Université Jean Monnet

M1 End of year Project

Survey and Research Work Report

REINFORCEMENT LEARNING AND APPLICATIONS WITH THE OPENAI GYM TOOLBOX

Submitted By:
Karthik Bhaskar

UNIVERSITÉ
JEAN MONNET
SAINT-ÉTIENNE

Contents

1 Introduction

Reinforcement Learning also called adaptive dynamic programming is a branch in machine learning, In reinforcement learning the system in the environment learns about the environment by performing trail and error actions or methods in that particular environment, for each action performed the reward is given in the form of reinforcement signal which can be 0 or 1 or any real number which may be positive or negative.

The goal is to obtain the maximum number of reward by performing optimal actions which leads to the transition of states in the environment, this process is continued until the environment is reseted or terminal state in the environment is achieved.

Reinforcement learning is used as tool to solve decision making problems in a large and complex Markov Decision Problem(MDP) where the environment is dynamic with infinite timespan, as in MDPs when it is complex and large, due to the curse of dimensionality and curse of modeling[1] makes the classic dynamic programming inefficient therefore the adaptive dynamic programming is used.

The applications of Reinforcement Learning is used is various fields some of them are as follows,

- Stock Trading - Where the reinforcement learning algorithm such as Q-Learning learns the optimal trading strategy.
- Games - The algorithms such as Q-Learning, DQN are used in understanding the game environment, learns the optimal strategies to apply in that game in order increase the chances of wining. Many Reinforcement learning techniques are used to beat the humans in the games, one of the example is DeepMind's AlphaGo which out passed the world champion in the board game called Go and also these techniques are used in playing video games such as Atari.
- Robotics - In the field of Robotics the reinforcement learning techniques are used in object detection, learning to walk like humans, In manufacturing field, where the robots learns to assemble the parts of a car by trail and error methods.

1.1 MDP framework

The Markov Decision Problem(MDP) is a framework where the system in the environment is linked to Markov chain where the system jumps from one state to other on selecting a particular action in that state in random time and the transition of the state of the system from one state to another state depends on the current state and not the previous visited states. [1]

The MDP framework has following entities,

- Environment - It is space where the agents operates or the actions of the agent is applied. Environment can be a real world or a simulated world. For example,Atari game is an environment where the agent plays in that simulated world.
- State - The state space is the situation of the agent in the environment at a particular time, the state of the agent changes to different states depending upon the actions performed.
- Action - An action is the control or the response chosen by the agent in the particular state of the environment to maximize the reward or to solve a particular situation. For example, In the game of Super Mario an action can be jump when there is an obstacle.
- Reward - It is the response given by the environment for executing an action in a particular state where the transition of the state took place. A reward can be a positive or a negative real number.
- Agent - An agent is the program that performs actions in the environment to solve the problem in-order to maximize the reward.For example, In the game of Super Mario, the agent is the Mario bot program.
- Policy - A policy is the action chosen in a particular state.
- Performance Metrics - The performance metrics is the function to monitor the performance of the agent so that the optimal policy is chosen based on the reward.
  There are two performance metrics,
  - Discounted Reward[1] - It is a reward calculated as a sum of all discounted rewards achieved by the agent over a particular time

– Average Reward[1] - It is a reward expected by the agent at each step.

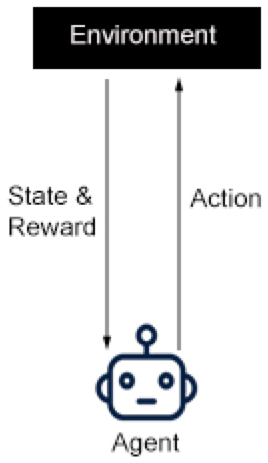


Figure 1: Example of MDP Framework

In the Figure 1 of the MDP Framework involves an Agent, Environment(E), set of finite States (S) and set of Actions(A) which is the subset of the States(S) and the reward(R).

