A Comparison of Q-learning and SARSA Algorithms to Solve OpenAI Taxi

Isabelle Kampono (S3918343) Zeid Hazboun (S4272722) Group 35

University of Groningen

1 Introduction

In an increasingly autonomous world, more and more algorithms are developed to train artificial intelligence to relieve humans of different workloads. One of the areas which are being developed is the autonomous vehicles field. In this project, we tackle the training of AI to be a taxi. OpenAI gym has a very helpful library that helped implement this, providing an interface to view the taxi and possible actions it can take. This environment is further elaborated upon in the environment specification section.

Reinforcement learning algorithms are a type of supervised machine learning algorithms that reward the correct behavior of agents and punish bad behavior. This is done by giving the agent positive or negative rewards, depending on the behavior shown. These types of algorithms are often implemented in situations where an agent has to traverse an environment and achieve a goal. In this scenario, we will use these algorithms to teach an AI to pick up a passenger, located in one of 4 possible spots, and drop them off at a location that is randomly placed in one of those 4 spots.

2 Environment

The environment is set as a 5 by 5 grid world that has 4 ”target” locations, namely: a red, a green, a yellow, and a blue square. At the start of the game, the passenger and building are randomly placed at two of these target locations, while the taxi will start at a random square on the grid. The game’s goal is to find the passenger, pick them up and drop them off at one of the target locations where the building is. An example of the environment can be seen in figure 1 where the building is on the red square and the person is on the yellow square.

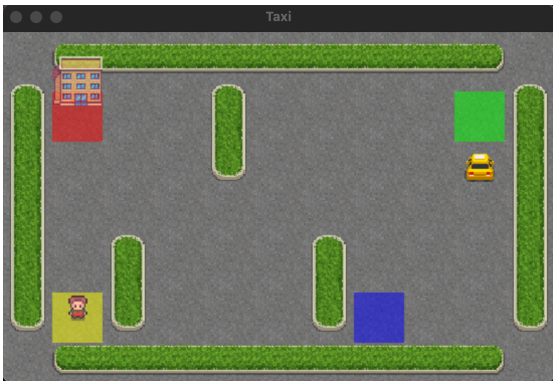


Figure 1: The environment layout

The environment will allow the agent to have 6 deterministic actions :

- move south
- move north
- move east
- move west
- pick up passenger
- drop-off passenger

Moreover, the library also has a built-in reward system for the agent as follows:

- -1 per step unless another reward is triggered.
- +20 for delivering the passenger to the right target location.
- -10 for executing "pickup" and "drop-off" actions at the wrong location.

Given these specifications, each episode will have 500 possible discrete states that include 25 possible taxi positions, 5 possible passenger positions including the position where the passenger is inside of the taxi) and 4 possible destination locations. An episode will automatically terminate once the agent drops off the passenger at the right location.

3 Algorithms

This paper will focus on the use of Q-learning and SARSA algorithms to tackle the taxi problem. We will compare both algorithms to a baseline algorithm that performs randomly in each step, while also using two exploration methods to further observe the performance of said algorithms.

3.1 Random

To have a better comparison of the performances that the algorithms obtain, we decided to include an agent that chooses a random action every time. This is done by randomly choosing one of the 6 actions within the action space in every step.

3.2 Q-Learning

Q-learning has risen to be a popular reinforcement learning algorithm in recent years. It is a model-free learning algorithm, which means that we let go of the transition probability and we assume that taking an action will change the state each time it is taken. The reason for its success is that it uses a table, referred to as the Q-table, that contains the valuation of state-action pairs. The Q-learning equation can be seen below, in equation 1.

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha[r_t + \gamma max_{a \in A} Q(s_{t+1}, a_t) - Q(s_t, a_t)] \tag{1}$$

- $Q(s_t, a_t)$: The current value of the state-action pair.
- α: The learning rate.
- r_t: The current reward.
- γ: The discount factor
- $max_{a \in A} Q(s_{t+1}, a_t)$: The action that returns the maximum reward for the next state.

After each instance of running the equation, the value of the state-action pair in the Q-table is updated. After a few iterations, the table converges into constant values. These values can then be used by the agent to decide which action to take.

3.3 State-Action-Reward-State-Action (SARSA)

Another algorithm we implemented is the SARSA algorithm. SARSA is an on-policy algorithm that improves the policy that it currently adopts by using an update rule. It is a model-free algorithm that also makes use of the Q-table to store utility values. However, instead of evaluating single state-action pairs like in Q-learning, it stores the trajectory of two state-action pairs in the Q-table. Meaning that it also takes into account the action a_1 taken at the next states_1

The SARSA update equation can be seen below. Most of the variables inside of it are the same variables used in theQ-learning update equation:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2}$$

where $Q(s_{t+1}, a_{t+1})$: is the value of the state-action pair that is taken at the next step.

