

ANALYZING REINFORCEMENT LEARNING ALGORITHMS ON OPENAI GYM'S FROZEN LAKE

ARTIFICIAL INTELLIGENCE IN PROCESSES AND AUTOMATION

 **Kristóf Horváth**

Computer Science for Autonomous Systems
Eötvös Loránd University
Budapest, Egyetem tér 1-3, 1053
hkristof03@gmail.com



ABSTRACT

In this report I discuss three reinforcement learning control algorithms. Control algorithms estimate the optimal action-value function directly from experience, without a model of the environment's dynamics. The algorithms are differentiated by their offline and online learning nature and the later category is further divided to off-policy and to on-policy temporal-difference learning methods. I present the algorithms and show empirical results for their theoretical properties on OpenAI Gym's FrozenLake-v0 environment.

1 Introduction

In reinforcement learning there is an agent that interacts with its surrounding environment. Through the learning process the agent learns how to map situations to actions in order to maximize a numerical reward signal. For every action the agent takes the environment responds with a state and a reward, on which the agent decides on the next action and so on. A reinforcement learning system can be further divided into a policy, a value function and optionally a model of the environment. The policy defines the agent behaviour such as which action it takes at a given time. The value function defines the amount of reward that the agent can expect to accumulate from a specific state. The model allows inferences about the behavior of the environment. The finite sequence of the agent-environment interaction is called Finite Markov Decision Process. The goal of reinforcement learning algorithms is to find optimal policies such as the agent accumulates the maximum possible reward over time in different environments. OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms. The diversity of its environments provides the necessary conditions for researchers to evaluate algorithms based on a common benchmark. In this report the introduced algorithms and the ideas behind them serve the basis for more advanced algorithms capable of beating expert human players in Atari games[1, 2].

2 The FrozenLake-v0 environment

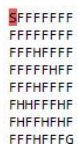


Figure 1:
FrozenLake-v0
8x8 map

The FrozenLake-v0 game is classified as a toy text environment, which helps beginners to quickly get started with reinforcement learning. The game has two different map sizes, a 4x4 and an 8x8 map. In this research, I analyze every algorithm's performance on the larger map on which it is more difficult for the agent to win. The map is comprised of 64 positions of a square, where each field belongs to one of the four different classes. There is one start and goal position denoted by S and G and located at the upper left and bottom right corner, respectively. The remaining two types of the fields are either a frozen spot or a hole on the ice, denoted by F and H, respectively. The purpose of the agent is to reach the goal from the start position while avoiding the holes. If the agent steps into a hole the game is over. The agent receives a reward of zero for every step which does not result in reaching the goal. If the agent reaches the goal, then the reward is one. The cardinality of the state and action space is 64 and 4, respectively.

The four actions the agent can decide on are moving left, right, up or down. The default behaviour of the environment presents a twist, because when the agent moves in any direction there is a probability for the agent to slip on the ice and thus arrive to a random position. Therefore, the best strategy for the agent is to avoid the holes from the maximum possible distance while moving towards the goal. This non-deterministic behaviour of the environment can be turned off. The map is shown on Figure 1.

3 Methods

In this section I present algorithms for control, where the values of state-action pairs are estimated.

3.1 Monte Carlo control with ϵ -soft policies for estimating $\pi \approx \pi_*$

Monte Carlo (MC) methods do not assume complete knowledge of the environment[3]. They estimate the values of state-action pairs by experience. They sample sequences of states, actions and rewards from their interaction with the environment. By learning from actual experience, they do not need knowledge about the dynamics of the environment (a model) a priori to reach an optimal behaviour. MC methods are defined for episodic tasks, where an episode eventually terminates independently of the agent's actions. An agent's actual experience consists of many episodes. These algorithms solve reinforcement learning problems by averaging the complete returns from the samples. The initial versions of MC algorithms are called offline methods, because they wait until the end of each episode to perform updates on the value functions. Without a model it is necessary to estimate the value of each action in order to suggest a policy, which determines the behaviour of the agent. Therefore MC methods estimate the optimal policy π_* . For this purpose, they evaluate the current policy by estimating $q_\pi(s,a)$, which is the expected return when starting in state s , taking action a , and thereafter following policy π . They consider a state action pair to be visited if state s was visited and action a was taken during an episode. First-visit MC method averages the returns following the first time the state s was visited and the action a was selected, while every-visit MC method averages all of the returns that followed each of the visits. The convergence of MC methods to the true values is quadratic if the visits to each-state action pair approach infinity. This can not be achieved if π is a deterministic policy, because in that case the agent always chooses the same action a from state s . Therefore, the requirement for the convergence of these methods is continuous exploration. Continuous exploration means that the probability to select every state-action pair needs to be assured. This is called exploring starts (ES) and it guarantees that as the number of the episodes converges to infinity, all state-action pairs will be visited infinite number of times.