In the MDP Framework the process works as follows,
The agent in the environment in a particular state "x"choses an action A from the set of actions "k" in the state "x" which is "A(k)" which acts on the Environment(E) then the Environment(E) sends a new state to the agent which leads to the transition to another state "y" along with the immediate reward R(x,A,y),for the transition from state "x" to "y" by choosing the action "A". As A(k) is action chosen then $\hat{A}$ becomes the policy selected by the agent.

1.2 OpenAI gym toolkit

OpenAI Gym Toolkit is a toolkit which allows the development and testing of Reinforcement Learning algorithms. This toolkit is developed by OpenAI which provides a common interface to different collection of tasks which is called Environment.

This toolkit concentrates on the system of episodes in Reinforcement Learning.Gym toolkit only provides the common interface to the Environments and does not provide the implementation of the Reinforcement Learning agent as this part is left for the implementor to develop agents with different RL techniques.

Gym toolkit provides two types of performance measures which is the final performance and sample complexity, the final performance is the average of rewards of each episode.[2]

The Environments provided by OpenAI Gym Toolkit is as follows,

- Classic control and toy text[2] : Provides implementation of basic jobs of RL
- Algorithmic[2]:Provides implementation of tasks involved in computation
- Atari[2]: Provides an environment to play Atari games
- 2D and 3D robots[2]:Provides an environment to carry out simulations of 2D and 3D robots

The documentation for installing OpenAI gym toolkit is provided in there website and their GitHub repository[3] . During the installation process of gym and required dependencies, I had created a virtual anaconda environment where everything required was installed.

2 Context of the project

The context of the project is a personal project to understand the concept of Reinforcement Learning. Making a survey of the state of the art algorithms in the field of Reinforcement Learning. To understand the concept of Reinforcement Learning in a better way the task was to experiment RL techniques on OpenAI gym toolkit. As this toolkit allows to compare Reinforcement Learning algorithms on different environments.

The theoretical part of the project was to survey the State of the art algorithms and understand the concept of Reinforcement Learning.

The practical part of the project was to provide experimental study of the application of Reinforcement Learning algorithms on the Atari environment of the OpenAI gym toolkit.

Different state of the art algorithms were surveyed during the process by referring to the papers written about the algorithms.

3 Survey of the state of the art

This section includes some of the well know Reinforcement Learning techniques and some of these techniques provides a benchmark results in RL and also when applied on Atari games. Some of these techniques are described in the following sections.

3.1 Q-Learning

Q-Learning is a Reinforcement Learning techniques which is model free technique[4] where it learns about the environment dynamically.

This techniques allows the agent to chose optimal policy in the MDP. The agent in the environment executes actions in a particular state and based on the immediate reward it learns the optimal action in that state, the agent tries all the available actions in the particular state and learns the best suitable action of that state based on the overall average reward.

Q-Learning uses a table which includes states and action taken in that state where the row is states and column is the action taken in state. It stores the values in the form of Q-values.

The Q-value is updated based on,

$$Q(s,a) = (1-\alpha) * Q(s,a) + \alpha * (r + \gamma * max(Q(s+1,a))) \tag{1}$$

The following are the steps[1] involved in Q-Learning.

1. Start
2. Initialize the Q-table with Q-Values equal to 0 where Q(s,a)=0. assign number of changes in the state "c" to 0 and start the simulation of to c_{max} number of iteration where the c_{max} is large enough
3. Starting with the arbitrary state (x) and select the action A of the state x with probability of 1/A(x). This allows to increase the Q-value in a greedy manner
4. After applying the action in the state "x" the transition of the state takes place to "y"and the reward is achieved which is r(x,A,y) and later increment the value of c

5. After the reward is achieved the Q-value is updated.

   $$Q(x, A) = (1 - \alpha) * Q(x, A) + \alpha * (r(x, A, y) + \gamma * max(Q(y, B))) \quad (2)$$

   Where B is another set of actions in state "y" in the equation 2

6. Check if $c < c_{\mathrm{max}}$ = True, assign the new state to "y" and go to step-3, else go to next step.

7. Select O(i) where "i" is a set of states in the space and O(i) should be the max(Q(i,B)) where B is the action and the policy selected is Ô by the algorithm.

