Q-Learning and SARSA: A comparative study on Cartpole-v0

Shahid Gulzar Padder
Eötvös Loránd University
Faculty of Informatics
Budapest

***Abstract*—Reinforcement Learning is finding its way in broader dimensions with the advent of time. Q Learning and SARSA are the building blocks for solving any problem in Markovian domain and thus enabling agent to perform optimally. To dive deeper into in this field classic games and Atari games play a significant role. In order to maximise score of an agent various parameter settings are performed. My approach is simple but robust in order to see the effect of varying the hyper parameters and other settings in order to achieve convergence. I have implemented Q and SARSA algorithms and compared them for three types of sensibilities i.e: α, ϵ and γ. I have also attached Jupyter notebooks for my approch.**

***Index Terms*—Reinforcement Learning, Q-learning, SARSA, AI Gym, Cartpole-v0 solution.**

I. Introduction

QLEARNING and SARSA are important algorithms and exhibit properties of asynchronous dynamic programming. These two Reinforcement Learning methods do not require information about the model. These two methods enable an agent to perform optimally in Markovian domain by facing the consequences of actions rather than building maps of the domain [1]. My motivation to select this problem: because it seems simple in working but it has a wide application ranging from robotics to marketing and much more.Policy can affect orientation of robot/customer base or next step in a virtual game etc. We first need to understand the environment we are dealing with. It is a cartpole or in simple words an inverted pendulum where it acts against the gravity. Here four things help in achieving a balanced pole:*-a) Cart position b) Cart velocity c) Pole angle d) Anglular velocity*. Fig. 1 depicts the Cartpole which consists of a cart and a pole.Here cart can move horizontally and pole exhibit rotational movement. Pole has a joint with cart and former starts to fall after sometime. Our task is to balance the pole with some policy.

Reinforcement learning(RL) finds many applications in diverse fields but there is still contradiction between exploration and exploitation strategy for action selection policy. On one hand SARSA has faster convergence but on the other hand Q-Learning has better overall performance. SARSA gets stuck in local minimum while Q learning takes longer time to learn.[2] Coming to a stronger and firm conclusion is out of scope for this brief assignment but these methods were used during solving the Cartpole-v0 environment.These two methods do not require any model knowledge. Unlike Monte-Carlo we don't need to wait for end of complete episode. In these methods update is made after each step. For both cases we have ϵ-greedy policy which is necessary for exploration. In both cases for every episode from current state s, action *a* is taken from *s* to new state *s'* keeping track of reward *r*. While ϵ-greedy policy is followed given by current Q-value function the action *a* is taken. And here we need to update *Q(s,a)* In SARSA, it is done by choosing an another action *a'* while following same current policy given above and *r+γQ(s',a')* as target.(2)

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a))] \quad (1)$$

SARSA is also known as *on-policy* learning as new action *a'* is chosen using same ϵ-greeedy policy similar to action *a*, the one which generated *s'* On the other hand in Q learning it is done by picking the **greedy action** a^g, In other words the action which maximizes the Q-value function in new state *Q(s',a)*:

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma Q(s',a^g) - Q(s,a))] \quad (2)$$

or similarly

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma max_u Q(s',a^g) - Q(s,a))] \quad (3)$$

Q-learning is also known by the name *off-policy* learning as the new action a^g is greedy in nature, not using current policy.

II. Literature Review

- Aaditya in [3] has modified the Cartpole-v0 environment with some noisy cases and controller based on Reinforcement Learning. Author has used six types of noises to force and neural network parameters for deep learning. Definition of 1 run= 1000 trials, 1 trial=60000 time steps and 1 time step: 0.02. One of the best strength of this approach is that you can achieve 100 percent success rate even if you introduce six different types of noises. I personally liked this approach and admire the clever technique. The only drawback in this approach is that it takes a lot of execution time.
- Nikhil in [4] has developed his own algorithm based on the Google's research paper: **Convolutional, Long Short-Term Memory, Fully Connected Deep Neural**

Shahid Gulzar Padder is with the Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary, e-mail: shahidgulzar4u@gmail.com
András Lőrincz is with the Faculty of Informatics, Eötvös Loránd University, Budapest, Hungary

