CSE2530 Computational Intelligence

Reinforcement Learning

Yoon Hwan Jeong, Oskar Lorek,
Ethan Keller, Andrzej Rubio Bizcaino

April 2, 2021

0 INTRODUCTION

The report elaborates on the implementation of the Reinforcement Learning Algorithm for the robot collecting groceries as discussed across assignments of the *CSE2530 Computational Intelligence* course. The aim of this algorithm, in relation to the robot scenario, is to enable the robot to learn a route through (supermarket) maze by setting rewards at the desired locations. We will look into the version of Reinforcement Learning being **Q-learning**, which is a model-free, off-policy RL algorithm based on updating the function $Q : S \times A \rightarrow \mathbb{R}$, where S is the set of states and A is a set of possible actions [1].

1 DEVELOPMENT

1.1

`getEGreedyAction` returns an action by doing either uniform random selection with ε probability or greedy selection with $(1 - \varepsilon)$ probability. When it does greedy selection, an action with the highest action value is selected. However, since `getValidActions` in `Maze` returns an ordered list of actions, the same action will be chosen repeatedly if there are multiple actions with the highest action value. This is an issue in the early stage of the learning when the agent has not learned anything yet. In order to counter this, we shuffle the list of actions returned by `getValidActions`. By doing so, we allow the agent to choose a different action even if it has not learned anything yet.

1.2

The agent's cycle goes as follows. We first select the action that the agent is going to perform. In our situation this action is the direction in which the robot should take a step in the maze. This decision is made in a greedy way explained in the previous answer. Basically, it greedily selects either a random action or the best action based on Q values (action values).

After the action selection, the agent will perform the action (take a step in the maze). This results in the agent arriving in a new state. We now update the Q value (action value). This is the 'learning' part. In this moment in time the 'Q learning' is not yet implemented so we explain this 'learning' later.

The Q values are now updated. We now continue the agent's cycle by checking if he has arrived at the destination in the maze. In case he did, we reset the agent to place him back at the start location in the maze.

1.3

The cycle explained in the previous section is repeated until the agent reaches a predefined, maximum number of steps.

1.4

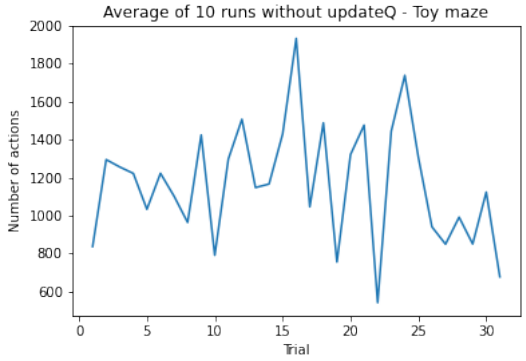

Figure 1: Average of 10 runs for toy maze without learning

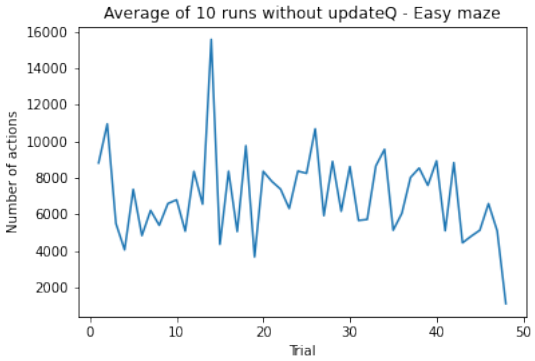

Figure 2: Average of 10 runs for easy maze without learning

1 and 2 show average number of actions in 10 runs without learning. The x-axis refers to the trial in each run and the y-axis refers to the average number of steps in each trial. Since the agent learns nothing, both `getRandomAction` and `getBestAction` act as a random selection. Therefore, the number of steps the agent takes per trial does not decrease and this is evident from the fluctuating graphs.

1.5

The update rule we used is the following:

$$Q(s,a)_{new} = Q(s,a)_{old} + \alpha(r + \gamma Q_{max}(s', a_{max}) - Q(s,a)_{old})$$

We are using a Q-Learning strategy which is off-policy meaning that the updates are not based on the actions that have been chosen. Instead we will update the action values based on the best possible next action $Q_{max}(s', a_{max})$. s' is the state that the agent is in after performing his latest action. α and γ are respectively the learning rate parameter and the discount factor parameter. The learning rate helps us tune the speed at which we are learning. Too large or too small learning rates can lead to local optima so this parameter must be tuned accordingly. The discount factor parameter is a value between 0 and 1 which allows us to stop the sum of rewards of going to infinity in situations with continuously recurring tasks.