8. Stop

3.2 SARSA

SARSA which means (state,action,reward,state,action) where the former S,A is older state and action and later is the next state and new action. It is a Reinforcement technique similar to Q-learning the difference is that in SARSA the Q-value is updated not based on max(Q(s+1,a)) but the Q-learning uses max(Q(s+1,a)) to update the Q-value. SARSA updates the Q-value using,

$$Q(s,a) = (1-\alpha) * Q(s,a) + \alpha * (r + \gamma * Q(s+1,a)) \tag{3}$$

SARSA is a on policy algorithm[1] and it uses episodic setting.

The following are the steps involved in SARSA as described in this paper[1]

- The Q-Values are initialized to random values
- For every episode, Repeat
  - Select a random state "x" and choose an action with limiting the exploration by greedy method
  - In each episode, repeat
    - Execute the action A with the transition to next state "y" with the reward of transition r and in the next state select new action "B"by limiting the exploration by greedy method and update the Q-value as,

      $$Q(x,A) = (1-\alpha)*Q(x,A)+\alpha*(r(x,A,y)+\gamma*Q(y,B)) \tag{4}$$

    - If the state "y"is the state of episode terminate then stop the episode or else continue the episode.

3.3 Deep Q-Network (DQN)

DQN[5] is a Reinforcement Learning algorithm that combines Reinforcement Learning with Deep Neural Network. This algorithm is step towards general artificial intelligence.

In this algorithm the agent in the environment interacts by the form of observation from previous actions, executing new actions and obtaining the reward from the environment. The main goal of this approach is to maximize the upcoming rewards.

The authors in the paper[5][6] states about the Algorithm as follows,
The agent acts upon the Atari emulator. The agents selects actions from available set of actions which is passed to the Atari emulator which changes the state in the environment and the score of the game.

As the agent is not aware about the internal state of the environment,It will become aware by seeing the pixel image obtained by the game and obtains the reward which is the occurred changes in the score.

The score of the game is dependent from actions executed previously and observations made and response of the actions made previously is obtained after 1000s of time-steps as the agent is only aware of the current screen and it is hard to analyze the response. Therefore the algorithm is provided with all the actions executed and observations made as an input. So the agent learns about the environment from the actions executed and the observations made. The environment is finite in termination.

Here the γ is used which is the future discounted reward which is equal to 0.99 and $R_t = \sum_{t\prime=t}^{T} \gamma t\prime - t r_{t\prime}$[5] where T is the time of termination and Q^* is the optimal policy which is calculated as maximum of the mapping of the solution policy to the executed action.The value of the executed action obeys Bellman optimality equation i.e. the action $a\prime$ is the optimal solution from the $s\prime$ which increases the reward.

In this algorithm a Q network is trained with the weights of θ to substitute the optimal goal value with the approximated goal values in every iteration. Here the goal values rely on the weights θ and the Q-learning algorithm is used with weights at each time-step and this technique is model free and policy free.

The Benchmark results in the paper[5] states that this algorithm was able to play and surpass the humans in almost 50 atari games and few of the games which performed below human level.

The agents where trained for 10 million time-steps and they were evaluated for every 250,000 time-steps.

3.4 Double Q-learning

Double Q-learning[7] is an off-policy Reinforcement Learning technique developed to overcome the drawbacks of Q-learning. According to the authors in the their paper[7] explains Q-Learning performs very poorly due to the overestimation of the action values as the Q-Learning technique uses maximum of value of an action as an approximation towards the expected value of action to be maximum.

Double Q-learning is developed by applying double Q-estimators to the Q-Learning technique. In this paper[7] Q-Learning is estimated as a single Q-factor to predict the value of the upcoming state which is max Q(s+1,A). In Double Q-Learning it stores 2 Q-functions which are Q^{A} and Q^{B} where each of the Q-function is updated by the value of the other Q-function for the upcoming state. Here to update the value Q^{A}, a^{*} and Q^{B} is used and similarly to update the value Q^{B}, b^{*} and Q^{A} is used.

The following is the algorithm given in the paper[7] for Double Q-Learning.

