Submitters:

Ben Ganon 318731007

Omri Ben Hemo 313255242

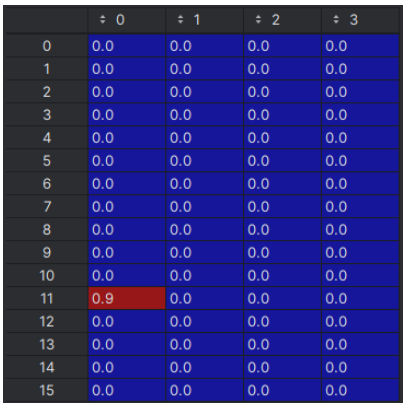
RDL Exercise 1

Section 1

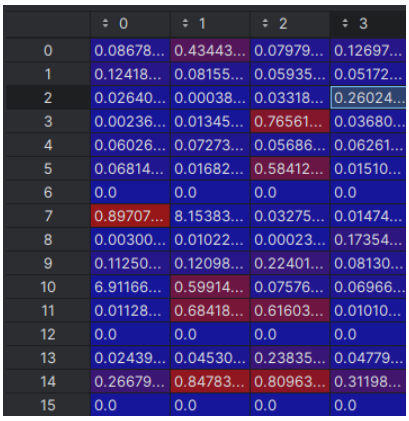
1. Methods such as Value-Iteration cannot be implemented in environments in which the dynamics are unknown or too complicated to calculate because the Value-Iteration algorithm iterates over all states and evaluates them, meaning that we are attempting to evaluate many states which take too long or are impossible to calculate, meaning the algorithm would never converge on an optimal policy.
2. Model free methods resolve the problem above by iterating over episodes and these episodes' states instead of attempting to pass all states, making the number of states seen finite (and calculatable). This ensures convergence is possible
3. The difference between SARSA and Q-Learning is that while SARSA is on-policy, Q-learning is not.
   What this means is that while SARSA learns and improves from the current policy, Q-learning can approximate the optimal policy. Via $max_a\ Q(S', a)$ in Q-learning we can greedily 'guess' what the optimal policy would choose and 'aim' for that.
4. Choosing an action with $\varepsilon - greedy$ sampling means we have a chance to take non-optimal pathways, meaning we can balance exploration and exploitation.
   Always taking the greedy option means maxing out exploitation, which can miss some paths to a better reward later on.

Final Solution:

- Final Parameters:
  - Learning rate: 0.9
  - Disctount: 0.9
  - Greedy epsilon initially: 0.5
  - Greedy epsilon minimum: 0.1
  - Greedy epsilon decay: 0.999
  - Epochs: 5000
- Q-table values:
  - 500 epochs:

    - 

	0	1	2	3
0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0
11	0.9	0.0	0.0	0.0
12	0.0	0.0	0.0	0.0
13	0.0	0.0	0.0	0.0
14	0.0	0.0	0.0	0.0
15	0.0	0.0	0.0	0.0

  - 2000 epochs:
    - 

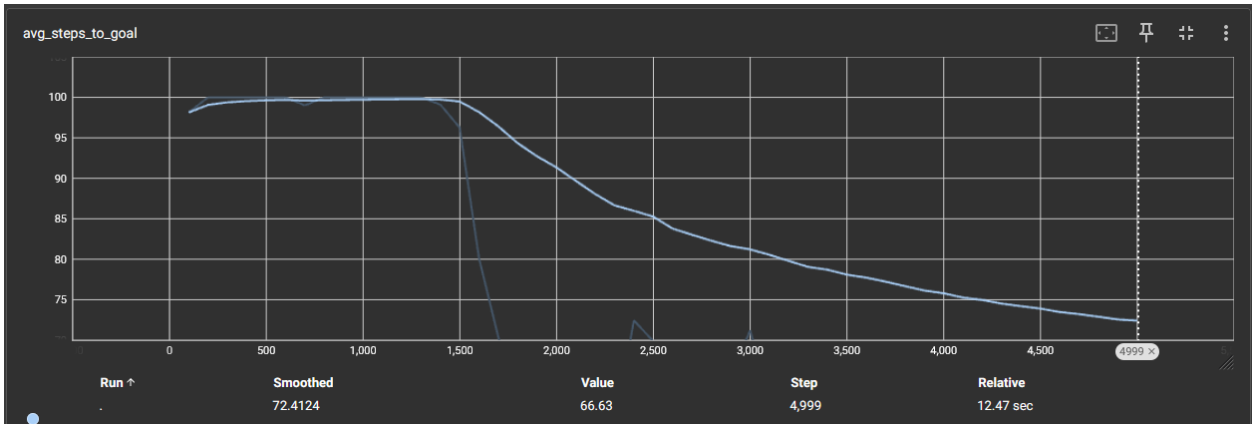
	0	1	2	3
0	0.08678...	0.43443...	0.07979...	0.12697...
1	0.12418...	0.08155...	0.05935...	0.05172...
2	0.02640...	0.00038...	0.03318...	0.26024...
3	0.00236...	0.01345...	0.76561...	0.03680...
4	0.06026...	0.07273...	0.05686...	0.06261...
5	0.06814...	0.01682...	0.58412...	0.01510...
6	0.0	0.0	0.0	0.0
7	0.89707...	8.15383...	0.03275...	0.01474...
8	0.00300...	0.01022...	0.00023...	0.17354...
9	0.11250...	0.12098...	0.22401...	0.08130...
10	6.91166...	0.59914...	0.07576...	0.06966...
11	0.01128...	0.68418...	0.61603...	0.01010...
12	0.0	0.0	0.0	0.0
13	0.02439...	0.04530...	0.23835...	0.04779...
14	0.26679...	0.84783...	0.80963...	0.31198...
15	0.0	0.0	0.0	0.0

