Reinforcement Learning Practical
Winter Is Coming

Anna Gumenyuk (s3893464) Luc Cronin (s4326245)

University of Groningen, Faculty of Science and Engineering

Abstract

An agent is trained to solve the "FrozenLake-v1", Toy Text environment from OpenAI Gym. The agent was trained using Q-Learning and SARSA. These learning algorithms were used in conjunction with the following exploration strategies: greedy, ϵ-greedy, action preferences, and upper-confidence bound (UCB). Using a random agent as a baseline, these learning algorithms and exploration strategies were compared. The results convey the average reward per episode and indicate that learning is achieved.

Introduction

Reinforcement learning is a type of machine learning where an agent learns to make decisions by receiving rewards or penalties based on its actions. It is widely used in many real-world applications, such as autonomous robots, video games, and recommendation systems. One of the classic problems in reinforcement learning is the Frozen Lake problem, where an agent needs to navigate to reach its goal while avoiding holes.

In this report, we present an analysis of various reinforcement learning algorithms, including SARSA and Q-Learning, and how they were used to solve the Frozen Lake problem. The algorithms were tested with different exploration strategies, such as greedy, ϵ-greedy, action preference, and UCB to evaluate their performance. An agent who operates randomly was used as a baseline for comparison to the other strategies. The results were analyzed and visualized through graphs to show the average rewards for each strategy.

This report aims to provide insights into the behavior of different reinforcement learning algorithms and to demonstrate how they can be applied to solve a classic problem in the field. The results of the analysis can be used as a reference for future researchers who want to explore reinforcement learning further.

Problem Definition

The Frozen Lake environment is a grid world navigation problem that provides a platform for the analysis and comparison of various reinforcement learning algorithms. The particular version used was the FrozenLake-v1. The environment consists of a 4x4 grid with 16 distinct states. The agent starts at the top-left corner and moves towards the bottom-right corner. The agent has four possible actions at each state: move up, down, left, or right. In this specific configuration of the environment, the "is_ slippery" parameter is set to false, meaning that the agent will move in the intended direction in every step. The goal of the agent is to reach the bottom-right corner of the grid. The reward system is sparse; the agent only receives a reward of +1 for reaching the goal. There is no penalty for falling into a hole. This makes the task of navigating the grid to the goal a difficult challenge for the agent, as the reward function offers no indication in regards to better or worse actions. A better action would be one which alters the agent's position closer to the goal, in contrast, a bad move would achieve the opposite effect. Hence, the agent

must learn to navigate to the goal through trial and error. The episode ends when the agent either reaches the goal or falls into a hole. The agent's performance is measured by the average reward obtained over a set number of episodes.

Related Background

Markov Decision Processes(MDPs)

A Markov Decision Process (MDP) is a mathematical framework for modeling decision-making problems under uncertainty. It is widely used in various fields such as economics, engineering, and artificial intelligence, to analyze and solve sequential decision-making problems [4]. The Markov property is a requirement if MDPs are to be successful. It states that the future state of a system is dependent only on its current state, and not on its past history. In other words, the probability distribution of future states depends only on the current state and not on any preceding states. An MDP consists of a set of states, a set of actions, a transition function, and a reward function.

The states in an MDP represent the different conditions or circumstances the decision-making agent can be in at a given point in time. The actions represent the possible choices or decisions that the agent can make. The transition function defines the probability of transitioning from one state to another based on the action taken. The reward function defines the immediate reward or benefit received by the agent after making a decision.

The goal of an MDP is to find an optimal policy that maps each state to actions that maximize the expected total reward over a certain period of time. This policy represents the strategy that the agent should follow to achieve the best outcome. The optimal policy can be obtained using techniques such as value iteration or policy iteration, which are algorithms that iteratively update the value function or policy.

Grid World Environments

Grid-world environments are a type of discrete environment commonly used in the study of Reinforcement Learning. These environments consist of a grid of cells, with the agent able to occupy one cell at a time and can move to an adjacent cell. The MDP framework is well suited for grid-world environments, as these environments are characterized by the Markov property. This property allows the reinforcement learning algorithms to break down the problem into smaller, manageable parts and make decisions based on the current state.

In grid-world environments, the state of the system can be defined as the position of the agent in the grid, and the set of possible actions can be defined as the possible moves the agent can make. The rewards and penalties are determined by the state and action, allowing the MDP framework to be applied to the problem.

The combination of the simplicity of grid-world environments and the ability to apply the MDP framework makes these environments a popular choice for the study of reinforcement learning algorithms.

