Frozen Lake Path Finding Using Reinforced Learning

Chas Hamel
CS 5033 Spring 2024

1. Introduction

To demonstrate the implementation of Reinforced Learning (RL), the Frozen Lake Environment from OpenAI's Gymnasium[1] is used. With an observation space of an 8x8 grid, the user has 63 possible movement positions not including the starting state. Using Q-Learning and Monte Carlo algorithms, the models are trained so the Frozen Lake character can traverse over the lake to the Goal Position without falling into Ice Pockets, **Figure 1**, using singularly Left, Right, Up, and Down movements per each action. During model training, the model works to maximize the reward received amounting to one reward per successful traverse through the observation space.

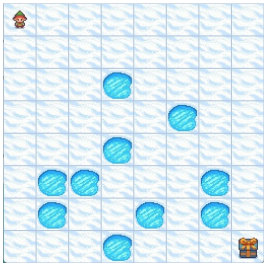

Figure 1. Frozen Lake Environment

Using the iterative update formula for the optimal action-value function $Q^*(s, a)$, we find the expected cumulative reward for each movement the Frozen Lake character takes as they traverse the Frozen Lake.

$$Q^*(s, a) = Q(s, a) + \alpha(r + \gamma max_a Q(s', a')) \text{ where,}$$

$Q(s, a)$ is the est. action-value function for (s, a),
α is the learning rate,
r is the immediate reward for taking action a in state s,
γ is the discount factor, and
$Q(s', a')$ is the max action-value function for (s', a')

For the Monte Carlo algorithm, the iterative update formula updates the action value function $Q(s, a)$ based on observed returns per episode.

$$Q(s, a) = \frac{\sum_{i=1}^{N(s,a)} G_i}{N(s,a)} \text{ where,}$$

$Q(s, a)$ is the est. action-value function for (s, a),
$N(s, a)$ is the frequency of action a, is used in state s, and
G_i is the obs. return after action a in state s for time i.

2. Hypotheses

Using the Total Rewards Gained, Rewards per 100 Attempts, and Algorithm Runtime as the quality metrics, I hypothesize that:

1. The Q-Learning Model will maximize the Average Rewards per 100 attempts and ultimately the Total Rewards Gained over n attempts when compared to the Monte Carlo Algorithm.
2. The Q-Learning Model will have a lower runtime over n attempts when compared to the Monte Carlo Algorithm.

3. Learning Experiments

The selection reasoning and processes for establishing the Q-Learning and Monte Carlo algorithms is discussed below. For both algorithms, $attempts(n) = 15,000$ and greedy policy $epsilon = 1.0$ with a decay rate of 0.0001 for all learning and evaluation processes. The Frozen Lake Environment has the option of creating a "Slippery" Lake, where the agent does not always follow the assigned action. For this project, this feature is deactivated.

Q-Learning The first algorithm, Q-Learning, is selected due to the ability to find the optimal policy for the presented Markov Decision Process (MDP). For the Frozen Lake environment, Q-Learning learns the optimal policy by choosing to Explore new directions or to Exploit known directions in an attempt to prioritize long term or short term rewards, respectively.

As described in the introduction, the Q-Learning algorithm uses a learning rate, α, or the probability of the agent choosing to explore or exploit known Q-values as well as a discount factor, γ, where future rewards are discounted as to drive the agent to either prioritize immediate or future rewards.

To find the optimal combination of these hyperparameters, where the total rewards were maximized, a sweep of all possible combinations of $\alpha \in [0.1, 0.9]$ and $\gamma \in [0.1, 0.9]$ was performed, resulting in 81 individual evaluations. Due to

the high number of parameter combinations tested, the top 10 results are visualized in **Figure 2**.