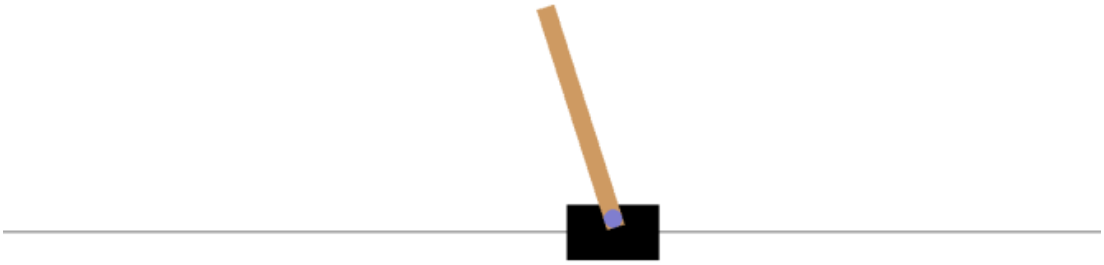

Fig. 1. Cartpole balancing

Network. He has combined convolutional, long short-term memory and fully connected networks according to Google's research paper but the author wasn't able to get as promising result as that of research paper despite using recent technique. I have implemented the solution with basic approaches but was able to achieve almost similar converging results.

- Ronn in [5] has used Q learning as well as Deep Q-learning agent. Here is the problem is assumed to be solved when average reward after 100 episodes is equal or greater than 195. For my case the average reward was 170 but sometimes it achieved closer results to this approach. It is obvious fact that this method is more technical than mine.
- Ferdinand in [6] has first choose epsilon and alpha parameters as constraints but achieved poor scores later on he used adaptive learning and was able to converge in 200 steps. I have converged in even lower steps for my approach using the same method and SARSA
- Adi in [7] has used four different approaches to solve the given environment: i) Random movements ii) Using weight vector iii) Using deep neural networks. For the first approach he has achieved pretty bad score than mine. For the second approach it achieved scores similar to mine. Third and fourth approches outperformed my approach.

III. Methods

- For my environment solution I have chosen Q-Learning and SARSA as these methods are discussed in lecture and require no model.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Fig. 2. Q Learning algorithm (Sutton Barto)

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(terminal\text{-}state, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Fig. 3. State-action-reward-state-action **SARSA** algorithm (Sutton Barto)

- Principle in Q learning is a fixed strategy.Strategy that we use differs from the strategy that we evaluate.
- Q learning is an online algorithm. It exploits actual experiences only i.e; older experiences are not used.
- On the other hand SARSA converges faster in general. Same strategy for exploration and exploitation.
- Lets explain both algorithms briefly.

Q-Learning is an off policy RL algorithm. Choosing a

strategy π we choose a greedy action in most of the time (Exploitation). We choose another action (Exploration). Q in Q-learning denotes quality. In other words how a given action **a** is fruitful in gaining some future reward. Immediate rewards are finite. Fig. 2 is Q learning algorithm [8]. In the next few steps I will show how we implement Q-learning algorithm.

- **Creating Q-table**: Whenever Q learning is performed we create a Q-table or which follows the shape of [state, action] and firstly we initialize values to zero. Updating and storing of *q-values* takes place after an episode. This table is very important for the agent to choose the best action according to the q-value.
- **Q learning and updates**: In the next step agent interacts with the given environment and thus making updates to the state action pairs in the environment.
  Taking action: *Exploration and Exploitation* An agent interacts with a given environment in of the two given ways. The first way is to use q-table as the reference and select the action based on the maximum value of those actions. This way is known as *Exploitation* because we use already available information to make a particular decision. On the other hand the second method is to take actions randomly and this approach is called *Exploration*. We do not consider here the max future reward but we select action randomly.Exploration is equally important as exploitation.We cannot discover new states if we always rely on exploitation. We need to have a balance between these two using ϵ and thus setting the value of how much you need to explore/exploit. After every step or an action updates occur and end when an episode is finished. Finishing refers to terminal point by agent. It can be anything ranging from ending of a game to completion of the desired objective. It is an obvious fact that the agent will not learn too much after a single episode but after various steps and episodes it will finally converge and learn the optimal q-values.
  There are three basic steps: i) Agent in state(s1) starts, then takes an action (a1) and a reward(r1) is received. ii) In the second step refers to q-table and selects either highest value(max) or random(epsilon) action. iii) In the third step updation of q-values takes place. We haven't yet mentioned about **Learning rate** α, **Discount factor** γ and **Reward** R as stated in eq(1,2,3).
  Learning Rate: Learning rate, also referred α, in simple words can be defined as how much take up new value as compared to the old value.
  Discount factor: Gamma or γ is known as discount factor. Its primary aim is to make a balance between immediate and future reward.
  Reward: After completing a certain action at a given state reward is received. A reward can come into existence at any time-step or specifically at the terminal time-step.