  - Final table (5000 epochs):
    - 

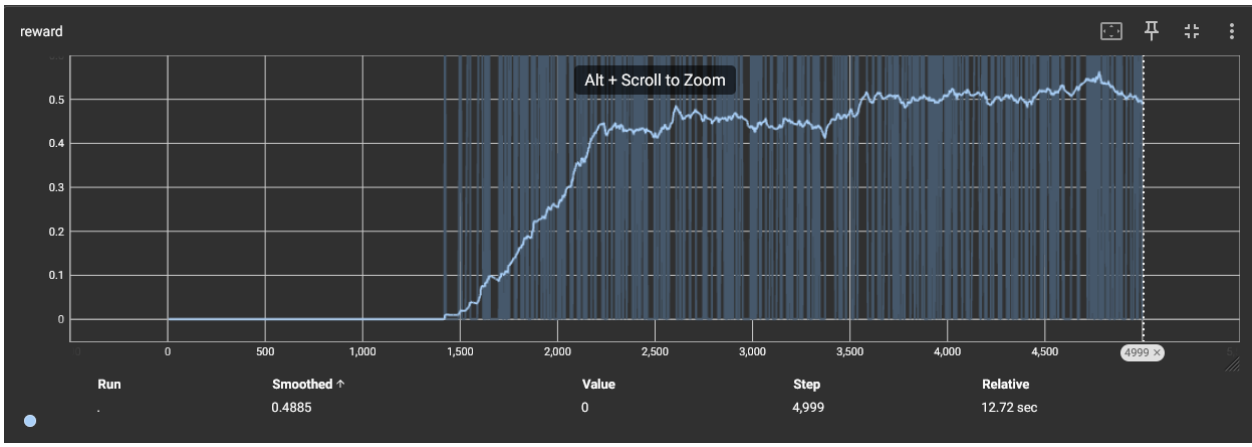
	0	1	2	3
0	0.11077...	0.09486...	0.14694...	0.111731...
1	0.14543...	0.08879...	0.08544...	0.09214...
2	0.02445...	0.00019...	0.04699...	0.40362...
3	0.26064...	0.13047...	0.50312...	0.15905...
4	0.08734...	0.08587...	0.08307...	0.09019...
5	0.111116...	0.08642...	0.01805...	0.01595...
6	0.0	0.0	0.0	0.0
7	0.42850...	0.04335...	0.61283...	0.02114...
8	0.00813...	0.00019...	0.00490...	0.07980...
9	0.11108...	0.13216...	0.50093...	0.11893...
10	0.14537...	0.88086...	0.65137...	0.00796...
11	0.46600...	0.18197...	0.96166...	0.64865...
12	0.0	0.0	0.0	0.0
13	0.01979...	0.04418...	0.67628...	0.04035...
14	0.23238...	0.55138...	0.32270...	0.20432...
15	0.0	0.0	0.0	0.0

- Plots:

- Average steps to goal:



- Reward per episode:



Section 2:

1. We sample replays in random order to ensure we "refresh" the model's memory with all kinds of states and actions, ensuring it does not forget previous learning segments.
2. The older set of weights allows us to converge the "value" network, while the target network is the same. This ensures the improvement of the model is "aiming" correctly (like the example in the lecture – we want to chase where the target will be in x steps and not where it is now.