3.4 Policies

For both the Q-learning and SARSA, we implemented two different policies and examined the differences between them. Each policy is explained below:

- ϵ-Greedy: Similar to the greedy policy discussed before, the algorithm chooses the action corresponding to the maximum $Q(s, a)$ value in the q-table. However, there is a probability ϵ that the agent chooses a random action. This allows the agent to explore and not get stuck in a local maximum. At the start of training, we start with an ϵ value of 1.0. As we train the model, the epsilon value decreases exponentially as shown by the equation below, where λ is the decay rate and t is the episode we are in.

$$\epsilon = 0.01 + (1.0 - 0.01) * \exp^{-\lambda t} \tag{3}$$

- Optimistic Initial Values (OIV) : this policy initializes the Q-table "optimistically", so instead of initializing the table using zeros, we use ones. This policy encourages exploration since the $argmax$ function in Q-learning's update equation always finds the action-state pair with the highest value. As for the SARSA algorithm, using optimistic initial values will inflate the utility values at the start of training which in turn, should make convergence faster.

4 Experiment Design

We began the experiment by visualizing the environment and using a few random actions to better understand the OpenAI library. We also explored how the values are returned by each function, as well as how the states/actions are represented in the program. After developing our understanding of the library, we began our implementation.

We started with the random algorithm, which was the simplest to implement due to its lack of hyperparameters. The only parameters we required were the number of episodes to run the algorithm for and the number of steps that the taxi can take. It was run for 5000 episodes, with a maximum of 200 steps for each episode. After the data was collected from the agent using the random algorithm, we ran the agents using the Q-learning algorithm as well as SARSA. We used the same number of episodes and steps as the random algorithm, but we introduced the alpha, gamma, and epsilon hyperparameters. We also performed hyperparameter tuning to determine which values are the best to use. More information on which values we used can be seen in the parameter tuning section. After testing the different algorithms with different hyperparameters, the optimal values for each algorithm were deduced. Then, the relationship between the values as well as the algorithms was examined using a two-way analysis of variance test (ANOVA). This test was chosen because we wanted to examine the interaction of both variables.

After finding the best hyperparameters, we tried out different exploration methods on both algorithms and ran a statistical analysis to examine the significance of the differences between algorithms and between exploration methods.

5 Results

In this section, we will introduce the results we obtained from the methods we discussed at the end of the previous section, starting with the first method.

5.1 Random algorithm

First, we run the random algorithm as a baseline to compare our other algorithms. Figure 2 shows the total reward per episode.

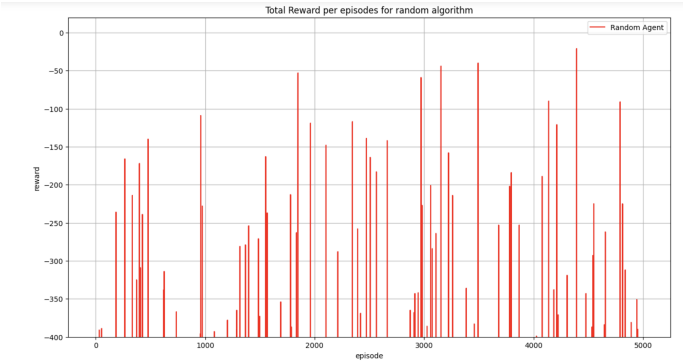


Figure 2: performance of random agent

Examining the graph, we can conclude that the random agent was inconsistent in its actions and there is no sign of any learning. While this result was expected, figure 2 provides us with a graph that we can use to prove our other agents are learning alongside our quantitative results. Some space can be seen between a few values. This is because we limited the graph to only show y values in the range of -400 to 20. This is because the algorithm would start with a very large negative reward and would make the plot rather difficult to view.