In the MC ES algorithm we initialize the Q table with zeros (it can be random too) and we loop through the episodes. In each episode we maintain a set for the unique state action pairs that we visit during the episode. We also maintain an array where we store all of the state action pairs in order that we visit, and another array for the rewards that we receive. During the episode we follow a policy that supports the convergence condition i.e. maintaining exploration. For this purpose I chose $\epsilon - greedy$ policy with exponential decay for every algorithm I analyzed. Until an episode ends we generate the discrete time steps by taking actions according to $\epsilon - greedy$, and we store the resulting state and reward. When an episode ends we perform the updates on the Q table by iterating on the unique-state action pairs, summing up the rewards and discounting them appropriately that we received from the first occurrence of the state-action pair throughout the episode. The discounting of the future rewards for each of the unique state-action pairs is done according to (1),

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = R_{t+1} + \gamma G_{t+1} = \sum_{n=1}^{\infty} \gamma^n R_{t+n+1} \quad (1)$$

where $0 \leq \gamma \leq 1$ is the discount rate.

3.2 Q-learning: Off-policy TD Control

Temporal-difference (TD) learning methods also learn from experience without a model of the environment's dynamics[3]. TD methods bootstrap i.e. they perform updates based on learned estimates before an exact outcome is available. Algorithms in this category learn in an online fashion, on an incremental basis. While Monte Carlo methods wait until an episode ends, TD methods require one time step to wait, or in the case of an n-step TD method they have to wait until the first n-step long series of states, actions and rewards are available. TD methods are also guaranteed to converge to the true value functions. The condition that has to be fulfilled requires a sufficiently small step-size parameter. The probability of the convergence is 1 if the step-size parameter decreases according to the stochastic approximation conditions (2-3).

$$\sum_{n=1}^{\infty} \alpha_n(a) = \infty \quad (2)$$

$$\sum_{n=1}^{\infty} \alpha_n^2(a) < \infty \quad (3)$$

The two conditions guarantee the convergence to the true state-action values. The first condition prescribes large enough steps that eventually will overcome initial conditions and fluctuations, and the second condition guarantees the convergence by requiring the step size to become small enough as time passes. Q-learning is an off-policy TD control algorithm, which is defined by (4).

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \quad (4)$$

The reason why Q-learning is called off-policy because it updates its Q-values using the next state S_{t+1} and the greedy action a , and its estimation of the total discounted future reward is independent of the policy being followed. The learned action-value function Q directly approximates the optimal action-value function q_* [4].

In the Q-learning algorithm we initialize a step size parameter $\alpha \in (0, 1]$ and a small $\epsilon > 0$. The latter parameter is used to ensure and maintain exploration. It is decremented as time passes and supposedly Q converges to q_* , therefore the agent is able to exploit the environment more efficiently. This is called the trade-off between exploration and exploitation. The Q table should be initialized arbitrarily except for the terminal states, which should be 0. In each episode the algorithm first initializes the environment and then loops over it. At each iteration it chooses an action according to the $\epsilon - greedy$ policy. It means that with a certain probability the agent decides on a random action or it takes the best action from $Q(S, A)$, in which case we assume that it is the best possible choice for the agent to exploit the environment. Then it takes action A and observes the reward and next state R, S' respectively. Afterward, it performs the Q-learning update (4) and steps into the next state. The updates are performed in each episode until a terminal state is reached, when a new episode begins. The quantity $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$ is called TD target and the difference between the TD target and $Q(S, A)$ is called TD error. In the Q-learning update, the Q table is updated incrementally by moving in the direction of the TD error with small step sizes.

