 highlighted.

In scenario 23, agent 13 seems to pay attention to agent 6, but the reciprocal is not that strong. They are physically close, so it is surprising that agent 6 does not pay as much attention to agent 13.

4.1

Table 1: Performance metrics

```
Constant velocity:: ADE: 1.89 m | FDE: 4.99 m | Miss Rate: 0.51
ML model, no map :: ADE: 1.20 m | FDE: 3.30 m | Miss Rate: 0.46
ML model, map    :: ADE: 1.09 m | FDE: 2.96 m | Miss Rate: 0.44
ML model, GAT map:: ADE: 1.05 m | FDE: 2.86 m | Miss Rate: 0.43
```

Regarding the performance metrics:

- ADE measures how far off the prediction was from the true position of the agent, on average.
- FDE only measures how far off the agents predicted position in the end was compared to the true end position.
- ADE seems like a more useful metric, since it contains information about the full scenario, while FDE only measures the final prediction.

The performance of the four methods is as expected from previous exercises. When comparing the ADE for the different methods it is clear the `constant_velocity` model differs quite significantly from the other three methods.

Using ML with many parameters causes the computation to take significantly more time. As shown in 2.4, already `model_map` could be too slow for realtime applications.

Our results indicate that having map-based information makes predictions better. Moving from no map to map decreases ADE by 0.11 m, and FDE by 1.69 m.

4. Discussion

Q-learning is a model-free method that can work in an unknown environment through exploration. It does not require previous knowledge to learn the characteristics and instead does so by interacting with it. By varying the greedy factor ε the exploration can be controlled. Increasing ε increases and encourages the exploration. Value iteration is a model-based method and therefore requires previous knowledge of the environment to compute the results.

The `gat_map` prediction model provides the most accurate prediction and follows the target path closely, except for when sharp turn occur after the observed path. However, for use in real time it would not be the best option since it takes 0.5s to predict. In real time the best model would be the `no_map` model, which is done in 50ms.