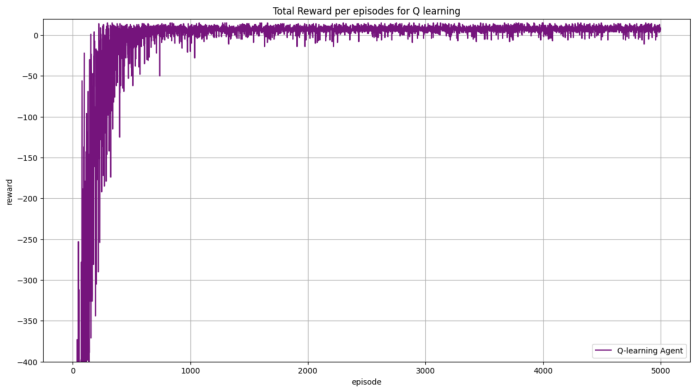
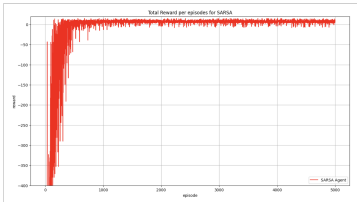
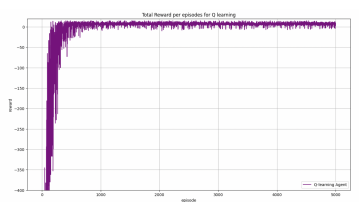


Figure 3: performance of Q-learning agent with $\epsilon = 0.9$



(a) best SARSA



(b) best Q learning

Figure 4: Performance of SARSA and Q learning with best hyper-paramters

5.2 Parameter Tuning

We decided to run both the Q-learning and SARSA algorithms with varying alpha α and gamma γ values to find the best-performing algorithm. We tried α values ranging from 0.1 to 0.3 and γ values ranging from 0.7 to 1.0. Tables 1 and 2 show the performance of the algorithms.

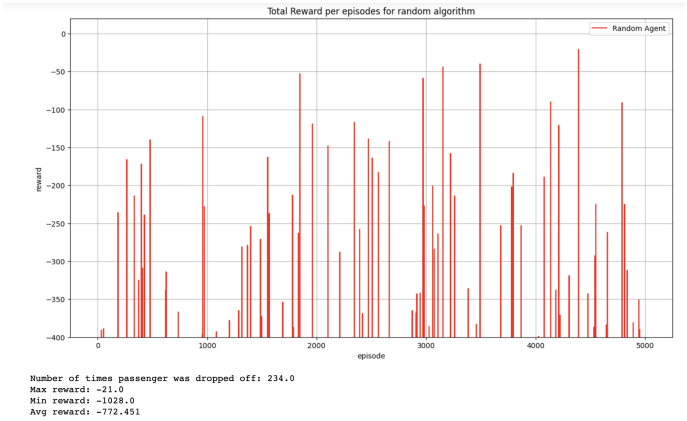


Figure 5: performance of random agent

Table 1: performance of Q learning agent with varying α and γ values

alpha	gamma	mean reward per episode
0.1	1.0	-20.2488
0.1	0.9	-21.128
0.1	0.8	-23.15
0.1	0.7	-25.6668
0.2	1.0	-12.281
0.2	0.9	-12.8268
0.2	0.8	-13.7972
0.2	0.7	-15.6524
0.3	1.0	-8.902
0.3	0.9	-10.4868
0.3	0.8	-9.7298
0.3	0.7	-11.3496

Tables 1 and 2 show that an alpha of 0.3 and a gamma of 1 produce the best-performing algorithm with an average reward of -8.9 for the Q-learning algorithm and -11 for the SARSA algorithm.

A two-way ANOVA was performed to analyze and compare the main effects of alpha and gamma and the interaction effect between alpha and gamma on the average reward per episode of both algo-

Table 2: performance of SARSA agent with varying α and γ values

alpha	gamma	mean reward per episode
0.1	1.0	-21.0454
0.1	0.9	-21.0988
0.1	0.8	-22.0042
0.1	0.7	-29.7494
0.2	1.0	-12.45
0.2	0.9	-14.0408
0.2	0.8	-46.3386
0.2	0.7	-84.6462
0.3	1.0	-11.3224
0.3	0.9	-17.34
0.3	0.8	-73.9702
0.3	0.7	-85.426

rithms. Simple main effects analysis showed that alpha did have a statistically significant effect on average rewards for the Q learning algorithm ($F(1) = 7.38, p = 5.16e - 06$) and for the SARSA algorithm($F(1) = 114.21, p = 0.0264$). It also showed that gamma did have a statistically significant effect on average rewards for both the Q learning algorithm ($F(1) = 7.95, p = 0.0225$) and the SARSA algorithm($F(1) = 34.1, p = 0.000386$). Moreover, the two-way ANOVA on the SARSA performance revealed that there was a statistically significant interaction effect between alpha and gamma ($p = 0.011713$)