3.3 SARSA: On-policy TD Control

SARSA is considered an on-policy method because it updates its Q-values using the Q-value of the next state S_{t+1} and next action A_{t+1} that is the result of the current policy π [3][5]. The update that SARSA performs is defined by (5).

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \quad (5)$$

As an episode consists of alternating states and state-action pairs, this update is performed after every transition from a nonterminal state S_t . If the next state S_{t+1} is terminal, then $Q(S_{t+1}, A_{t+1})$ is zero by definition. The reason why the update rule is called SARSA is because it utilizes a quintuple of events that represents a transition from one state-action pair to the following one $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$. SARSA converges to the optimal policy π_* and action-value function $Q_*(S, A)$ with a probability of 1 if we guarantee that all state-action pairs are visited infinite number of times (continuous exploration) and if the policy converges to the greedy policy. This means that from the initial exploration phase the agent should decrease the probability of exploring and increase the probability of exploiting the environment as time passes.

SARSA learns the values of state-action pairs from the policy it follows, while Q-learning learns them by following the greedy policy. Both Q-learning and SARSA converge to the real value function if the convergence conditions (2)(3) are satisfied. Q-learning tends to converge slower, however it also has the capability to continue learning while changing policies. The difference between Q-learning and SARSA is how the Q table is updated after each action. SARSA uses the Q value which is the result of following the $\epsilon - greedy$ policy, while Q-learning uses the maximum Q value over all possible actions. In the update it might look like that Q-learning follows the greedy policy without exploration, however when it actually takes an action the action is derived from the $\epsilon - greedy$ policy.

4 Experiments

In the following section, I analyze the performance of the previously introduced algorithms on the FrozenLake-v0 8x8 problem. For every algorithm, I revisit the meanings and effects of the parameters that can be tuned to attain better

convergence properties. On Figure 2, several different configurations are shown to set up the $\epsilon - greedy$ policy. Every curve starts from 1, which means when the first episode begins the agent randomly chooses actions from the available action space. Most of the curves reach the interval of $[0.4, 0.2]$ around the 20k-th and 40k-th episodes, which means the agent takes a random action with a probability of 0.4 and it follows the greedy policy e.g. it selects the best action that is available with a probability of 0.6, for example.

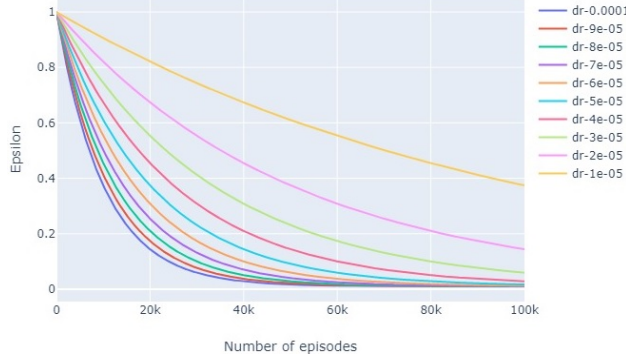


Figure 2: Illustration of different exponential decay rates for $\epsilon - greedy$ policy. With the help of the policy the agent starts with high probability of exploration. As the episodes pass, the agent shifts towards exploitation because it knows its environment better.

The results of the MC experiment on Figure 3 confirms the expectations that it needs a high probability of exploration for a relatively long period. It is an offline learning algorithm and it has to wait until the end of each episode to sample experience and average the results. Convergence requires continuous exploration, which need is further intensified by the random nature of the FrozenLake environment and its size. MC method requires a lucky series of random actions to get from the start position to the goal, which results in useful updates in the Q table.

The need for high initial exploration rate is reflected on Figure 4. The agent was only able to accumulate rewards over time for high initial exploration probabilities. There is a significant difference in the rate of convergence of $Q(S, A)$ to $Q_*(S, A)$ for small variations in the initial ϵ values. The exponential decay curves that were used for the experiment is shown on Figure 5 for different initial ϵ values.

