# CS 5033 - RL Project Report

**Airi Shimamura - 5033   Khoi Trinh - 5033**

## 1. Domain Introduction

CartPole v1 is a game environment provided in the Gym package from OpenAI. In this environment, there is a pole, attached to a cart (hence the name CartPole). The cart moves along a friction-less track. Force is applied in the left and right direction of the cart. The goal of the game is to keep the pole upright for as long as possible. For each step taken, a +1 reward is given, including the termination step. The maximum points achievable in the game is 475. There are a few conditions that, if met, will end the game.

First, if the pole angle is greater than 12 degrees, the game ends.

Second, if the cart position is greater than 2.4 (or the center of the cart reaches either end of the display), the game ends.

Finally, if episode length is greater than 500, the game ends.

For the criteria related to the CartPole environment, the agent will play the game for a total of 10000 episodes, and in each episode, the last 100 steps will have their rewards recorded (if the number of steps is less than 100, then the average will be over however many steps was taken for that episode) in order to generate an average. If any of the failing conditions is met, a penalty of -10 is given, otherwise, a reward of +1 is given.

## 2. Background

### 2.1. Reinforcement Learning

The goal of reinforcement learning (RL) is not simply to play a game. Rather, the driving force is to create systems that can be adaptive in the real world (Arulkumaran et al., 2017).

The essence of RL is learning through interaction and reward-driven behavior. The RL agent interacts with the environment and the results of its actions; it can then adjust its behavior in response to the reward(s) received (Arulkumaran et al., 2017)

This trial-and-error behavior can be considered the root of RL.

Reinforcement learning has been used in many applications such as process control, dispatch management, robot control, game competition, etc. (Qiang & Zhongli, 2011). In this project, we will explore using reinforcement learning to play the CartPole game, using 2 algorithm: SARSA and Q-Learning.

### 2.2. SARSA

SARSA is an on-policy temporal difference learning algorithm (Sutton & Barto, 2018).

It is a popular reinforcement learning algorithm. At each time step, the agent selects an action according to its policy, observes the next state and reward, and updates the estimated value of the current state-action pair using a learning rate and discount factor. The algorithm maintains a Q-value function that estimates the expected future rewards for each state-action pair. The agent selects an action using an exploration-exploitation strategy and takes the selected action in the environment.

The update equation for SARSA learning is:

$$Q(s, a) \leftarrow Q(s, a) + \alpha * (R + \gamma * Q(s', a') - Q(s, a))$$

where $\alpha$ is the learning rate, and $\gamma$ is the discount factor for future reward. For any given state-action pair, a new state-action value is obtained with a small correction to the old state-action value (Graepel et al., 2004)

The behavior of using the next state and action to update the current state and action value gives rise to the name SARSA (State, Action, Reward, [next] State, [next] Action).

### 2.3. Q-Learning

Q-learning is a reinforcement learning algorithm that enables an agent to learn an optimal policy by observing and updating the estimated value of state-action pairs.

The agent selects an action and observes the next state and reward. Q-Learning is a very similar algorithm to SARSA learning. However, Q-learning uses an off-policy learning approach, updating the Q-value function using the maximum expected future reward (Sutton & Barto, 2018).

The update equation for Q-Learning is:

$$Q(s, a) \leftarrow Q(s, a) + \alpha * (R + \gamma * max_{a'} Q(s', a') - Q(s, a))$$

Notice that the update equation always uses the maximum

Q value to update the current state, action value.

In terms of convergence rate and stability, Q-learning performs better than SARSA in the CartPole problem as it allows to learn the optimal policy. However, though Q-learning is simpler and less expensive to implement, it can lead to overestimation of the action values and result in sub-optimal policies (Nagendra et al., 2017). Additionally, due to lack of robustness and slow convergence in some cases, it is challenging to use Q-learning in real-world applications (Manju & Punithavalli, 2011).

Also, besides epsilon greedy method, softmax exploration is also used as exploration strategy for Q-learning. It selects actions based on the probability determined by a softmax function. This strategy allows to balance exploration and exploitation of high-value actions, therefore even though it is hard to tune the temperature parameter, softmax performs better than the $\epsilon$ greedy exploration strategy (Tijsma et al., 2016).

## 3. Hypothesis

**Hypothesis 1**: We expect the Q Learning algorithm to perform better overall than the SARSA algorithm over 10000 episodes.

**Hypothesis 2**: For SARSA, we expect exponential decay of $\epsilon$ to perform the best, as in, have the highest average reward over 10000 episodes.