Final parameters used in this section:

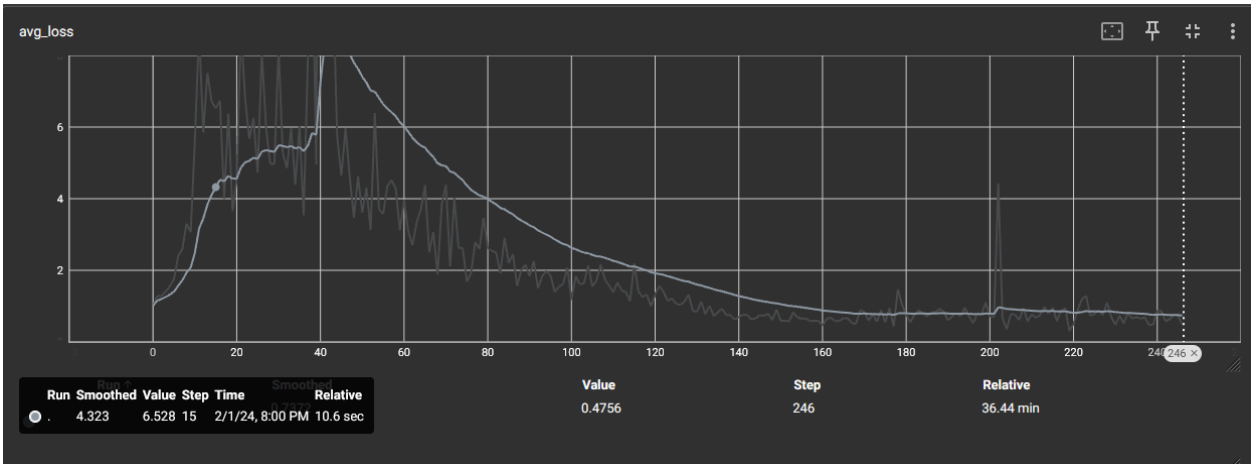
- Experience replay size – 10000 (we found that larger sizes did not help the model converge, and suspect that this is because when the replay queue contained very old replays, the model would "remember" bad iterations from long ago)
- Minibatch size – 16
- C-step - 6 (we updated the target model every 6 steps)
- Learning Rate – 0.005
- Discount – 0.95

- Optimizer – Adam (we tried sgd and rmsprop extensively, but found adam converged solidly)
- Greedy epsilon – 1
- Greedy epsilon decay rate – 0.9
- Minimum greedy epsilon – 0.01
- NN architecture – we settled on 3 layers of size [32, 32, 32]
  (we tried 5 layers but that network converged slower)

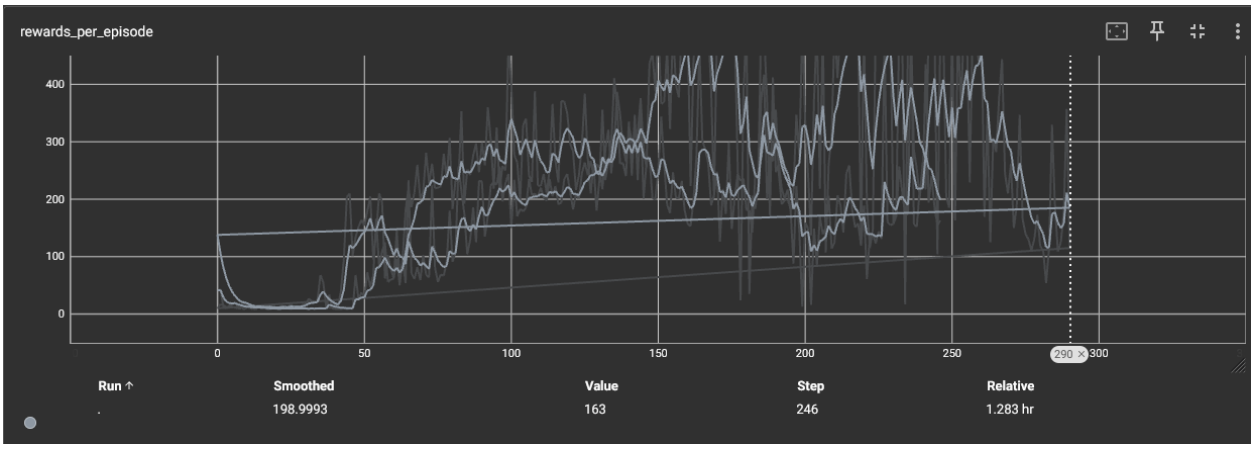
HP that appear to affect the model's convergence most – network architecture, learning rate, epsilon and epsilon decay, and the C-step updating.

We could not accomplish the desired results (time constraints) but it appears the model was on the way to converging:

Average loss:



Rewards per episode:



Section 3:

For this section, we chose to improve the network by training the agent with double DQN networks.