Temporal Difference Learning

Temporal Difference (TD) learning is a concept in reinforcement learning that makes the error explicit in the approximation of the reward function. This is achieved through the temporal difference error, which is the difference between the estimated reward and the actual observed reward. This calculated error is then used to incrementally update the value function. TD learning algorithms are widely used in reinforcement learning as they can handle problems with large or continuous state spaces and action spaces, and they can learn from incomplete information, such as an incomplete sequence of rewards [5]. There are two main types of TD learning algorithms: on-policy learning and off-policy learning. On-policy learning updates the value function using the same policy that is being executed in the environment, while off-policy learning updates the value function using a different fixed policy.

One popular algorithm in TD learning is the SARSA (State-Action-Reward-State-Action) algorithm, which is an on-policy algorithm that learns the state-value function by updating the estimated value

of each state-action pair based on the observed rewards and the estimated values of the next state-action pair. Another popular TD learning algorithm is the Q-Learning algorithm, which is an off-policy algorithm that learns the state-action value function by updating the estimated value of each state-action pair based on the observed rewards and the maximum estimated value of any possible next state-action pair.

Sparse Reward Environment

A sparse reward environment is a type of reinforcement learning environment where rewards are infrequently encountered, with large intervals between rewards. In these environments, learning the optimal policy becomes a challenging task as the reinforcement signal is limited and the agent has to make decisions based on limited information. This can make it difficult for the agent to determine the impact of its actions and to converge to the optimal policy. Sparse reward environments are common in real-world applications, such as robotics, game playing, and autonomous vehicles [3].

Grid-world environments such as the Frozen Lake problem have sparse reward structures, where rewards are only granted for reaching the terminal state and there are no intermediate rewards along the way. Despite these challenges, MDPs can still be applied to sparse reward environments by using value-based reinforcement learning algorithms. These algorithms can then be used to determine the optimal policy, which maximizes the expected cumulative reward over time. By using these algorithms, it is possible to overcome the challenges posed by sparse reward structures and achieve effective decision-making in grid-world environments.

Methods

Problem Definition

The Frozen Lake environment was simulated by using the OpenAI Gym package [2]. The Frozen Lake is a grid-based environment with a discrete action and state space. The state space is made up of 16 tiles arranged as a 4x4 grid. There are three holes positioned within the environment, these holes count as terminal states and will restart the game. The only remaining type of terminal state is the goal which is located on the bottom right of the grid. All other states are safe spaces. The actions available to the agent include, left, down, right & up. The action space is consistent throughout the states. The reward function for each state is 0, except for the goal state which will earn a reward of +1.

Algorithms and Strategies Used

In this study, two reinforcement learning algorithms were implemented and tested on the Frozen Lake environment: Q-Learning and SARSA (State-Action-Reward-State-Action). These algorithms were selected as they are two of the most commonly used algorithms in reinforcement learning, and they both belong to the class of Temporal Difference (TD) learning algorithms. In conjunction with these learning algorithms, the following exploration strategies were implemented; greedy, ϵ-greedy and Upper Confidence Bound (UCB).

Q-Learning

Q-Learning is an off-policy algorithm, meaning that the algorithm learns from the actions that are taken, rather than the current policy the agent is acting under. Q-values which represent the expected return from taking a specific action in a specific state, are stored in a Q-table. The Q-Learning algorithm updates the action values, by using the Q function defined as follows

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

where α defines the learning rate, gamma defines the decay factor, $\max_{a \in A} Q(s_{t+1}, a)$ defines the largest Q-value for the next state.

After various configurations of α and γ values, we found $\alpha = 0.4$ and $\gamma = 0.8$ to be the most optimal.

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

SARSA, on the other hand, is a model-free, on-policy algorithm. Unlike Q-Learning, the SARSA algorithm updates the action-value function by considering the expected return from taking the action that was actually selected, rather than the best action that could have been selected. The SARSA algorithm also updates the Q-values using its own state-action function.

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

where α defines the learning rate, γ defines the decay factor and $Q(s_{t+1}, a_{t+1})$ defines the action to take in the next state under the current policy.

After various configurations of α and γ values, we found $\alpha = 0.2$ and $\gamma = 0.8$ to be the most optimal.

