Q Learning Algorithm in continuous space: Python implementation using OpenAI environment

Zara Thomas (acwg618) - City, University of London

I. Define A Domain And The Task

REINFORCEMENT (RL) learning can be used to make an agent learn to interact with an environment. The goal is to optimize the behaviour of the agent with respect to a reward signal that is provided by the environment.

This project uses Reinforcement Learning techniques, in particular Q Learning methods to solve the pole-balancing task as shown in fig.1. The objectives are to apply forces to a cart moving along a frictionless track to prevent the pole hinged to the cart from falling over and to also prevent the cart from straying too far from its initial position.

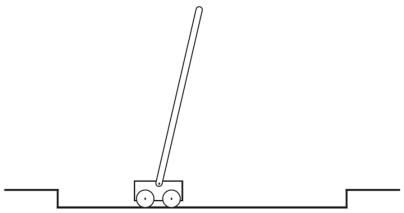

Fig. 1: The pole-balancing task

Due to ease of use, Open AI gym will be used with python in order to implement the solution. Open AI Gym is a toolkit for developing and comparing reinforcement algorithms on a wide range of environments e.g. Pong and Pinball.

In this environment, the system is controlled by applying a force of -1 or +1 to the cart i.e. the two discrete actions in this scenario are pushing the cart pole left and right.The episode ends when the pole is more than 15 degrees from the vertical, or the cart moves more than 2.4 units from the centre [2].

This problem can be solved when modelled as a Markov Decision Process (S, A, R, P) where [1]:

- S, s_t is the set of all states
- A is the set of all possible actions
- R: the reward function that maps a state s_t, an action a_t and the resulting state s_{t+1} into a reward $R(s_t, a_t, s_{t+1})$
- P: is the state transition probability matrix
- $P^a_{ss'}$: $\mathbb{P}[s_{t+1} = s' | S_t = s, A_t = a]$

II. Define A State Transition Function

The state transition function defines the range of states that are available to an agent given its current state $s_{t+1} = \delta(s_t, a_t)$. At every time step, the state is defined by the agent's position, velocity, angle and angular velocity. Therefore the state-space has four dimensions of continuous values. This added complexity is simplified by discretizing the space into 10 bins. Therefore, there are 10^4 i.e. 10,000 states. Two discrete actions(moving the cart pole left and right) are available for the agent to move to the next state (given the 10,000 states present in this model).

III. Define The Reward Function

An agent receives an immediate numerical reward r_{t+1} (where $r+_{t+1} = r(s_t, a_t)$) once it has transitioned to a new state. A reward of +1 is provided for every timestep that the pole remains upright. If the episode terminates, a reward of -1000 is received. The agent's goal is to maximize the total amount of reward it receives over the long run. A reward of +1 provides an incentive for the agent to keep the pole upright.

IV. Choose A Learning Policy

A policy π_t is a function that maps states to probabilites of selecting a possible action, where $\pi_t(s, a)$ is the probability that $a_t = a$ if $s_t = s$. Q-Learning is an off-policy method whereby the policy that is learned is not necessarily the optimal policy. On-line learning involves a balancing act between making the best decision given actions that have been effective in producing rewards (exploitation) and trying out new actions in an attempt to make better action selections in the future (exploration)[1]. ϵ-greedy policies are used to achieve this balance, meaning that an action is selected randomly with a probability of ϵ otherwise, for the majority of the time the action with the highest estimated action-value is chosen with a probability of $1 - \epsilon$. The policy improvement theorem ensures that any ϵ-greedy policy with respect to Q will eventually converge to an optimal value [1].

V. A Graphical Representation

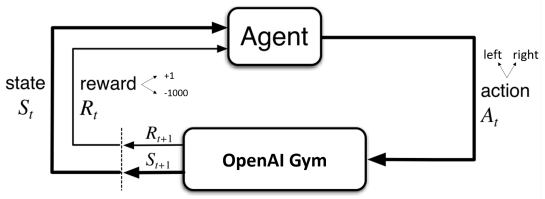

Fig. 2: The agent-environment interaction in Cartpole

VI. R-Matrix

In this problem, all states have the same reward (+1) except for the losing state (-1000) as defined in section III. Hence, there is no R-Matrix defined explicitly.

