Sapienza University of Rome
AI Section 2B
Reinforcement Learning Project

Hossam Arafat - Matricola: 1803850

27 September 2018

SAPIENZA
UNIVERSITÀ DI ROMA

Abstract

Reinforcement learning is a sub-field of Machine Learning and Artificial Intelligence that focuses on the problem of making decisions in unknown or uncertain environments, and learning through trial and error - by directly interacting with the environment. This report presents an interface where the user is able to run On-Policy SARSA or Off-Policy Q-Learning agents on any environment of choice in Open AI Gym. To showcase the interface, the following environments were developed:

1. Cliff Walking Environment
2. Windy Gird-World Environment

The code for this project was written in Python and used Open AI's Gym and some starter code developed by Microsoft. The algorithms are able to find the optimal policy provided the agent explores enough of the state space and there is a reasonable schedule for exploration and exploitation.

1 Environments

1.1 Gym

Gym is toolkit for creating, testing, and comparing Reinforcement learning algorithms. It was developed and is maintained by OpenAI. The gym library contains a collection of environments that have a shared interface, allowing the development of general "agent" algorithms that interact with them.
The environment evolves in a series of discrete time steps $t = 0, 1, 2..$
The agent starts at some state $s_t \in S$ and at each time step t, it takes an action $a_t \in A$ and observes the resulting state s_{t+1} and reward R until it reaches the terminal goal state T.

1.2 Custom Environments

Using Gym's extremely modular and shared interface, 2 environments described in [1] were created to test and showcase the differences between the On-Policy SARSA and Off-Policy Q-Learning algorithms.

1.2.1 Cliff Walking Environment

As shown in Figure 1, "Cliff Walking" is a simple 4x12 grid of cells, bounded by walls. The initial state has the yellow agent starting at the bottom left corner, with the green goal at the bottom right. The agent can move in the 4 cardinal directions and and receives a reward of -1 at every transition and -100 at any such transition that leads the agent to any of the "Cliffs", or purple grid cells.
The agent is unaware of the environment's transition or reward models.

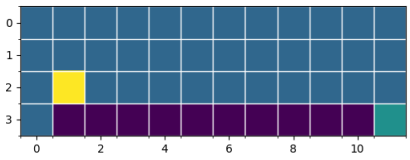

Figure 1: The Cliff Walking Environment

1.2.2 Windy Gird-World Environment

As shown in Figure 2, "Windy Grid World" is a simple 7x10 grid of cells, bounded by walls. The initial state has the yellow agent starting at the top left corner, with the green goal at cell (8,4). The agent can move in the 4 cardinal directions but wind of strength 1 in the light purple columns and 2 in the darker purple might randomly and forcibly push the agent upwards. The agent is unaware of the environment's transition or reward models and receives a reward of -1 at every time step until the goal is reached.

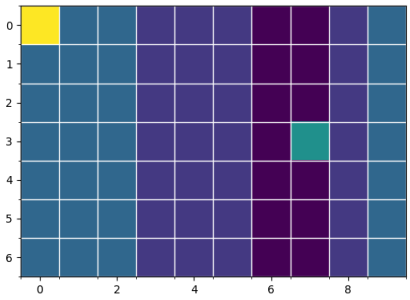

Figure 2: The Windy Gridworld Environment

2 Implementation

Two Reinforcement algorithms were implemented on the 2 environments described above, namely:

1. On-Policy SARSA ($TD[0]$)
2. Off-Policy Q-Learning

In a Reinforcement learning problem, the agent's goal is to interact with the environment and maximize the expected future discounted reward $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R'_k$ by finding the optimal policy $\pi^*(s)$; a mapping from states to the optimal actions to take in every state.
The optimal policy can be found by first defining an action-value function $Q_\pi(s, a)$ that estimates the value of taking an action a in a given state s under policy π. This function is defined as the expected return $\mathbb{E}[G_t]$ starting from s, taking action a, and thereafter following policy π. Thus, the optimal policy can be found by finding the optimal Q values and choosing the greedy action that maximizes the Q function at every state: $\pi^*(s) = \underset{a}{\text{argmax}}\, Q^*(s, a)$.
Such process of iteratively finding the action-value function of a given policy, $Q_\pi(s, a)$ and improving the policy is nothing other than the widely known Generalized Policy Iteration (GPI) algorithm.

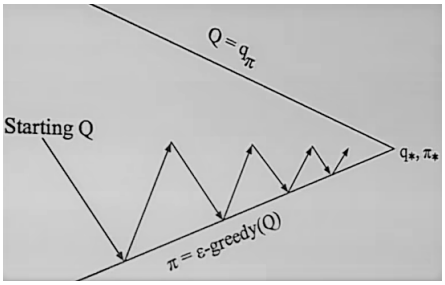

Figure 3: The Generalized Policy Iteration Algorithm

2.1 The Exploration-Exploitation Dilemma

As mentioned earlier, the TD methods discussed thus far calculate the optimal action-value function $Q^*(s, a)$ and execute Policy Improvement by taking the action with the highest Q value, $\underset{a}{\text{argmax}}\, Q(s, a)$.