1.6

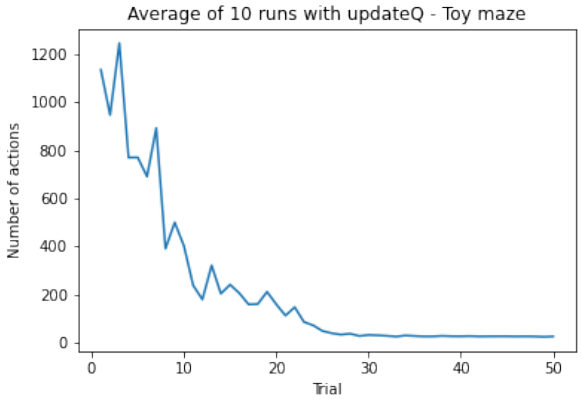

Figure 3: Average of 10 runs for toy maze with learning

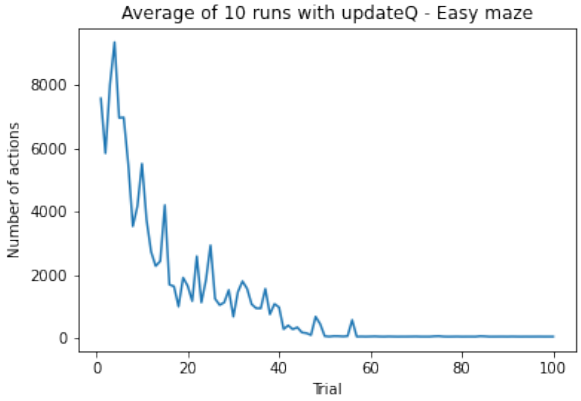

Figure 4: Average of 10 runs for easy maze with learning

3 and 4 show average number of actions in 10 runs with learning. The x-axis refers to the trial in each run and the y-axis refers to the average number of steps in each trial. The stopping criterion was increased to 300,000 steps for the easy maze as 30,000 steps were not enough for the agent to learn the easy maze. After taking an action, Q is updated and this affects subsequent calls to `getBestAction`. As the agent does more trials, the updated Q values lets the agent exploit the best path found so far and further explore from that path to find new best paths. This can be seen in the graphs as the number of steps decreases and becomes stable in the later trials. This indicates that the agent has indeed learned the mazes.

2 TRAINING

2.1

To see the change in algorithm behaviour due to different values of parameters, we should have a reference of a training session with the default parameters value. On the graph below we have shown the transparent run of 10 training trials and the average of number of steps per trails of the training in red.

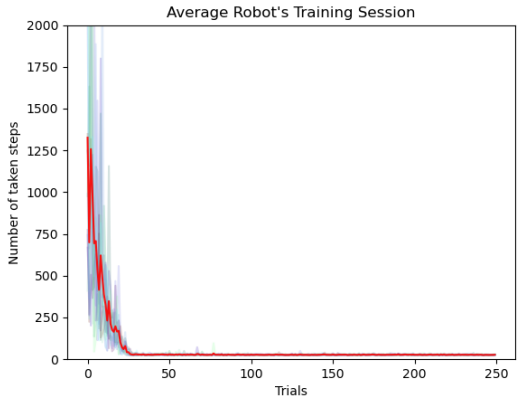

Figure 5: Robot's Training under default conditions: $\epsilon = 0.1$, $\alpha = 0.7$, $\gamma = 0.9$

As we can see the robot starts with taking a lot of steps/actions until it reaches the reward during first trials. At first the robot is completely unaware of the maze's environment and reward location, therefore it takes its steps randomly and gather information by updating Q-value. However, over time the robot develops a well functioning action policy which leads to the convergence to same value for all training trials.

Let's start with the investigation of the impact of epsilon on the algorithm training. Epsilon determines the probability of exploration, which deals with probability of the robot taking a 'random' action. Thus, $1 - \epsilon$ stands for probability of exploitation of the algorithm, determining the probability of our robot taking a greedy action. On the graph below we have presented an average of 10 learning sessions with different values of ϵ.