**Hypothesis 3**: For Q-Learning, we will implement a softmax policy, but we expect it to not perform as well as either of the $\epsilon$ decay methods.

## 4. Experiments Results and Analysis

### 4.1. Hypothesis 1

This comparison is done between Khoi's SARSA algorithm, and Airi's Q-Learning algorithm. Below is the graph of average reward for each algorithm, with $\epsilon$ kept static at 0.5

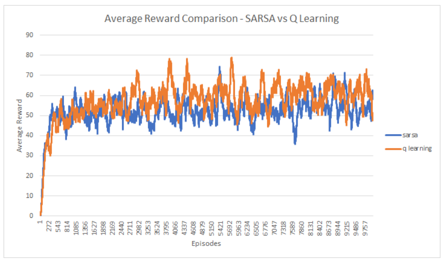

*Figure 1.* Comparison of SARSA and Q Learning

Looking at the figures, we can see that Q-Learning perform slightly better than SARSA, as the average reward stayed at a higher range for longer. Moreover, Q-Learning has an average reward of $57.44$ while SARSA's average is only $52.41$.

### 4.2. Hypothesis 2

We explored three different methods of $\epsilon$ decay. First, $\epsilon$ is kept static at a value of $0.5$. Second, $\epsilon$ followed the decaying function of $\epsilon = 0.5 * \frac{1}{no.of episodes}$. Finally, $\epsilon$ follows the exponential decaying function of $\epsilon = 0.5 * e^{-0.001*no.of episodes}$

For SARSA, comparing the three methods of $\epsilon$ decay, yields the following results:

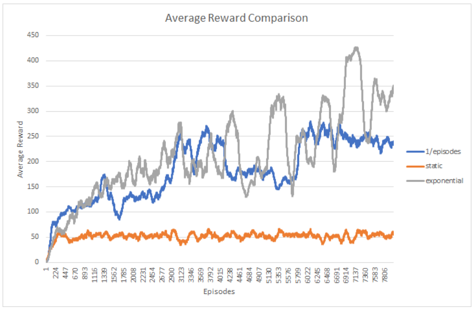

*Figure 2.* Average Reward of SARSA, Comparing Different $\epsilon$ Decay

Additionally, the mean and standard deviation for each method are as follows:

| Method | Mean | Standard Deviation |
|---|---|---|
| Static | 52.41 | 7.089 |
| 1/Episodes | 116.56 | 45.775 |
| Exponential | 192.32 | 68.383 |

*Table 1.* Mean of Average Reward and Standard Deviation

As these results show, for the SARSA algorithm, exponential decay of $\epsilon$ performed the best.

### 4.3. Hypothesis 3

For Q Learning, in addition to the three different methods of $\epsilon$ decay mentioned above, Airi implemented a softmax policy. It chooses an action $\alpha$ from all actions $A$ based on the probability determined by a softmax function, which is $\frac{e^{\frac{Q(a)}{\tau}}}{\sum_{b \in A} \cdot e^{\frac{Q(b)}{\tau}}}$. where $\tau$ is temperature parameter used to control the randomness of predictions and Airi sets $\tau = 0.1$ for this experiment.

Here are the comparison graph and table:

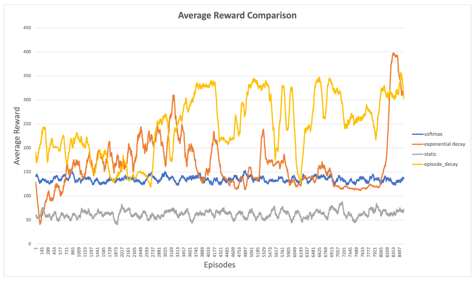

*Figure 3.* Average Reward of Q-Learning, Comparing $\epsilon$ Greedy and Softmax

| | Mean | Standard Deviation |
|---|---|---|
| Static | 61.35 | 7.6 |
| 1/Episodes | 194.52 | 68.62 |
| Exponential | 177.71 | 52.26 |
| Softmax | 133.88 | 6.94 |

*Table 2.* Mean of the Average Reward and Standard Deviation.

From the above table and graph, we can see that the mean of average reward for episode decay is higher than exponential decay, but exponential decay managed to reach a higher maximum mean of average reward. Furthermore, softmax did not perform well when compared to the two $\epsilon$ decay methods, but it performed better than keeping $\epsilon$ static at 0.5.