However, a key insight is that if we improve the policy by only constantly taking the greedy action, we may never take actions that lead to slightly poorer states that are gateways to even bigger rewards later on. Essentially, constantly exploiting the values we know prevents the agent from exploring and finding new states that yield even greater rewards.

This is handled by using $\epsilon - greedy$ policy improvement, where, with a small probability ϵ, the agent preforms a random action, and with a probability $1 - \epsilon$ the agent picks the greedy action.

2.2 Temporal Difference Methods

Temporal difference methods utilize the Bellman equation to find $Q^*(s, a)$ using Bootstrapping in an iterative manner. Simply put, the agent starts with some estimate of $Q(s, a)$ -usually 0, it then takes an action a_t from the current state s_t, observes the immediate reward R it acquires as well as the successor state s_{t+1} it ends up in. It then bootsraps and updates the initial guess of the expected return from state s_t, $\mathbb{E}[G_t] = Q(s_t, a_t)$ towards the immediate reward it acquired R, and the expected future return from state s_{t+1}, $\mathbb{E}[G_{t+1}]$. The update occurs by taking the difference between the two weighted by a small scalar value α and adding it to the current estimate of $Q(s, a)$.

The aforementioned difference is often referred to as the TD error, and the term $Q(s, a)$ is subtracted from as the TD Target. More formally:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\big[TD\ error\big]$$

such that:

$$TD\ error = \big[TD\ Target - Q(s_t, a_t)\big]$$

where α; a small scalar value that denotes how much weight is placed on the new experience just learned from the environment.

2.2.1 On-Policy SARSA

The fundamental difference between SARSA and Q-Learning is the TD Target employed in each method. SARSA is an On-Policy method, where the agent is trying to estimate the values $Q_\pi(s, a)$ by following policy π.
In SARSA, the TD Target is constructed as follows:the immediate reward R is added to our expected discounted future return of taking action a_{t+1} according to policy π and acting optimally thereafter.

$$SARSA\ Target = \big[R_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\big]$$

And the Full SARSA update is:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\big[R_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\big]$$

2.2.2 Off-Policy Q-Learning

Q-Learning is an Off-Policy method, where the agent is trying to learn the values of its target policy, π, a greedy one in this case, by following an $\epsilon - greedy$ behaviour policy, μ.
In this Off-Policy method, the agent picks the action a_t it takes in state s_t using the $\epsilon - greedy$ behaviour policy, μ, but then updates the current estimate of $Q(s, a)$ towards the alternative action A'_{t+1} suggested by the greedy target policy, π.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[R_t + \gamma Q(s_{t+1}, A'_{t+1}) - Q(s_t, a_t)\big]$$

The greedy target policy selects a greedy action A' as the action that will maximize the Q value of the next state $\pi(s_{t+1}) = \underset{A'}{\operatorname{argmax}}\, Q(s_{t+1}, A'_{t+1})$.

$$Q - Learning\ Target = \big[R_t + \gamma Q(s_{t+1}, \underset{A'}{\operatorname{argmax}}\, Q(s_{t+1}, A'_{t+1})) - Q(s_t, a_t)\big]$$

Since $Q(\underset{A'}{\operatorname{argmax}}\, Q(s_{t+1}, A'_{t+1}))$ is simply the maximum Q value at the next state, $\max Q(s_{t+1}, A'_{t+1})$, the Full Q-Learning update becomes:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\big[R_t + \gamma \max Q(s_{t+1}, A'_{t+1}) - Q(s_t, a_t)\big]$$

3 Experiments

SARSA and Q-Learning TD methods were applied to the Cliff-Walking and Windy Girldworld Environments. $\epsilon - greedy$ action selection was implemented where the ϵ is decayed gradually from its initial value every fixed number of given time steps. This allows the agent to start taking exploratory actions and sample / visit enough $\{s_t, a_t, R, s_{t+1}\}$ sequences from the environment to learn $Q^*_\pi(s, a)$.
A modular interface was developed where the user can specify on the command line the following:

1. The Algorithm or Agent to run (SARSA or Q-Learning)
2. The Environment (Can be any of Gym's Environments)
3. The Number of iterations to run
4. Epsilon
5. Alpha
6. Number of steps after which epsilon is decayed by 0.98 of its original value

More instructions on the usage of this project are provided in the ReadMe file provided. As shown in Figure 4, The interface includes a Simple 4x4 Gridworld environment that is used to quickly test the convergence of algorithms. The yellow agent starts in the top left cornet with the cyan goal in the bottom right corner. The agent receives a reward of 0 at every time step and a reward of 1 when the goal is reached.

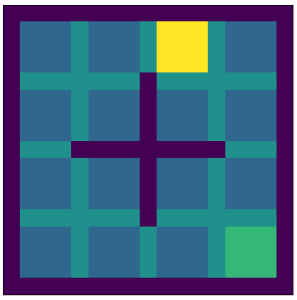

Figure 4: Simple Gridworld Environment used for quickly testing the convergence of algorithms

4 Results

4.1 Cliff Walking Environment

4.1.1 Q-Learning

As shown in Figure 5, the agent has learned to find the optimal policy that reaches the goal in the shortest path possible of only 13 steps, achieving a reward of -13.