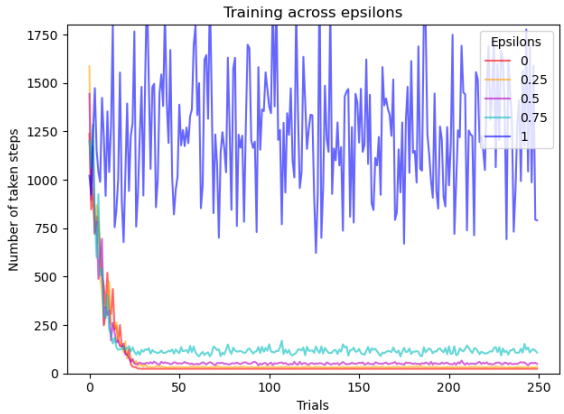

Figure 6: Robot's Training with different epsilons (exploration rate)

As we can see the impact of ϵ is significant in the algorithm training. With $\epsilon = 1$ it can be observed that the learning do not converge to a stable result. This is because with that value the algorithm will always take the random action taking, each trial, a lot of steps before reaching the reward. For all values of $epsilon < 1$ we see that the training curves look similarly to the run with the default values, however with different stable number of actions for each epsilon. We see the tendency of training with smaller values of ϵ have better (lower) convergence. This is because of the maze with a signle reward, which match the expectation of more greedy setup performing better than the more random training sessions.

2.2

Similarly as with ϵ, we can investigate the impact of different values of learning rates α on the training of the Q-learning algorithm. The parameter α directly impacts the Q function with $\alpha = 0$ meaning that there is no update to the Q function, and $\alpha = 1$ leading to dropping of all previous policy and rely completely on new estimation of Q function. On the graph 7, we can see the impact of α on the robot's learning.

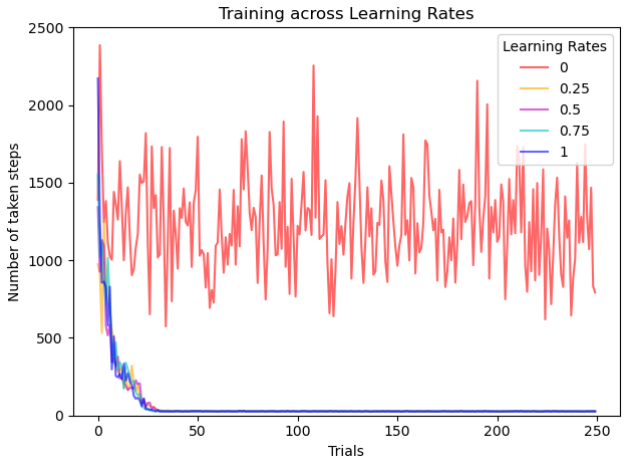

Figure 7: Robot's Training with different Learning Rates

As we can see for $\alpha = 0$ we obtain similar results to once setting the training setup with $\epsilon = 1$ (no convergence - fluctuating around 1250 number of steps/actions). Despite these reasons, the conclusion is the same, being the complete randomness of the algorithm throughout the training. In case of $\alpha = 0$ the algorithm has no chance to 'gather the information', by updating the Q function, which leads to using the random approach for the whole training session. For other values of α, we observe convergence to the same value. By investigating the closeup of the Robot's Training in graph 8, we can observe that the training sessions with different α do not differ much in their convergence, however there is a slightly better performance of higher values α sessions at the very beginning of the training, which may indicate that such models are slightly better at finding generalization of the optimal policies. However, $\alpha = 1$ session converges a bit faster to an optimum, which once again shows that because of the simple maze environment and

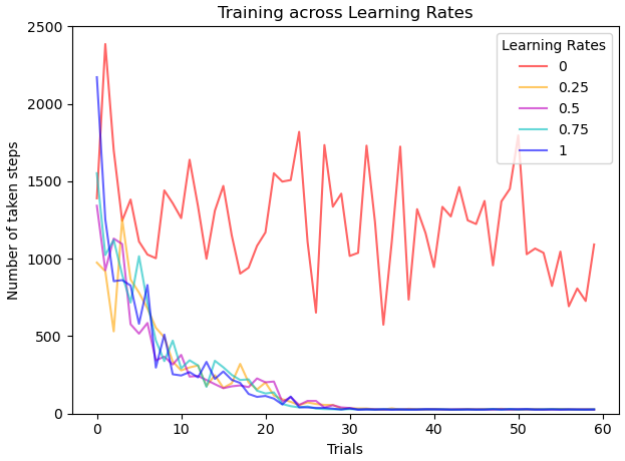

Figure 8: Robot's Training with different Learning Rates limited to 60 trials

single-reward the more actively-learning and greedy algorithms will perform better.