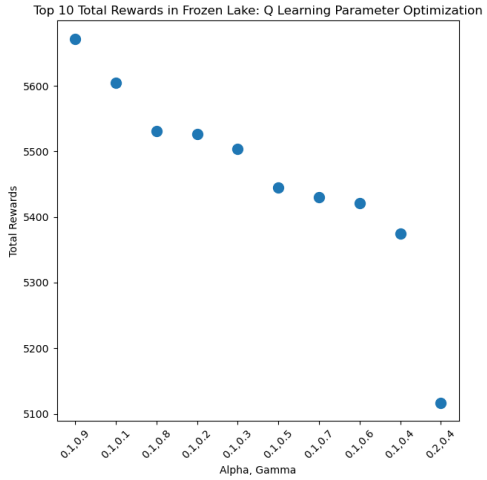

Figure 2. Q-Learning Top 10 Hyperparameters

As $\alpha = 0.1$ and $\gamma = 0.9$ equates in the overall maximum of 5,672 total rewards / 15,000 attempts.

Monte Carlo The second algorithm, Monte Carlo, is selected as the algorithm is Model-Free and the Frozen Lake agent will interact with the environment in discrete episodes where a high amount of exploration is required.

Compared to Q-Learning where α and γ are driving hyperparameters, the Monte Carlo algorithm uses γ to discount the rewards within G_i.

To find the value of γ which produces the maximum number of rewards, a sweep of possible values of $\gamma \in [0.1, 0.9]$ is performed. **Figure 3** shows the result of rewards achieved for each value of γ, where $\gamma = 0.8$ results in 6,956 total rewards / 15,000 attempts.

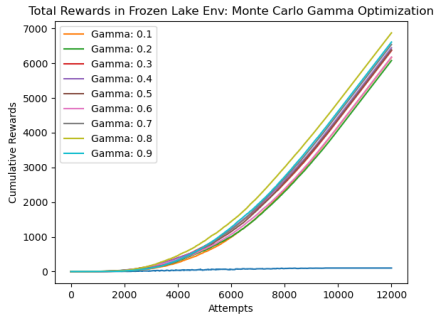

Figure 3. Monte Carlo Rewards Per Selected γ

4. Algorithm Evaluation

With the optimal hyperparameters identified for each algorithm, 10 repetitions of the Q-Learning algorithm with $\alpha = 0.1$ and $\gamma = 0.9$ as well as 10 repetitions of the Monte Carlo algorithm with $\gamma = 0.8$ are run with $n = 15,000$ attempts each and ϵ=1.0 with a decay rate of 0.0001.

After completion, the running sum of rewards produces 5,605 average rewards for Q-Learning and 6,705 average rewards for Monte Carlo methods. **Figure 4** below reflects the reward attainment for both algorithms while **Figure 5** reflects the average rewards per 100 attempts within one text cycle. As depicted in both Figures, the Monte Carlo algorithm achieves the optimal policy earlier than Q-Learning and can exploit this policy to maximize rewards through the learning process.

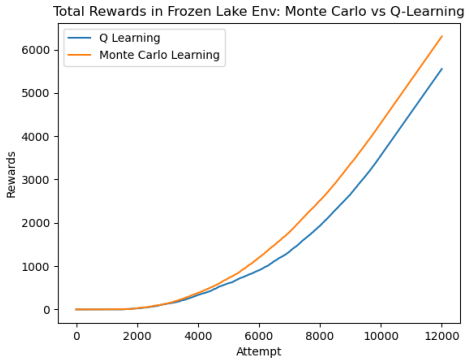

Figure 4. Reward Summation: Q-Learning vs Monte Carlo

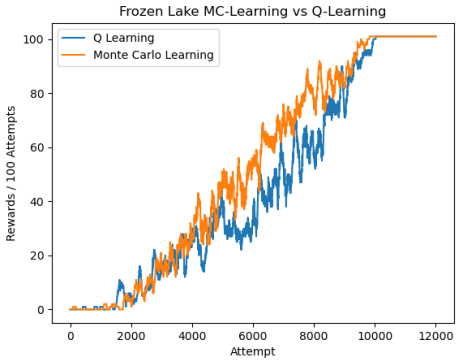

Figure 5. Rewards / 100 attempts: Q-Learning vs Monte Carlo

From a efficiency and statistical perspective, **Table 1** reflects the average runtime for each algorithm during these 10 cycles while **Table 2** articulates the average and standard deviation rewards for each.