VII. Define and set the parameters for Q-Learning

Q-learning in its simplest form is defined by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

where $[s, a, r, s']$ is an experience tuple as described earlier, α denotes the learning rate ($0 < \alpha \leq 1$) and γ, a discount factor. The learning rate determines how quickly the error ($[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$) i.e. the difference between the target and the old estimate is reduced. At 0 values are never updated and at high values e.g. 0.9, learning occurs quickly. The discount rate determines the importance of future rewards. At 0, the agent is myopic and strives for immediate rewards whereas, as γ approaches 1, the agent will strive towards future rewards with greater weight (delays the reward). The parameters of the Q-learning algorithm are found in Table 1.

TABLE I: Q-Learning Parameters

Parameter	Value
α	0.5
γ	0.9
ϵ	0.1
π	ϵ-greedy

ϵ has been defined as 0.1 so that 10% of the time, the agent will choose a random action to encourage exploration. α is 0.5 as this is a good initial heuristic for the learning rate.

VIII. Original Q-matrix and updates during learning

The Q-matrix is implemented using a Python dictionary where new observations(states) encountered by the agent within the continuous space is added during learning. Hence, the initial Q-matrix is an empty dictionary, not a NxN matrix of zeroes. Therefore, no Q-matrix is available.

IX. Performance vs Episodes

Q-learning is evaluated using 10,000 episodes in each run. This is replicated 10 times, to allow for volatilities in agent behaviour due to the stochastic nature of the policy. The results were then averaged and smoothed to produce the graphs in the following section. The effects of varying the four parameters α,γ and ϵ and the reward values are evaluated.

A. Q-Learning With Default Parameters

Q-learning is evaluated using the cumulative reward scores and the number of steps achieved over time. Every time the agent terminates an episode and a reward of -1000 is scored. The graphs in the following section show the cumulative rewards obtained before the episode has terminated. Figure 3 shows how the agent is able to quickly learn how to keep upright - indicated by the increasing number of steps over time. As the cartpole game is able to continue for an infinite amount of time, the game is said to be solved once the agent is able to keep upright for 200 steps. The graphs shows that the agent is able to keep upright on average once it has learned the best policy. Convergence occurs tat approximately 3000 episodes. However, there are still instances where the agent terminates prematurely, due to the agent picking a random action (as a result of epsilon). These termination points are represented by the peaks in the graph and occur less often as the agent continues learning.

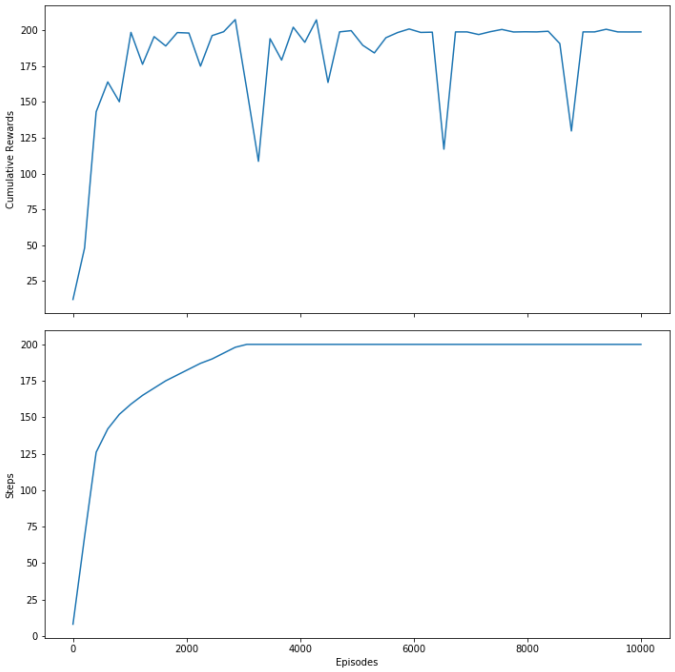

Fig. 3: Q-Learning using default values

B. Effect of Varying α

In this section, the effect of varying α on Q-learning performance is investigated. α values with differing orders of magnitude were compared ($0.001, 0.01, 0.1$ and 1.0).