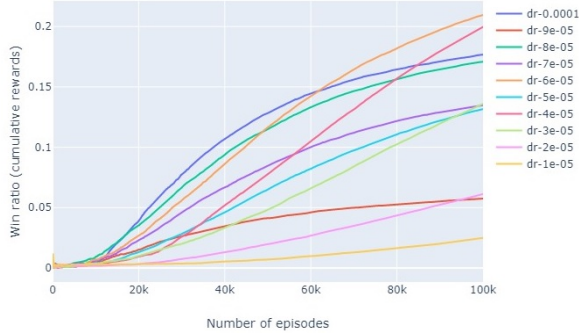


Figure 3: Monte Carlo method's convergence of $Q(S, A)$ to $Q_*(S, A)$ means increasing cumulative rewards over time. It is shown that the slower the decrease in the probability of exploration the better the results are.

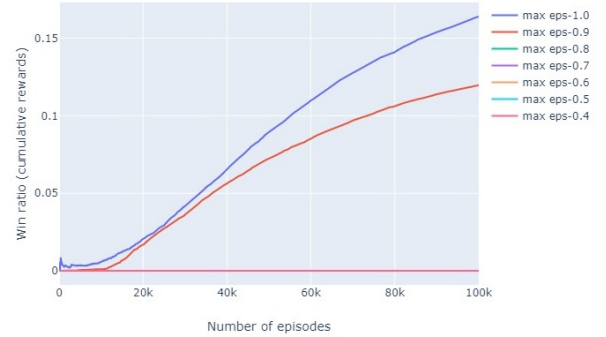


Figure 4: Cumulative rewards over time in the Monte Carlo experiment. Higher initial exploration probabilities result in better convergence of $Q(S, A)$ to $Q_*(S, A)$. For low initial exploration probabilities there is no convergence to the true values.

On-policy first-visit MC control method averages the appropriately discounted returns according to (1) for every unique state-action pair that is occurred in an episode. The returns are averaged from the first occurrence of a given unique state-action pair and with the resulting value the update is performed on $Q(S_t, A_t)$. Since the FrozenLake environment by default provides a reward of 0 for every non-terminal state and a reward of 1 for the final move if it ends in the goal, the Q table suffers from very small updates as γ decreases and as the state-action pair is many steps away from the goal. Figure 6 shows that the convergence of MC is satisfactory in the only case when the agent takes into account the future rewards at the maximum possible rate it can, when $\gamma = 1.0$.

Q-learning performance and the rate of convergence of $Q(S, A)$ to $Q_*(S, A)$ increases as the probability of exploration decreases at a higher rate. Q-learning updates the Q values as soon as the experience is available, therefore the convergence starts to improve much earlier than in the case of MC, and at a much higher rate as it is shown on Figure 7.

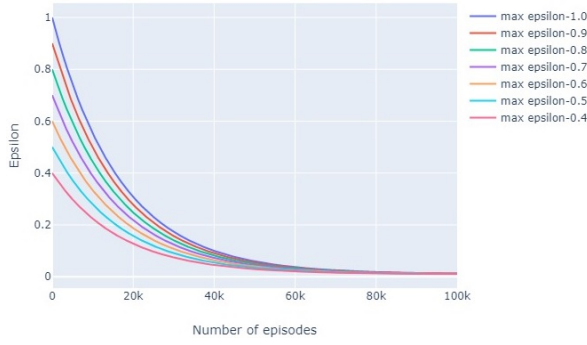


Figure 5: Exponential decay of the probability of exploration with different initial ϵ values and same decay rate of $6e^{-5}$.

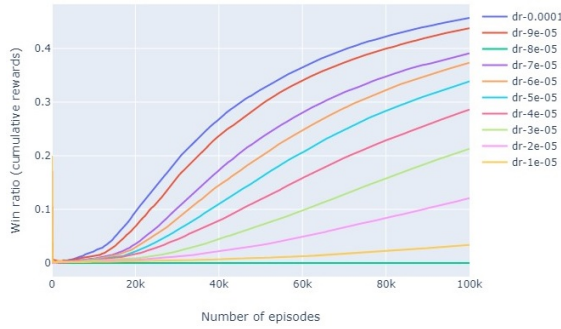


Figure 7: Q-learning is able to trade-off exploration against exploitation much earlier than MC. Higher decay rates and therefore the earlier decrease in the probability of exploration results in better convergence properties.