## 5. Methods Comparison and Related Work Review

Swagat Kumar implemented a Deep Q Network approach to this problem in 2020. Their DQN result is shown below:

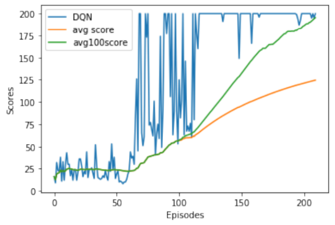

*Figure 4.* Performance of Kumar's DQN. Avg100score is the average of the last 100 episodes

It should be noted that Kumar used v0 of the CartPole environment, which has a much lower maximum reward threshold of 195. As such, their model were able to solve the problem in only 200 episodes (Kumar, 2020). Moreover, in the paper, Kumar also stated that DQN is much faster compared to the usual Q-Learning approach, which solved the problem in 300 episodes.

We expect the same results would apply to our v1 environment, if we were to implement a Deep Q Network approach.

Another comparison is from (Wang et al., 2013). Here, Wang used a different metrics and concluded that SARSA performs slightly better than Q-Learning, taking less time to train, as shown in this figure taken from the paper.

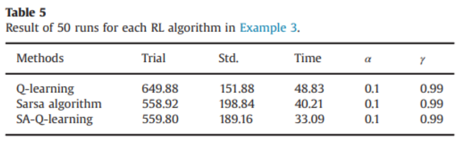

**Table 5**
Result of 50 runs for each RL algorithm in Example 3.

| Methods | Trial | Std. | Time | $\alpha$ | $\gamma$ |
|---|---|---|---|---|---|
| Q-learning | 649.88 | 151.88 | 48.83 | 0.1 | 0.99 |
| Sarsa algorithm | 558.92 | 198.84 | 40.21 | 0.1 | 0.99 |
| SA-Q-learning | 559.80 | 189.16 | 33.09 | 0.1 | 0.99 |

*Figure 5.* Performance of Wang's Algorithms

Compared to our implementations, where we have the average running time to go through 10000 episodes for Q Learning is 192.343 seconds, while SARSA takes 180.925 seconds to perform the same task. So, our results matched that of Wang's, if we used the same comparison metrics. However, because mean rewards is used as our metrics, we considered Q Learning to be the better algorithm.

## 6. Conclusion and Future Work

Throughout the course of this project, we were able to implement a SARSA and a Q-Learning algorithm that successfully played the CartPole game. As hypothesized, Q Learning performed slightly better than SARSA overall. Secondly, Exponential decay of $\epsilon$ yielded the largest average reward for SARSA. Thirdly, for Q Learning, softmax policy performed worse than either of the $\epsilon$ decay method.

In the future, we wished to be able to save our models to memory, and be able to recall them in a fresh environment without having to go through the training steps again. Moreover, we also wanted to implement eligibility traces with backwards view of TD($\lambda$) to see how it compares. Additionally, we would like to implement a DQN to compare to Kumar's results in (Kumar, 2020). Finally, we would also like to implement some sort of custom reward function(s), as opposed to a fixed -10 or +1 reward, to see if learning time would improve.

Aside from the CartPole environment we used, there exists the Pendulum environment within the same Gym package with similar properties. We think this will provide an interesting challenge and comparison to solve.

## References

Arulkumaran, K., Deisenroth, M. P., Brundage, M. & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, *34(6)*, 26–38.

Graepel, T., Herbrich, R. & Gold, J. (2004). Learning to fight. In *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education* (pp. 193–200).

Kumar, S. (2020). Balancing a cartpole system with reinforcement learning – a tutorial.

Manju, S. & Punithavalli, M. (2011). An analysis of q-learning algorithms with strategies of reward function. *International Journal on Computer Science and Engineering*, *3(2)*, 814–820.

Nagendra, S., Podila, N., Ugarakhod, R. & George, K. (2017). Comparison of reinforcement learning algorithms applied to the cart-pole problem. In *2017 international conference on advances in computing, communications and informatics (ICACCI)* (pp. 26–32).

Qiang, W. & Zhongli, Z. (2011). Reinforcement learning model, algorithms and its application. In *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)* (pp. 1143–1146).

Sutton, R. S. & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tijsma, A. D., Drugan, M. M. & Wiering, M. A. (2016). Comparing exploration strategies for q-learning in random stochastic mazes. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)* (pp. 1–8).

Wang, Y.-H., Li, T.-H. S. & Lin, C.-J. (2013). Backward q-learning: The combination of sarsa algorithm and q-learning. *Engineering Applications of Artificial Intelligence*, *26(9)*, 2184–2193.