The results demonstrate that there is a clear relationship between the learning rate and the cumulated rewards obtained on average. The rate at which cumulated rewards are obtained increases between α values of 0.001 and 0.1. However, at $\alpha = 1$ the same trend is not observed with the second lowest rate observed. Interestingly, α values greater than 0.01 appear to reach steady state at approximately the same time (5500 episodes). Figure 2b demonstrates that there is a trade off between the size of the learning rate and the rate at which the agent acquires more rewards. Smaller learning rates on average returned a greater number of steps (rate of learning) far more quickly e.g. $\alpha = 0.1/0.01$. The cumulative rewards obtained initially demonstrate a slightly different behaviour - with greater gains obtained the higher the learning rate is. At the start of learning, the agent is likely to make a lot of errors. Hence, the quicker the agent is able to take these learnings into account the higher the gains will be. However,once the rate of learning begins to stabilize, intermediate α values return the best cumulative rewards in the long term. These α values do not give new states a high influence, which may otherwise affect the performance of the agent if taken heavily into account.

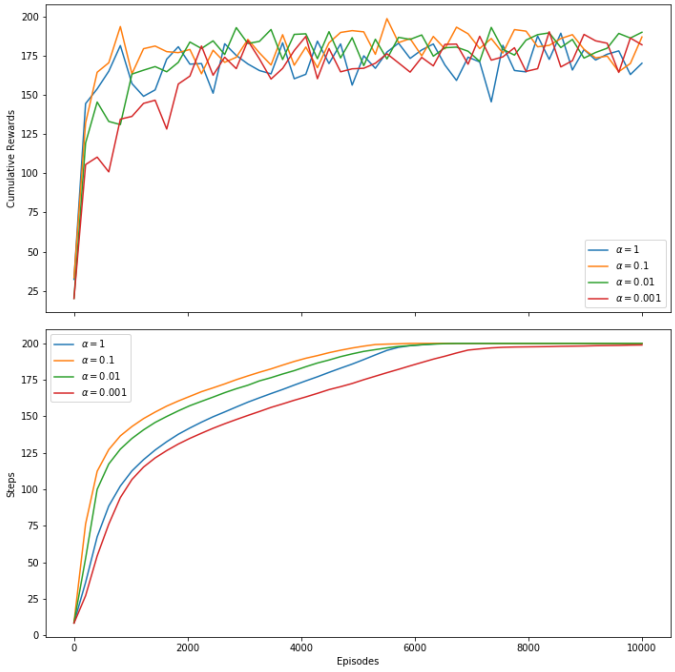

Fig. 4: Varying α values

C. Effect of Varying γ

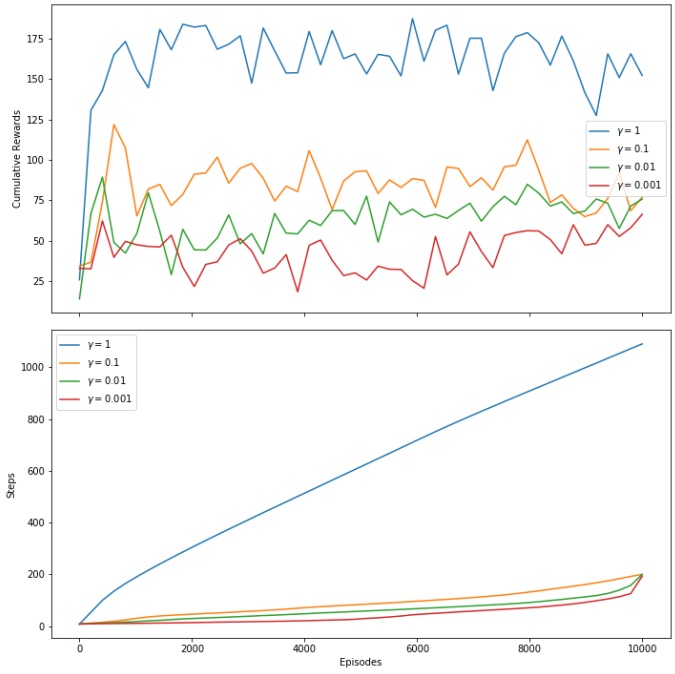

Fig. 5: Varying γ values

This experiment investigates how the discount factor affects the Q-learning performance of the model. γ values with differing orders of magnitude were compared (0.001, 0.01, 0.1 and 1). Figure 3, demonstrates a clear relationship between the discount factor and the cumulative rewards obtained. Increasing the γ results in an increase in the cumulative rewards obtained, however, there is a significant increase for $\gamma = 1$. Therefore, decreasing γ results in a decrease in performance as well. This is because the agent must be able to take account of previous results in order to make the best move in the future. Low gamma values favour current rewards and not long term rewards.