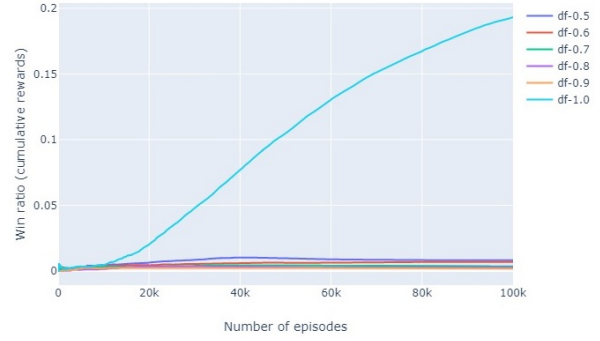


Figure 6: The effect of γ on the convergence of $Q(S, A)$ to $Q_*(S, A)$ in the Monte Carlo algorithm. Lower values than 1.0 significantly prevent the algorithm to converge, mainly because of the environment's reward system.

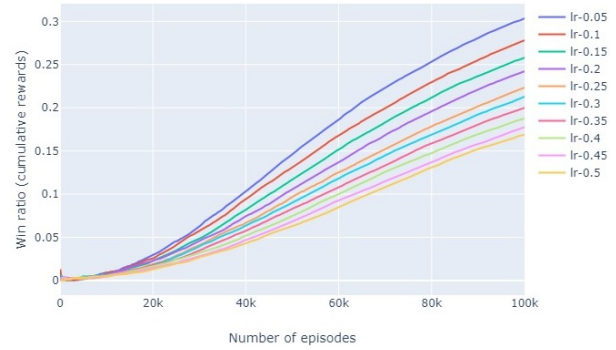


Figure 8: Q-learning presents better convergence properties towards the optimal policy π_* with smaller steps in the direction of the TD error.

The rate of convergence towards the true values increases for smaller step sizes in the direction the TD error. On Figure 8 it is shown that for smaller learning rates the convergence is faster. Q-learning presents an almost perfect positive correlation between the increasing γ discounting rates and the convergence rates to the optimal state-action values in the FrozenLake environment, shown on Figure 9. In the approximated optimal range for the values of γ , there is only a slight difference in the rate of convergence to the true values, as illustrated on Figure 10. As the policy π converges to the optimal policy π_* the cumulative rewards increase that the agent receives from the environment, as it is presented on the Figure 11. A window size of 1000 was used to calculate the moving average of the cumulative rewards. At the same time the length of the episodes converges to the interval of $[60, 70]$, where the best performing agent with a learning rate of 0.05 was able to reach a moving average of 0.5. At this point the agent wins every second game.

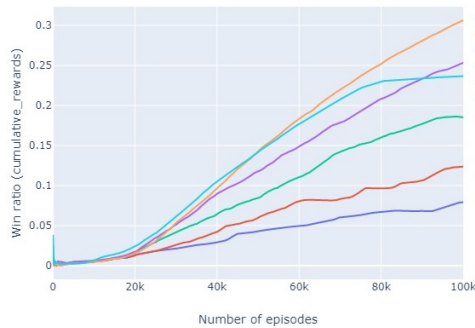


Figure 9: Q-learning convergence properties for different γ discounting rates. For almost every value of γ the positive correlation between higher γ values and the rate of convergence towards the true values is established.

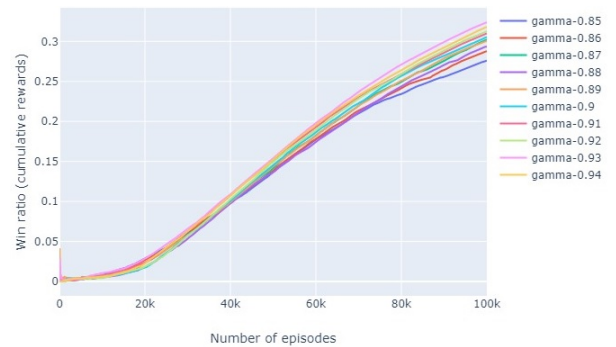


Figure 10: There is only a slight difference in the rate of converge for different γ values in the optimal interval.