Greedy Strategy

The Greedy exploration is very straightforward: the key idea of this strategy is to constantly exploit. This refers to choosing the action that has the highest estimated Q-value, or the highest expected cumulative reward. A greedy decision process can be represented by the following equation:

$$A_t = \arg\max_{a \in A} \mathrm{Q}(s_t, a) \tag{3}$$

where A_t is the action to be chosen at time t if its Q-value is the highest.

Epsilon-Greedy

$$\pi_t(s_t, a_t) = \begin{cases} 1 - \epsilon & \text{if a} = \arg\max_{a \in A} Q(s_t, a_t), \\ \epsilon & \text{otherwise.} \end{cases} \tag{4}$$

where, ϵ denotes the probability of exploring the environment, $\arg\max_{a \in A} Q_t(s_t, a_t)$ denotes the action with the highest Q-value. The ϵ hyper-parameter is not constant is and updated by a decay factor after every time step.
After a series of configurations of the ϵ and decay factor, we found the optimal balance between these two hyper-parameters to be $\epsilon = 0.8$ & decay factor = 0.1.

Upper-Confidence Bound (UCB)

The main idea behind the upper-confidence bound strategy is the reduction of the uncertainty in the estimate of the action value. The uncertainty will increase for actions that have not been taken and will decrease for actions that have been selected. In other words, when the action is selected, the uncertainty of that action value gets reduced based on the following functions:

$$A_t = \arg\max_a \left[Q(s_t, a) + c\sqrt{\frac{\ln(t)}{N_t(s_t, a)}} \right] \tag{5}$$

where A_t is the action to be selected, $Q(s_t, a)$ is the estimated value of the state-action at episode t, $N_t(s_t, a)$ is the number of times the action a has been selected at state s_t, and c is a constant that determines the level of confidence. UCB uses a square-root term to represent the uncertainty or variance in the estimated value of action a. When action a is selected, the uncertainty decreases, as reflected in the decrease of the term in the denominator with the increase of $N_t(s_t, a)$. On the other hand, every time an action other than a is selected, t increases, which causes the uncertainty estimate to increase, as reflected in the increase of the term in the numerator. Ultimately, all actions will be selected, but as time goes by, it will take longer for actions with lower value estimates or that have already been selected more frequently to be selected. In our implementation, after various configurations of c, we found $c = 10$ to be the most optimal one.

Action Preference

While the previous algorithms relied on action-value estimates, this strategy involves developing a preference associated with each action. The action with the highest preference has a higher probability of being chosen. The probability is defined by the softmax or Boltzmann distribution. The preference, however, cannot be interpreted in terms of rewards. What matters is the preference for one action over another. As such, the probability of the action is described by the equation:

$$\pi_t(s,a) = \frac{e^{H_t(s,a)}}{\sum_{k=0}^{n} e^{H_t(s,k)}} \tag{6}$$

where $H_t(a)$ is the preference value for action a and time t. The preference values for each action get updated via the following set of equations:

$$H_{t+1}(s,a') = H_t(s,a') + \alpha(r_t - \hat{r}_t)(1 - \pi_t(s,a')), \tag{7}$$

$$H_{t+1}(s,a) = H_t(s,a) + \alpha(r_t - \hat{r}_t)\pi_t(s,a) \quad \text{if } a \neq a' \tag{8}$$

where, a' denotes the action selected, a denotes the actions which were not selected, α denotes the learning rate, $\hat{r}_t$ denotes the average of all the rewards gathered until time step t. A series of values were trialed for α, however none allowed for convergence of the agent.

Study Design

The experimental design was kept simple. The algorithms were trained across a hundred experiments. Within each experiment, there were a hundred episodes. An episode consists of the agent beginning in the start state and going through the environment and reaching one of the terminal states, causing the environment to reset. The results were obtained by averaging the reward per experiment.