D. Effect of Varying ϵ

In this section, the probability with which the agent explores the environment is varied. The exploration factor (ϵ) determines the probability with which the agent chooses to explore the environment rather than exploit the knowledge that the agent has already gathered from the environment. ϵ values with different orders of magnitude were compared ($0.001, 0.01, 0.1$ and 1.0). When $\epsilon = 1$, the agent performs the worst - accumulating rewards at the slowest rate. This is most likely because the agent is unable to benefit from exploitation as the agent is always selecting actions at random. The graph showing steps vs episodes corresponds with the performance of models observed in the cumulative rewards graph with the worst performing models achieving fewer steps in comparison to the best performing models.

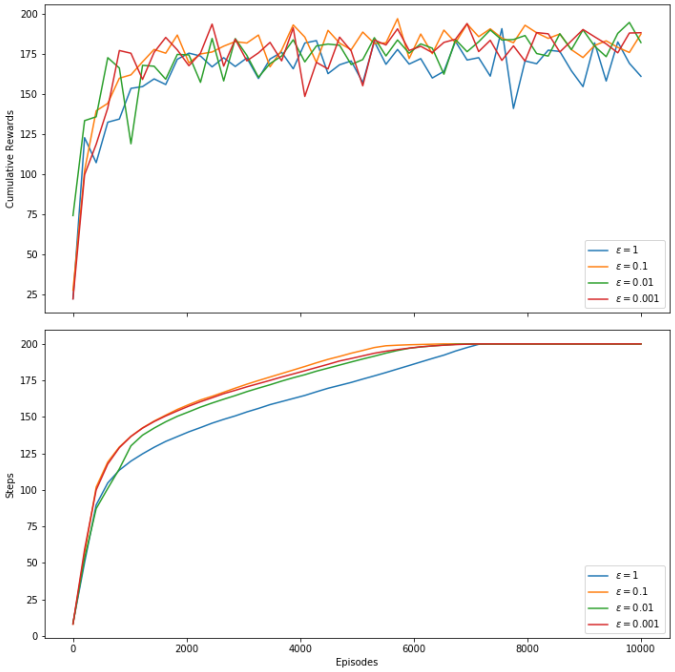

Fig. 6: Varying ϵ values

E. Effect of Varying reward

The effect of varying the final reward on the Q-Learning performance was experimented with. The reward values evaluated were $(-1000, -500, -50, 0$ and $500)$. The graphs in figure 6 clearly demonstrate that the reward values have a significant effect on both the cumulative rewards and the rate at which the agent learns. Negative values enable the agent to learn how to keep the cartpole upright at a significantly higher rate. Non-negative rewards still enable the agent to learn however, at a significantly lower rate. This is because the agent is motivated

by the +1 reward gained from staying upright. Therefore, no final reward is required to guide the agent towards the goal state, as the model itself provides this incentive.

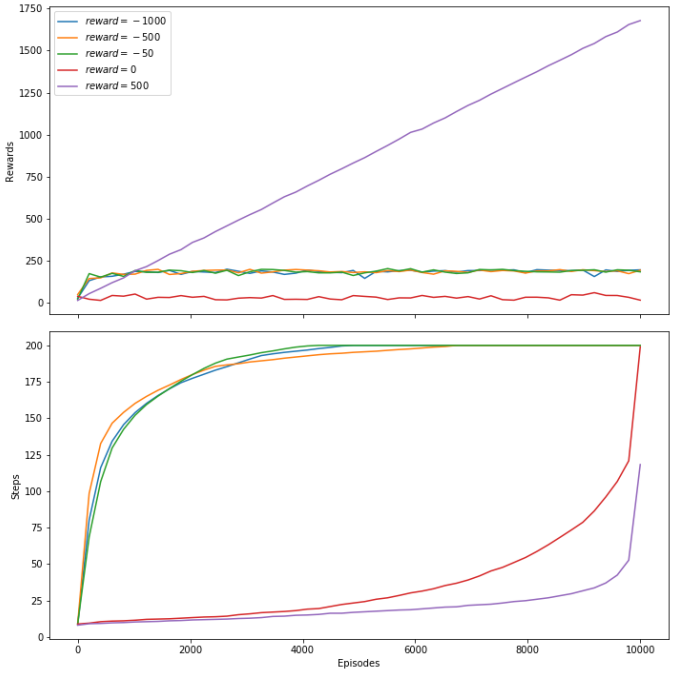

Fig. 7: Varying reward values

X. SARSA

Rather than using an off-policy temporal difference algorithm i.e. Q-learning, an on-policy algorithm is evaluated - SARSA. This policy involves estimating $Q^{\pi}(s,a)$ for the current behaviour policy π, all states s and all actions a [1]. The algorithm for SARSA is similar to Q-Learning and is as follows:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)$$