5.3 Comparison of Algorithms

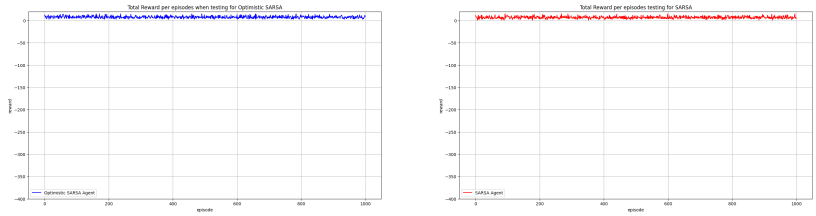
Before comparing the two algorithms together, we decided to create a random agent to use as a baseline and compare the SARSA and Q learning algorithms against it. An independent samples t-test between the average rewards per episode of the random algorithm ($\mu = -15.4, \sigma = 5.68$) and the average rewards per episode of the SARSA algorithm ($\mu = -10.8, \sigma = 92.1$) reveals that they are significantly different ($t(11) = -13.9, p = 0.007$). The same test was conducted on the average reward per episode of the random algorithm ($\mu = -15.4, \sigma = 5.68$)and the average reward per episode of the Q-learning algorithm ($\mu = -9.43, \sigma = 87.4$) which also found that the two are significantly different ($t(11) = -14.3, p = 0.0361$).

After fine-tuning the hyperparameters for the Q-learning and SARSA algorithms, we decided to compare the two best-performing runs from each algorithm. As seen in figure four, both algorithms perform quite well and converge after about 1000 epochs. The hyperparameter tuning process showed that both algorithms perform best with an alpha α of 0.3 and a gamma of γ, so we decided to compare those two performances. After collecting results and conducting an independent T-test between SARSA's mean reward per episode ($\mu = -10.8, \sigma = 92$) and Q-learning's mean reward per episode ($\mu = -9.44, \sigma = 87.44$), we found that the difference was not statistically significant ($p = 0.438$). This is as expected since the SARSA algorithm we implemented was an on-policy algorithm basing itself on the greedy policy, which means that it will behave as a greedy agent that uses argmax as a method to choose its actions at the start of the learning process.

5.4 Testing of Algorithms

Following the hyperparameter tuning and learning of the algorithms, we chose to test them using the optimal values of hyperparameters found ($\alpha = 0.3, \gamma = 1$). furthermore, we used a maximum number of 1000 episodes, each allowing a maximum of 200 steps. The graphs of all the algorithms can be found in figure 6.

Looking at the results, we can conclude that all the algorithms were successful when it came to testing the agents since there is no point where the agent's performance increases or decreases, meaning it is using the optimal policy. It was found that out of all four algorithms, the $\epsilon - greedy$ algorithm had the highest mean of 7.98. However, a one-way ANOVA was performed to compare the effect



(a) SARSA OIV test

(b) SARSA ϵ- greedy test

(c) Q-learning ϵ- greedy test

(d) Q-learning OIV test

Figure 6: Testing of SARSA and Q learning with best hyperparamters

of the exploration method/algorithm on rewards, and it was found that there was no significant effect ($F(3) = 1.03, p = 0.38$). Furthermore, testing showed that the algorithm performed the same as it did during the later stages of learning.

6 Discussion of Results

As seen from the discussion section above, it was observed that both SARSA and Q-learning are statistically significantly different from the random algorithm. However, there was no significant difference between the ϵ-greedy SARSA algorithm, ϵ-greedy Q-learning algorithm, OIV SARSA algorithm, and OIV Q-learning algorithm.

As expected, both ϵ-greedy algorithms perform quite similarly. This is because the $argmax$ function in Q-learning's equation causes it to have a similar action selection mechanism as the SARSA algorithm, i.e it will choose the action with the highest $Q(s, a)$ valuation. Even with an OIV policy, it was found that there was no significant difference between the Q-learning and SARSA algorithms since they still have similar action selection methods.

Moreover, it was observed that both algorithms perform best with an alpha of 0.3 and a gamma of 1.0. A gamma of 1.0 removes the concept of discounted rewards from our agents. We believe that the algorithm still performed well despite this due to the agent being given a reward of -1 per step. Thus, removing the discounted rewards would not negatively affect the agent

After carrying out this experiment, we have come up with some points of improvement that other researchers can implement. Firstly, we would recommend implementing more exploration strategies as this research only focuses on two. We would also recommend tuning the algorithm with a larger set of values to ensure the most optimal ones were found.