The second algorithm the that I have used is **SARSA**.*State-action-reward-state-action* (SARSA) is another famous algorithm to learn **MDP** policy. Whole algorithm is depicted in the Fig 3. It is clear from the name that the updation of q-value depends on the present state **s** , the action agent takes **a**, the reward **R** after choosing this action, the state **s'** in which agent enters after taking action **a**, and lastly action **a'** which agent chooses in new state.

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a))] \quad (4)$$

It is known as on-policy learning algorithm because SARSA interacts with a given environment and updates the policy π on the basis of actions taken.

Hyperparameters:

Learning Rate: Learning rate, also referred α, in simple words can be defined as how much take up new value as compared to the old value. 0 learning rate means will not learn anything and 1 will make it consider most recent information

Gamma: Gamma or γ is known as discount factor. Its primary aim is to make a balance between immediate and future reward. 0 factor makes agent opportunistic because of considering only current rewards. On the other hand 1 factor will make wait for the long term bigger reward.

Initial conditions (Q(s0,a0)): As we know that SARSA is an iterative algorithm, it assumes an initial condition before first updation occurs. A lower initial value can encourage exploration.

These two methods do not require any model knowledge. Unlike Monte-Carlo we don't need to wait for end of complete episode. In these methods update is made after each step. For both cases we have ϵ-greedy policy which is necessary for exploration. In both cases for every episode from current state s, action *a* is taken from *s* to new state *s'* keeping track of reward *r*. While ϵ-greedy policy is followed given by current Q-value function the action *a* is taken. And here we need to update *Q(s,a)*

IV. Experiments, Results and Discussion

My main aim of the software task is based on Python scripts and in these scripts I have done a comparative study of **SARSA** and **Q-learning**. But for my approach I have chosen to play around the hyperparameters. I have divided the sensibility into three categories.

- **EPSILON/** ϵ **sensibility**: Analysis of sensibility to the exploration rate
- **ALPHA/** α **sensibility**: Analysis of sensibility to the learning rate
- **DISCOUNT/** γ **sensibility**: Analysis of sensibility to discounting rate

I changed my mode of sensibility for both SARSA and Q-learning. I have used code from github but the author has done it only for Discount/ γ but I have extended it to ϵ and α as well. In the implementation the maximum number of episodes to converge is 6000.

Table I shows α sensibility for Q and SARSA approaches. Table data will vary after more iteration but here for convenience of page limit we used only five iterations. Per iteration I varied α to these values: (**0.1, 0.2, 0.3, 0.5, 0.7**) respectively. In case of Q learning with α=0.2 after two iteration and 270 episodes convergence was obtained. For SARSA after five iterations and α=0.1. Similarly for ϵ AND γ we can check the iterations and sensibility respectively. On the other hand we

TABLE I
Q LEARNING AND SARSA ALPHA (α) SENSIBILITY FOR FIVE ITERATIONS

It. No.	QL α val	Ep. to converge	S α Val	Ep. to converge
1	0.1	1175	0.1	657
1	0.2	1250	0.2	739
1	0.3	1103	0.3	458
1	0.5	301	0.5	945
1	0.7	456	0.7	591
2	0.1	659	0.1	494
2	0.2	270	0.2	469
2	0.3	600	0.3	368
2	0.5	746	0.5	410
2	0.7	580	0.7	611
3	0.1	633	0.1	524
3	0.2	469	0.2	910
3	0.3	859	0.3	1556
3	0.5	449	0.5	539
3	0.7	609	0.7	368
4	0.1	824	0.1	564
4	0.2	743	0.2	567
4	0.3	275	0.3	811
4	0.5	437	0.5	448
4	0.7	502	0.7	489
5	0.1	651	0.1	296
5	0.2	1525	0.2	770
5	0.3	397	0.3	737
5	0.5	476	0.5	927
5	0.7	329	0.7	380