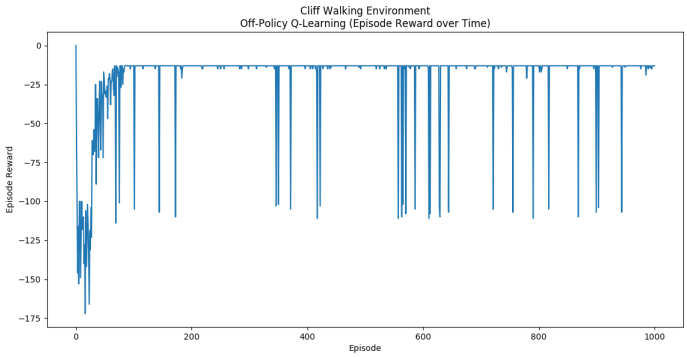

Figure 5: Rewards achieved by the Q-learning Agent on the Cliff Walking Environment

4.1.2 SARSA

As shown in Figure 6, the agent takes the outcomes of successor action into account and has consequently learned to find a "safer" policy than the one found by the Q-Learning agent that reaches the goal in 17 steps, achieving a reward of -17.

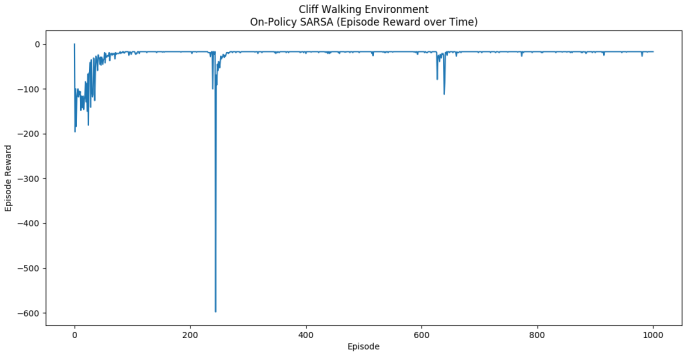

Figure 6: Rewards achieved by the SARSA Agent on the Cliff Walking Environment

4.2 Windy Grid World Environment

In this case, both agents are able to find the same optimal policy that reaches the goal in the minimum number of steps, 15, obtaining a reward of -15.

4.2.1 Q-Learning

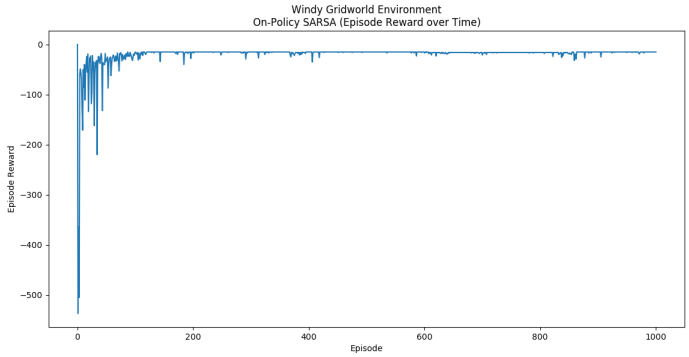

Figure 7: Rewards achieved by the Q-Learning Agent on the Windy Gridworld Environment

4.2.2 SARSA

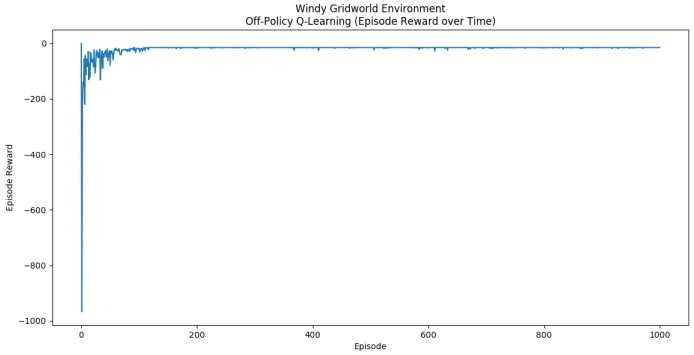

Figure 8: Rewards achieved by the SARSA Agent on the Windy Gridworld Environment

5 Conclusion

This work was able to design a modular agent-environment interface using Gym that can successfully run On-Policy SARSA and Off-Policy Q-Learning algorithms on any of Gym's environments.
To showcase the interface, 2 simple environments mentioned in [1] were selected.
The basic idea behind TD Learning was introduced and the differences between On and Off policy methods were successfully showcased.

6 Future Work

This interface can be improved upon by adding other Reinforcement Learning algorithms such as Q Learning with Function Approximation, DQN, DDQN. Moreover, given a more powerful machine, experiments can be run on Gym's Atari environment that contains more complex environment with a larger state space.

7 References

[1] Richard S. Sutton and Andrew G. Barto, (1998). Reinforcement Learning: An Introduction. Cambridge, MA: MIT Press.