Results

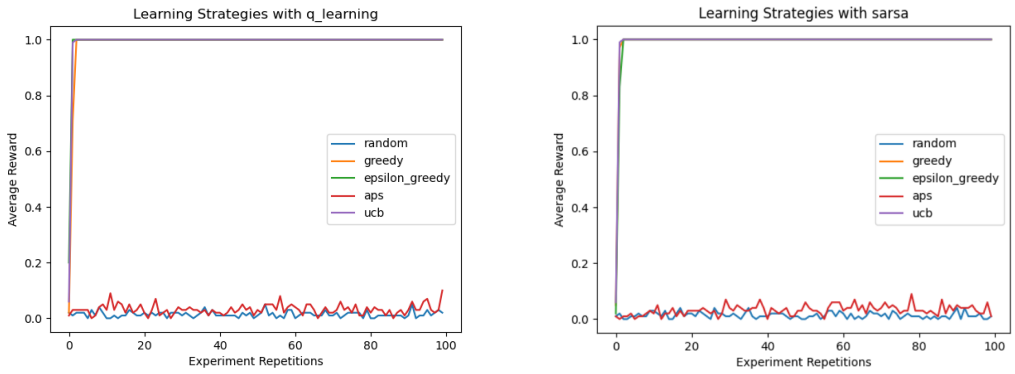
The results of our study are effectively depicted in figure 1. It can be observed that it is challenging to distinguish between the greedy, ϵ-greedy, and upper confidence bound strategies on the graph. This is because the average reward for these three strategies reaches 1 within the first couple of experiments, regardless of the type of reinforcement learning paradigm employed. The performance remains as such for the rest of the experiments, meaning that the agent has learned to make correct moves to reach the goal state. On the other hand, it is evident that the action preferences strategy performs worse compared to the other strategies. This strategy appears not significantly better than a random strategy, while all the other strategies seem to outperform it.

To investigate these results further, a two-way analysis of variance (ANOVA) was performed to investigate the effect of the reinforcement learning paradigm (SARSA and Q-learning) and exploration strategy on the mean reward of the agent across the experiments. The results showed a significant main effect of the exploration strategy ($F(4, 990) = 7624.49,\ p < 0.05$), indicating that different strategies contributed differently to the reward values.

However, there was no significant main effect of the choice of paradigm ($F(1, 990) = 0,\ p = .99$). These results suggest that the choice of the strategy has a more significant impact on the reward values than the choice of a paradigm. Moreover, the results indicated that there was no significant interaction between the reinforcement learning paradigm and the exploration strategy ($F(4, 990) = 0.22,\ p = .93$). This indicates that the effect of the strategy on the reward values did not depend on the paradigm used.

Furthermore, we performed multiple comparisons of means of rewards between different strategies, disregarding the type of reinforcement learning algorithm because of the insignificant effect. The results show that there was a significant difference in the mean reward between the ϵ-greedy and action preferences strategies ($M = 0.95,\ SE = 0.01,\ t = 78.77,\ p < .001$), between the greedy and action preferences ($M = 0.96,\ SE = 0.01,\ t = 78.82,\ p < .001$), and between UCB and action preferences ($M = 0.96,\ SE = 0.01,\ t = 79.39,\ p < .001$).

There was also a significant difference between the random and epsilon-greedy strategies ($M = -0.97,\ SE = 0.012,\ t = -80.00,\ p < .001$) and between the random and greedy strategies ($M =$



(a) A plot of the average rewards across the experiments obtained by the agent using different strategies with the Q-learning paradigm.

(b) A plot of the average rewards across the experiments obtained by the agent using different strategies with SARSA paradigm.

Figure 1: Performance plots of the agent with Q-learning and SARSA paradigms.

$-0.97,\ SE = 0.012,\ t = -80.06,\ p < .001$) and between UCB and random strategies ($M = 0.977,\ SE = 0.012,\ t = 80.63,\ p < .001$).

However, there was no significant difference between the random and action preferences strategies ($M = -0.015,\ SE = 0.012,\ t = -1.24,\ p = .73$), or between the greedy and epsilon-greedy strategies ($M = 0.0007,\ SE = 0.012,\ t = 0.058,\ p = 1.0$), or between UCB and epsilon-greedy strategies ($M = 0.007,\ SE = 0.012,\ t = 0.627,\ p = .97$), or between the upper confidence and greedy strategies ($M = 0.007,\ SE = 0.012,\ t = 0.569,\ p = .98$).

Discussion

In this study, an analysis of a series of learning algorithms and exploration strategies was conducted in the context of the Frozen Lake environment. The results were compared and analyzed. Throughout the discourse of this study, several points of discussion were found regarding the performance of each algorithm and strategy. With regards to learning paradigms, Q-learning and SARSA did not differ significantly. In light of [1], this is an interesting observation, as we expected SARSA to produce better results. Due to the similar effect of both paradigms, the performance of the agent was largely determined by its strategy selection.

Moving on to the strategies, the ϵ-greedy exploration strategy performed well. The performance was attributed to the balancing of the exploration and exploitation of the environment. Due to the sparse reward system of the environment, large amounts of exploration are needed. The Algorithm also stabilizes due to the decay rate affecting the epsilon. As the agent has had time to explore the environment more and collect sufficient information, it can then begin to exploit more and more.