2.3

Although in our investigation we have recorded a dependence between parameters and the algorithm's performance it does not mean that such dependencies would be applicable to all RL problems, especially that patterns would not necessarily hold for different values of ϵ. There is a trade-off in usage of high and low values of the exploration parameter. Its final value should be based on the nature of the problem. In the example above, we have trained our robot in the 10x10 square environment with a single reward located on the opposite corner. This scenario is extremely simple and attains only one maximum at the reward location. In such setup, training sessions closer to $\epsilon = 0$ would perform best as it would easily localise the local (and in this scenario also global) maximum and take fewer random steps, which ultimately results in smaller number of actions needed to attain the maximum. However, in the bigger and more complex scenarios, and more importantly, with multiple rewards/penalties localised in the maze, we could observe that a bit higher valued ϵ sessions would obtain better results, as by taking some random steps the algorithm would tend to break over the determined path to some optimum and be able to explore a path to the global maximum reward. However it is important to notice that the value of ϵ should not tend to 1 as with that the algorithm would behave the same and take constantly random actions.

3 OPTIMIZATION

3.1

The above mentioned epsilon-greedy algorithm had problems with the following maze:

```
1 1 0 1 0 1 0 0 0 1
0 1 1 1 0 1 1 1 0 1
0 1 0 0 0 0 0 1 0 1
0 1 0 1 1 1 0 1 1 1
0 1 0 1 0 1 1 1 0 1
0 1 0 1 0 0 0 1 0 1
1 1 1 1 1 1 1 1 0 1
1 0 1 0 0 0 0 1 0 1
1 0 1 0 0 0 0 1 0 1
1 0 1 1 1 1 1 1 0 1
```

Figure 9: Toy maze

when there were two places as target locations (upper right with size of the reward 5 and bottom right with size of the reward 10). The problems that we encountered were that the algorithm (with previously discussed parameters), converged to the solution where the robot goes to the the upper right target location, despite it being with a lower reward (5 vs 10).

This can be explained by the layout of the Toy maze. It can be seen that from the only junction in the last column there is only 3 steps "up" required to reach the target location with the reward of 5, whereas to reach the bottom target location, the algorithm needs 6 steps "down".

As a consequence, the Agent is much more likely to explore the upper part of the maze rather than the bottom part, with larger reward. And as there is a reward (even though small) the algorithm tends to reinforce this sub-optimal behaviour.

3.2

One way to overcome this problem would be to set very high value for ϵ, by doing so the Agent would be more likely to explore the longer corridor in the maze. But there is a problem that arose when implementing this solution - as the probability of the Agent to go to the "greedy" state is low, the Agent, even after many iterations, performs moves close to random walk with equal probability for each action. Thus, as a result it often goes to the closer, upper target location.

What we really want to achieve is a situation in which the agent tend to explore more when it is uncertain about its surroundings and exploits more as he gains knowledge about the structure of the maze. This can be achieved via decaying value of ϵ after each trial. We have established the following equation for it:

$$\epsilon_{t+1} = \beta\epsilon_t$$

where t corresponds to the number of a trial and β lies in $[0, 1]$ and is defined as a decay rate of the ϵ.

There are other, more sophisticated approaches to formulate the update criteria for ϵ, that also take into account the change of the Q values after one trial. This can be seen in the algorithm proposed by Tokic[2]. One could also develop an update criteria based on simulated annealing[3].

We now set up the parameters to: $\alpha = 0.7, \gamma = 0.9, \epsilon = 1.0, \beta = 0.995$, and we examined after each trial on which target location the Agent ended. We repeated the procedure 100 times and plotted the graph of the percentage of Agents that end up on the 10 reward target location.