TABLE II
Q LEARNING AND SARSA EPSILON (ϵ) SENSIBILITY FOR FIVE ITERATIONS

It. No.	QL ϵ val	Ep. to converge	S ϵ Val	Ep. to converge
1	0.001	605	0.001	1074
1	0.003	1107	0.003	1212
1	0.006	1314	0.006	378
1	0.01	434	0.01	808
1	0.013	1616	0.013	2622
2	0.001	1008	0.001	531
2	0.003	1136	0.003	396
2	0.006	1088	0.006	1420
2	0.01	696	0.01	872
2	0.013	430	0.013	736
3	0.001	123	0.001	692
3	0.003	614	0.003	634
3	0.006	525	0.006	130
3	0.01	701	0.01	1096
3	0.013	648	0.013	549
4	0.001	1149	0.001	453
4	0.003	703	0.003	862
4	0.006	1596	0.006	462
4	0.01	851	0.01	1430
4	0.013	1174	0.013	1571
5	0.001	390	0.001	1334
5	0.003	667	0.003	579
5	0.006	758	0.006	1352
5	0.01	1208	0.01	508
5	0.013	837	0.013	790

TABLE III
Q LEARNING AND SARSA GAMMA (γ) SENSIBILITY FOR FIVE ITERATIONS

It. No.	QL γ val	Ep. to converge	S γ Val	Ep. to converge
1	0.999	1104	0.999	439
1	0.8	596	0.8	699
1	0.7	1144	0.7	431
1	0.6	566	0.6	872
1	0.5	698	0.5	5999(NC)
2	0.999	1220	0.999	537
2	0.8	672	0.8	676
2	0.7	382	0.7	1227
2	0.6	269	0.6	2488
2	0.5	330	0.5	515
3	0.999	138	0.999	768
3	0.8	1156	0.8	690
3	0.7	607	0.7	447
3	0.6	554	0.6	2227
3	0.5	154	0.5	3141
4	0.999	155	0.999	1011
4	0.8	500	0.8	550
4	0.7	381	0.7	612
4	0.6	597	0.6	525
4	0.5	1079	0.5	917
5	0.999	543	0.999	144
5	0.8	570	0.8	958
5	0.7	786	0.7	338
5	0.6	727	0.6	5999(NC)
5	0.5	1516	0.5	1678

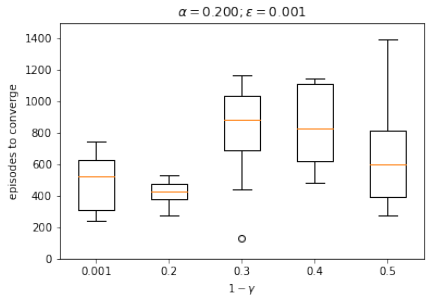

Fig. 4. Discount sensibility Q learning

can interpret box plots for all three sensibilities i.e: α , ϵ and γ. From the observations derived from the box plot performed best of discount sensibility of value **0.2** and worst for **0.5** learning rate. Similarly, we can interpret information about SARSA it performed worst for **0.5** and **0.6** discount sensibility. Agent wasn't able to converge even after 6000 episodes.Refer Fig 8, 6, 4, 9, 7, 5

V. CONCLUSION

I have been successful in converging my given environment and I tried to check all the possibilities of hyper parameters. All data about convergence, number of iterations required, number of episodes etc. is given in the Table I, II and III and respective box plots. In future I would like to improve my solution to faster convergence using Deep Q learning and Deep