The same conclusions can be drawn for SARSA as for Q-learning, with respect to the exponential decay rate for the probability of exploration and with respect to the learning rate. On Figure 13 it is demonstrated that earlier trade-off of exploration against exploitation results in faster convergence of $Q(S, A)$ to $Q_*(S, A)$. However, there is also an upper

limit and when it is reached the convergence degrades, which means the agent does not have enough time to explore its environment and therefore never learns to exploit it. Figure 14 shows that smaller step sizes in the direction of the TD error results in better convergence properties.

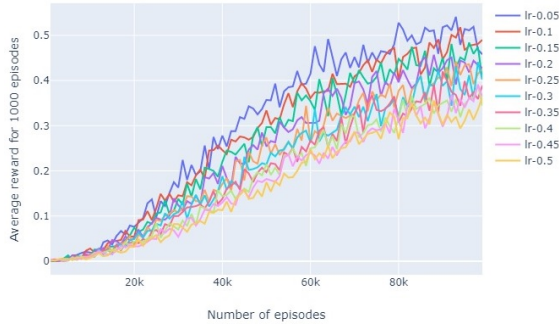


Figure 11: The moving average of the cumulative rewards with a window size of 1000 for different learning rates in the Q-learning algorithm. The rolling mean increases for smaller learning rates.



Figure 12: The length of the episodes converges to the same interval of [60, 70] discrete time steps for the agent in Q-learning, independently of the learning rates.

SARSA is also an online method that updates the Q values as soon as the experience is available. The main difference is that SARSA is an on-policy method and it influences how it updates the Q values, as it was discussed previously. Figure 15 shows that there is a higher variance in the convergence to the optimal policy for different values of γ compared to Q-learning.

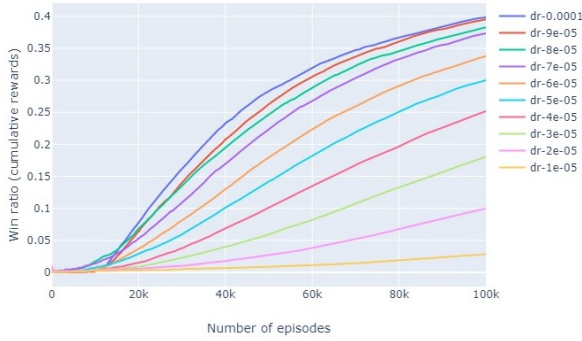


Figure 13: The convergence of SARSA to the true values is faster when the probability of exploration decays sooner. The higher the rate of decay results in faster convergence. The optimal rate of decay is around $1e^{-4}$.

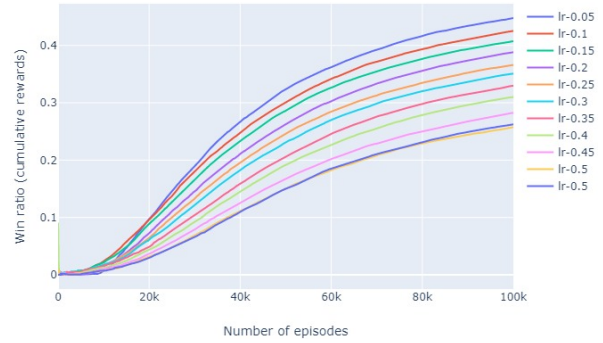


Figure 14: SARSA's rate of convergence to the optimal policy π_* is better with smaller step sizes in the direction of the TD error.

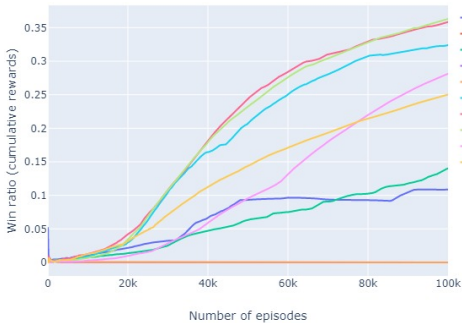


Figure 15: SARSA does not present as strong correlation between the convergence to the true values and the higher values of γ as Q-learning.