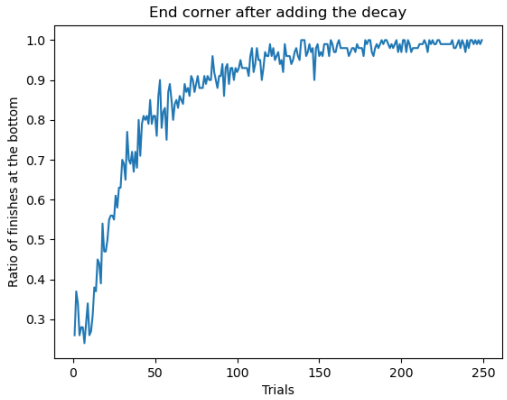

Figure 10: Ratio of robots going down (repeated 100 times) after adding ϵ decay

As can be seen from the graph above, we improved the algorithm by adding the decaying ϵ. Now the algorithm is less susceptible to convergence to the local optima that is not a global optima (for example: target location that is closer to the junction but has a lower reward)

3.3

Moreover, we investigated how the change of γ affects the optimal solution to go up for smaller reward, or to go down for the larger reward.

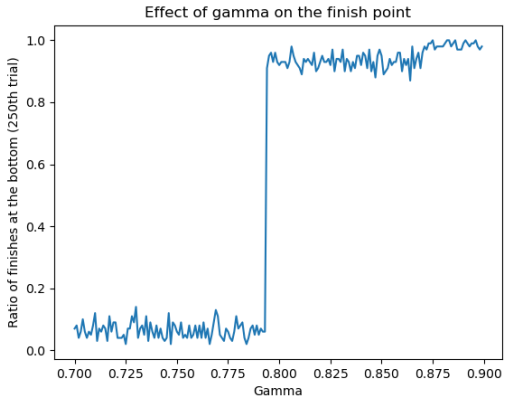

Figure 11: Behaviour of the algorithm with the change of γ

As can be seen from the graph above there is a very sudden change of the number of Agents that go up vs the number of Agents that go down around the value of γ equals 0.79. This can be explained by the fact that it is the value for which going up yields the same reward as going down. Below this value the discount factor makes the target location with smaller reward more optimal due to the fewer number of steps that have to be taken to reach it.

Consider the following equation:

$$Gain(decision) = reward \cdot (\gamma)^{n_{steps}}$$

Now we can find for which gamma the following holds:

$$Gain(up) = Gain(down)$$
$$5 \cdot \gamma^3 = 10 \cdot \gamma^6$$
$$\gamma = \frac{1}{\sqrt[3]{2}} \approx 0.7937$$

Which is in line with what we had observed in the graph.

We can even observe how very small change of γ affects the decision to go up or down:

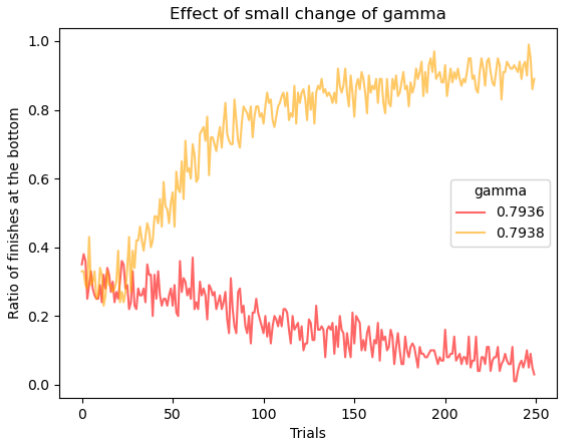

Figure 12: Small change of γ can drastically influence the decision

3.4 Conclusion

We developed a Reinforcement Learning Algorithm, which enables the robot to learn a route through maze (supermarket) by setting rewards at the desired locations. We implemented epsilon-greedy **Q-learning**, which is a model-free, off-policy RL algorithm. We extended the algorithm by adding a decay to the epsilon as the trials progress to mitigate the problem of more than one target locations with different rewards.

REFERENCES

[1] Richard S. Sutton. 1998. *Reinforcement learning : an introduction.* The MIT Press, Cambridge, MA.

[2] Michel Tokic and Günther Palm. 2011. Value-difference based exploration: adaptive control between epsilon-greedy and softmax. In *Annual conference on artificial intelligence*. Springer, 335–346.

[3] Peter JM Van Laarhoven and Emile HL Aarts. 1987. Simulated annealing. In *Simulated annealing: Theory and applications*. Springer, 7–15.