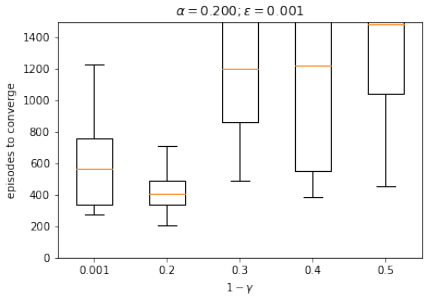

Fig. 5. Discount sensibility SARSA

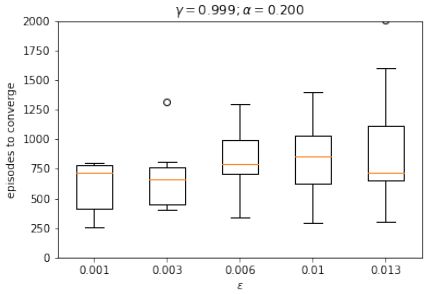

Fig. 6. Epsilon sensibility Q learning

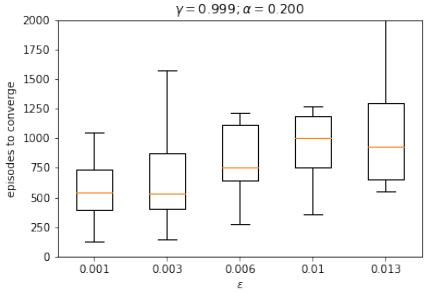

Fig. 7. Epsilon sensibility SARSA

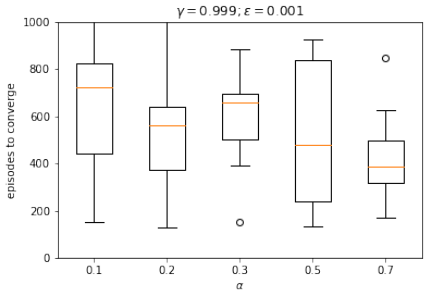

Fig. 8. Learning rate sensibility Q learning

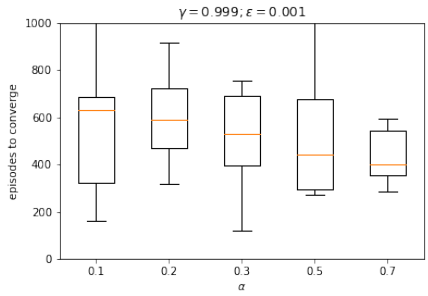

Fig. 9. Learing rate sensibility SARSA

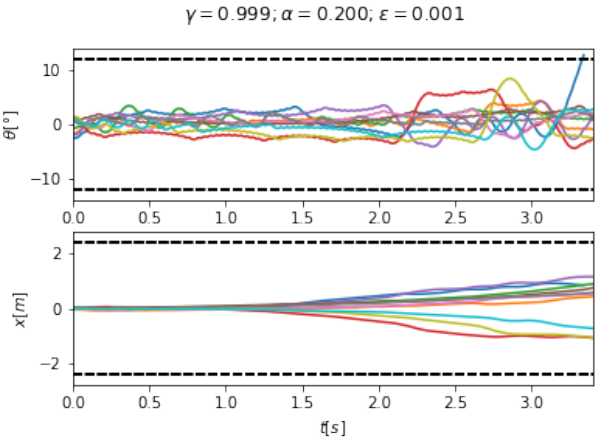

Fig. 10. Variable plot Q learning

neural networks. Currently, I had low computational power and time constraints.

ACKNOWLEDGMENT

The author would like to thank Prof. András Lőrincz for guiding us whole semester and giving us a chance to get hands on Reinforcement Learning.

REFERENCES

[1] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

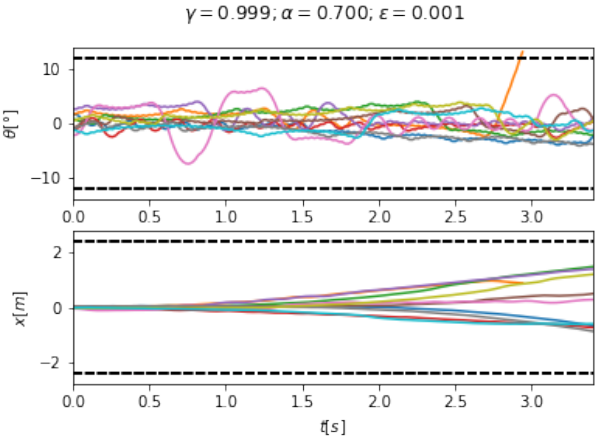

Fig. 11. Variable plot SARSA

[2] Y.-H. Wang, T.-H. S. Li, and C.-J. Lin, "Backward q-learning: The combination of sarsa algorithm and q-learning," *Engineering Applications of Artificial Intelligence*, vol. 26, no. 9, pp. 2184–2193, 2013.
[3] A. Patanjali, "Cartpolemod," *https://github.com/AadityaPatanjali/gym-cartpolemod*, 2017.
[4] N. Raghava, "Openai-cartpole-v0," *https://github.com/nikhilraghava/OpenAI-CartPole-v0*, 2017.
[5] R. Jacob, "Cartpole-v0," *https://github.com/RonnJacob/Cartpole-V0*, 2020.
[6] F. Mütsch, "Cartpole with q-learning," *https://dev.to/n1try/cartpole-with-q-learning—first-experiences-with-openai-gym*, 2018.
[7] A. Byte, "Cartpole," *https://github.com/adibyte95/CartPole-OpenAI-GYM*, 2018.
[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.