This update is carried out after every transition from a non-terminal state s_t. Unlike SARSA, Q-learning learns action values relative to the greedy policy and not the policy it follows.

The SARSA algorithm was implemented using the default parameters used in Q-Learning. Figure 7. demonstrates that the rate at which the agent learns is a significantly slower in comparison to Q-Learning. However, the rewards accumulated over time are extremely volatile. This is because with SARSA using the next action a_{t+1} additional variance is introduced into the update when the estimation policy is stochastic and this slows convergence [3].

XI. Conclusion

From changing the parameters α, γ, ϵ and the reward value, it was observed the changing γ and the reward value had the most significant effect on the Q-Learning performance.

High γ values cause the agent to focus solely on immediate rewards, losing information that could help the agent achieve

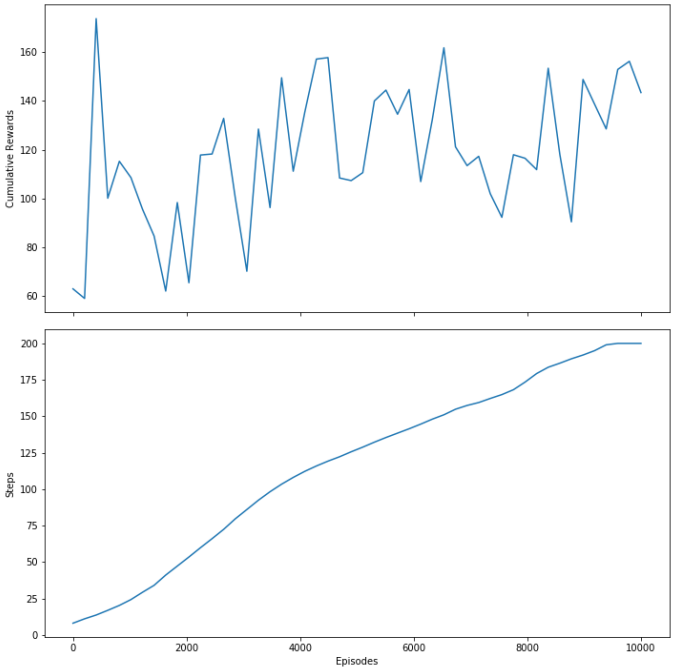

Fig. 8: Cartpole Performance using SARSA

greater long term rewards. Negative reward values has the effect of motivating the agent to achieve greater returns at a faster rate. Both α and ϵ affect the behaviour of the agents in determining the rate at which the agent learns. α directly defines the rate at which the agent learns and ϵ defines the probability with which a random action is picked over the optimal policy. Lower values of ϵ enable quicker convergence. The case with default values shows how the ideal Q-Learning performance should look like - quick convergence and in terms of the rate at which the agent learns the optimal policy and achieving the right balance between exploitation and exploration so that fewer errors are made in the long run. Lastly, SARSA was shown to converge towards the optimal policy but at a significantly slower rate.

Appendix A
State-Space

Num	Observation	Min	Max
0	Cart Position	-2.4	2.4
1	Cart Velocity	$-\infty$	∞
2	Pole Angle	$-41.8°$	$41.8°$
3	Pole Velocity At Tip	-∞	∞

References

[1] R. S. Sutton and A. G. Barto, Reinforcement learning, Learning, vol. 3, no. 9, p. 322, 2012.

[2] D. Hunter, Open AI Gym: Cartpole-v0 Documentation, 2017. [Online]. Available: https://github.com/openai/gym/wiki/CartPole-v0. [Accessed: 01-Feb-2018].

[3] S. Harm van, H. Hado van, S. Whiteson, and M. Wiering, A theoretical and empirical analysis of expected sarsa, in 2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, ADPRL 2009 - Proceedings, 2009