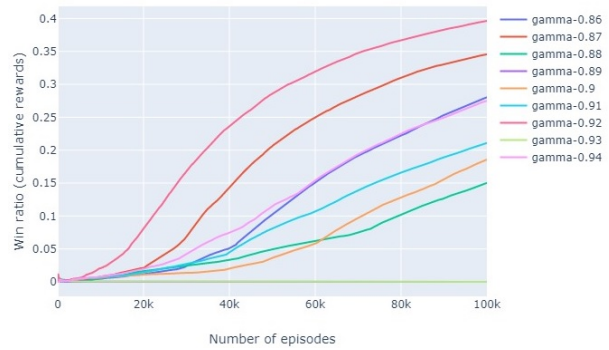


Figure 16: SARSA shows high variance in the rate of convergence for small variances in γ in the assumed optimal range.

In this case there is no positive correlation between the higher values of γ and the rate of convergence to the true values. In the assumed optimal interval for γ there still exists a high variance for small differences in the values of γ according

to Figure 16. On both Figure 15 and 16 with the values of 0.75 and 0.93 SARSA shows no convergence at all. This can also happen due to the unlucky initial series of state-action pairs that does not result in an effective exploration phase, however the probability of exploration decays anyway.

Every algorithm is able to converge to $Q_*(S, A)$ in the FrozenLake environment with different convergence rates. SARSA shows a relatively large difference between the initial convergence ratios compared to Q-learning. Both SARSA and Q-learning reach almost the same accumulated rewards over time as Q-learning catches up with SARSA later. This underpins the theoretical results that SARSA converges faster but Q-learning is able to keep learning while changing policies. The rate of convergence of Q-learning and Monte Carlo is approximately the same until around the 45k-th number of episodes. Regarding the final result, TD methods attain better scores on the FrozenLake problem. The final experiment was conducted from the selection of the best hyperparameters that were calculated previously. The result is shown on Figure 17.

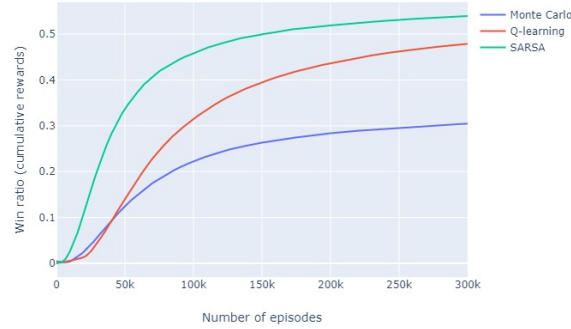


Figure 17: Comparison of MC, Q-learning and SARSA reinforcement-learning algorithms. SARSA shows the fastest convergence properties to the true values and Monte Carlo is the slowest.

5 Conclusion

In this final report I briefly introduced three reinforcement learning algorithms and analyzed their properties by running experiments on OpenAI Gym FrozenLake-v0 8x8 environment. Monte Carlo has the slowest convergence, because it has to wait until the end of each episode to update its Q values. The reward system of the FrozenLake environment also slows MC's convergence. SARSA's convergence is the fastest, however Q-learning is able to maintain learning while changing policies and in time its convergence catches up with SARSA. Every algorithm converges to the optimal Q values and is able to win the game with a high probability, when their parameters are tuned appropriately.

References

- (1) Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* **2013**.
- (2) Van Hasselt, H.; Guez, A.; Silver, D. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016; Vol. 30.
- (3) Sutton, R. S.; Barto, A. G., *Reinforcement learning: An introduction*; MIT press: 2018.
- (4) Watkins, C. J.; Dayan, P. Q-learning. *Machine learning* **1992**, 8, 279–292.
- (5) Rummery, G. A.; Niranjan, M., *On-line Q-learning using connectionist systems*; University of Cambridge, Department of Engineering Cambridge, UK: 1994; Vol. 37.