Table 1. Algorithm Average Runtime

Algorithm	Average(sec)	St. Dev(sec)
Q-Learning	5.45	0.14
Monte Carlo	46.64	3.56

Table 2. Algorithm Reward Statistics

Algorithm	Average Rewards	Award St. Dev
Q-Learning	5605.9	103.42
Monte Carlo	6705.3	174.11

Although the Monte Carlo algorithm outperformed Q-Learning from a total rewards perspective, the average runtime and standard deviation for this algorithm significantly under performed when compared to Q-Learning. In the Frozen Lake environment, the Monte Carlo runtime of 46.64 seconds is excessively high for 15,000 attempts when compared to Q-Learning. This could be due to the Monte Carlo algorithm's attempt to learn the complete environment when building the optimal policy.

5. Summary

In conclusion, the hypothesis that the Q-Learning algorithm would outperform Monte Carlo methods in total rewards accumulated was proven to be incorrect. The Monte Carlo algorithm effectively maps the entire Frozen Lake environment to build a complete understanding of an optimal policy and then continuously exploits this policy much earlier than the Q-Learning algorithm. In terms of algorithm efficiency, the hypothesis that the Q-Learning algorithm would achieve a lower runtime was found to be correct, with an average runtime roughly 8x more quick, when compared to Monte Carlo **(Table 1)**.

6. Literature Review and Future Work

Monte Carlo Tree Search

As described by Cameron Browne et al.(C.B. Browne et al., 2012), the Monte Carlo Tree Search (MCTS) method is used for "finding optimal decisions in a given domain by taking random samples in the decision space and building a search tree according to the results." By building this success tree, the MCTS algorithm shows signs of extreme efficiency, in comparison to the Monte Carlo methods implemented in this project. Additionally, the MCTS attempts to use a balanced approach to exploration vs. exploitation by expanding the search tree to find best nodes, such as Upper Confidence Bounds in comparison to sampling random states to estimate the optimized function. As the Monte Carlo algorithm in this project showed poor runtime performance in comparison to Q-Learning, it would be interesting to continue to investigate and implement MCTS algorithms.

Q-Learning Auxiliary Tasks

In the Q-Learning paper "Reinforcement Learning with Unsupervised Axiliary Tasks" by Max Jaderberg et al., (Jaderberg et al., 2016), Jaderberg describes auxiliary reward tasks for grid work type environments, such as the UNREAL agent. By using auxiliary rewards we can persuade the agent that certain moves in the environment are more or less a priority to the overall task, such as by negatively rewarding for backtracking within the grid space similar to focusing on the immediate reward. Giving the example of using UNREAL agent in the Labyrinth (Mnih et al., 2016), the UNREAL agent "achieves 87% of expert human-normalized score, compared to 54% with A3C."

Applied Use of Monte Carlo Learning

Moving away from grid world, or game theory implementations, Carlos E. Murillo-Sanchez discusses an applied approach of Reinforced Learning in Energy Systems (Murillo-Sanches, et al., 2021). Due to the difficulties found in controlling supply of energy in smart grids, Monte Carlo methods and stochastic modeling helps to optimize energy consumption in smart grids by "dynamically adjusting energy generation...based on fluctuating demand and supply conditions." Through the complex, and real world, application of Monte Carlo algorithms described in this work, we can move beyond game theory towards applied applications more rapidly.

7. References

1. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J.,and Zaremba, W. (2016). OpenAI Gym. arXiv preprint arXiv:1606.01540.

2. C. B. Browne et al., "A Survey of Monte Carlo Tree Search Methods," in IEEE Transactions on Computational Intelligence and AI in Games, vol. 4, no. 1, pp. 1-43, March 2012, doi: 10.1109/TCIAIG.2012.2186810.

3. Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D.,and Kavukcuoglu, K. (2016). Reinforcement Learning with Unsupervised Auxiliary Tasks. arXiv preprint arXiv:1611.05397.

4. Volodymyr Mnih, Adria‘ Puigdome‘nech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Proceedings of the 33rd International Conference on Machine Learning (ICML), pp. 1928–1937, 2016.

5. Murillo-Sánchez, C. E. (2021). A Review of Reinforcement Learning Applications in Energy Systems. Renewable and Sustainable Energy Reviews, 135, 110074. https://doi.org/10.1016/j.rser.2020.110074