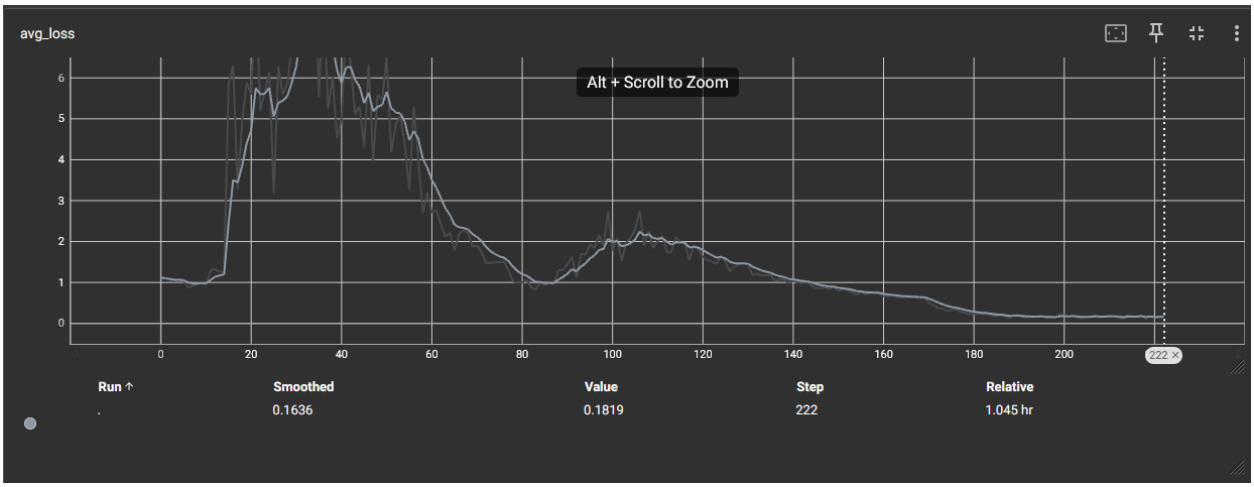
The network that chose the action was chosen randomly, then that network would be improved.

We didn't use the C-step update in this part since we saw that it did not help the models converge, opting for a more direct approach.

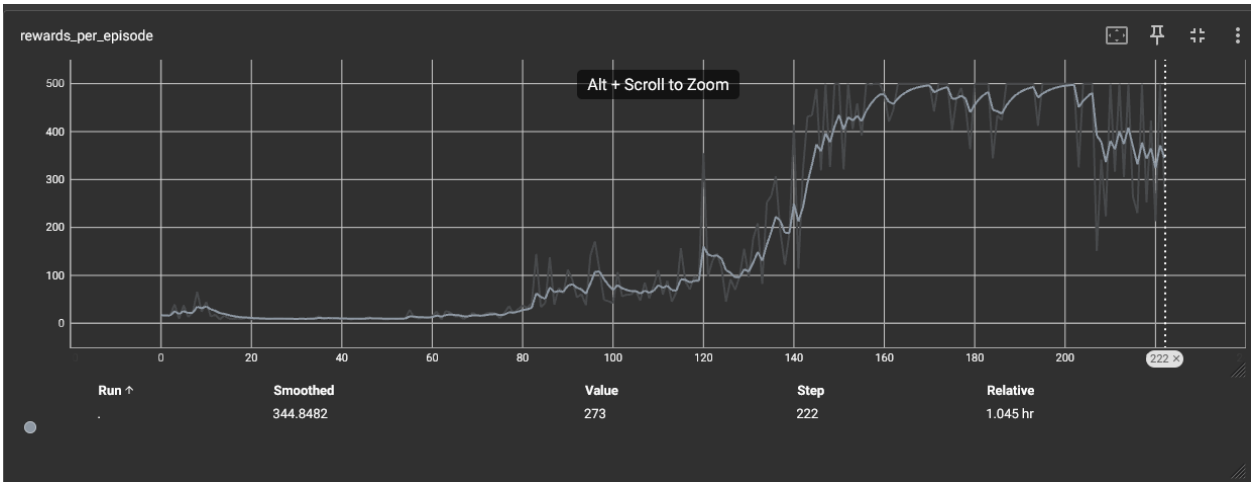
Final parameters used in this section:

- Experience replay size – 10000 (we found that larger sizes did not help the model converge, and suspect that this is because when the replay queue contained very old replays, the model would "remember" bad iterations from long ago)
- Minibatch size – 16
- Learning Rate – 0.01
- Discount – 0.95
- Optimizer – Adam (we tried sgd and rmsprop extensively, but found adam converged solidly)
- Greedy epsilon – 1
- Greedy epsilon decay rate – 0.9
- Minimum greedy epsilon – 0.01
- NN architecture – we settled on 3 layers of size [32, 32, 32]
  (we tried 5 layers but that network converged slower) for both networks

Average Loss:



Average rewards per episode:



It appears that this method (Double DQN) makes the convergence more robust (“smoother”) and allows it to keep a high average reward for longer (as seen in the plots)

Instructions to run the models:

Run the appropriate python file (section_2.py, section_3.py) with argument “train” or “test” to train or test the models.

They are configured not to render – if you wish to do so change the DO_RENDER variable to True.