Much like the reasoning for the success of ϵ-greedy, UCB also performs well as the balance between exploitation and exploration is well suited to the environment. The exploration component of the UCB algorithm takes the form of making the uncertainty explicit. The uncertainty is made formalized in a manner that allows for a thorough exploration of the environment. The further tuning of the exploration constant enabled the convergence of the agent.

It is very unclear as to the reasons for the success of the greedy algorithm as there is no balance between exploration and exploitation. This would make an interesting topic for future research to determine the reasoning for the positive performance of the greedy algorithm.

The action preferences exploration strategy performed worse than the other strategies. Despite initial expectations of better performance compared to the greedy strategy, it was found to not be significantly different from a random strategy. The poor performance can be attributed to the algorithm's inefficient learning rule that does not allow for effective learning from experience. This is due to the fact that, for most state-action pairs, the average reward $\hat{r}_t$ remains 0 throughout the episodes. As a result, the factor $(r_t - \hat{r}_t)$ is usually 0, nullifying other components, such as α and $\pi t(s, a)$ or $1 - \pi_t(s, a)$, leading to

$H_{t+1}(a)$ remaining unchanged even if the action a is chosen. This results in equal preferences for all actions given a state, leading to equal probabilities for all actions and behavior akin to that of a random agent.

The poor performance of the action preferences strategy in the Frozen Lake environment could also be due to the sparse reward structure of the task. In the original Frozen Lake environment, most state-action pairs have a reward of 0, making it difficult for the algorithm to learn effectively. However, when we introduce a penalty of -1 for falling through a hole, the performance of the action preferences strategy improved significantly and became comparable to other strategies. This improvement is clearly demonstrated in figure 2. This suggests that the action preferences strategy may perform better in environments with more meaningful learning signals as opposed to sparse rewards.

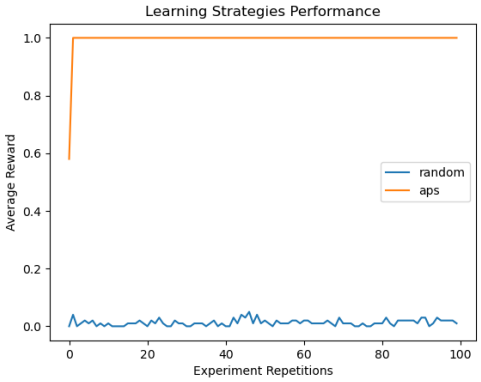


Figure 2: A plot of the average rewards across the experiments obtained by the agent using the action preferences strategy after implementing the negative reward for falling into the hole.

As such, the results of our experiments indicate that both Q-learning and SARSA reinforcement learning algorithms, combined with greedy, ϵ-greedy, and UCB exploration strategies, have been successful in training the agent to complete the task in the Frozen Lake environment. The only strategy that did not perform well was the action preferences strategy. Given the improvement in the action preferences strategy with the introduction of a penalty for falling into a hole, a possible direction for future research is to revise the reward function in the Frozen Lake environment. While designing a reward function is a complex task, one suggestion is to include both a penalty for falling into a hole and rewards for the agent being on safe tiles. For instance, the reward received at a given state after taking a particular action could be based on how close the agent is to the goal after performing that action. We believe that implementing this idea would provide valuable insights into optimizing the performance of agents in such environments.

An interesting point for future research would be the investigation of other exploration strategies which are not solely trying to maximize a reward. Exploration strategies such as information gain are focused on the acquisition of as much information about the environment as possible. Information gain is a measure of the reduction in entropy of the agent's beliefs about the state-action values after taking an action. The agent will take the action with the highest information gain following the exploration strategies logic. Hence, this exploration strategy has the potential to perform well in a sparsely rewarded environment.

Finally, we suggest training the agent on the stochastic version of the environment. Specifically, if the Frozen Lake environment becomes slippery, it means that the agent's movements become unpredictable. As a result, the agent might take an action that leads to a different state than expected, and the reward received might also be different. This makes the problem more complex, as the agent needs to learn to navigate in an environment that is more difficult to control. It would require the agent to have a more robust policy and a better understanding of the dynamics of the environment to complete the task successfully. Current paradigms and strategies are not appropriate for stochastic environments, so other approaches are necessary.

References

[1] Andrew Barto and Richard S. Sutton. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 2014.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[3] Maxim Lapan. *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt Publishing Ltd, 2018.

[4] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[5] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.