- Initialize the state "s", Q^{A}, Q^{B}
- Loop
  - choose an action "a" based on Q^{A} and Q^{B} of the state "s" and observe the reward and s′
  - Choose either to update A or B
    - If A is chosen to update then define the a^{*} = max Q^{A} of s′ and the action a and update the value for the Q factor $Q^{A}(s,a)=Q^{A}(s,a) + \alpha(s,a)(reward+\gamma\ Q^{B}(s\prime,a\ Q^{*}) - Q^{A}(s,a))$
    - If B is chosen to update then define the b^{*} = max Q^{B} of s′ and the action s. and update the value for the Q-factor of the $Q^{B}(s,a)$ as $Q^{B}(s,a) + \alpha(s,a)(reward+\gamma\ Q^{A}(s\prime,b\ Q^{*}) - Q^{B}(s,a)$
    - end if and else to update either A or B
    - update s with s′
- stop

4 Methodological contribution and results

As this was a learning phase for me I could not provide a personal solution in implementing the Reinforcement Learning techniques and also I did not have required GPU to train the models,So I referred the implementation of the Reinforcement Learning algorithms from other sources available from the Internet and tried to execute those implementations by myself. I will be providing the referenced implementations and its outcome here in this section. I have forked[8] the GitHub implementations of the repository.

4.1 DQN

One of the implementation for DQN was referenced from this GitHub repository[9] where this implementation was developed to make the agent play different atari games. This implementation includes the following parts

- This algorithm takes the screen pixel as an input, it uses a discounted reward which is 0.9 and uses a Q-function which has a value of an action in a particular state which stored in the form of Q-table. Q-learning method is used to calculate the Q-value based on this the agent choses an action to maximize the reward. To make this implementation work for wide range of games, the screen pixels is given as an input which provides the Q-values using a convolutional neural network.
- We can add any gym environment to this implementation
- In the observation state in this implementation the environment is started and the actions are performed to increase the Q-value and after performing the action, a state is received. this process is repeated until the episode terminates
- In the next step the the network is trained from the observations made which is called experience replay.
- In the end it outputs the reward from the execution

With this implementation I tried to learn how this implementation works with different environments and below are the screen shots of the output. And for few of the environment the agent did not respond.

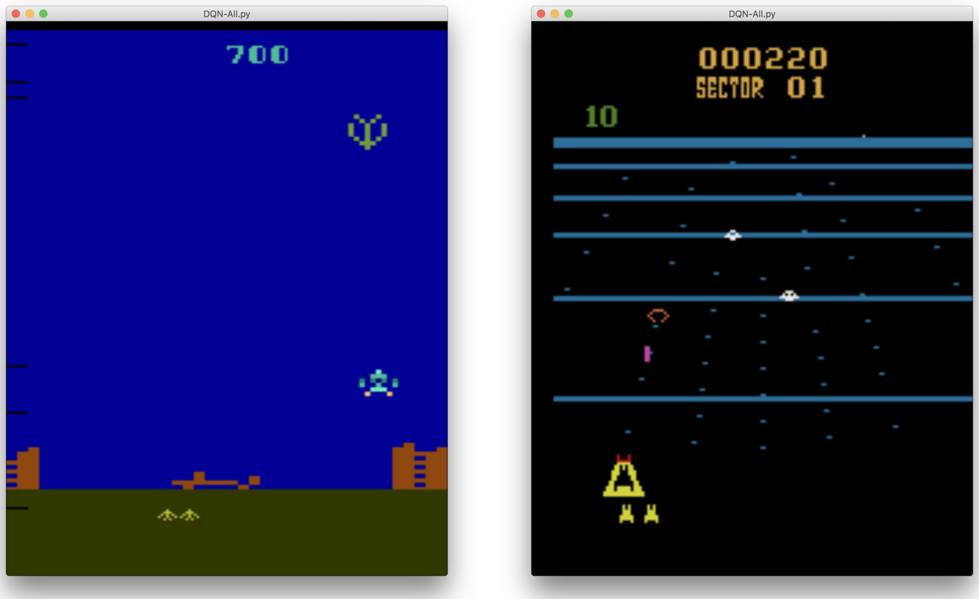


Figure 2: AirRaid-v0 and BeamRider-v0

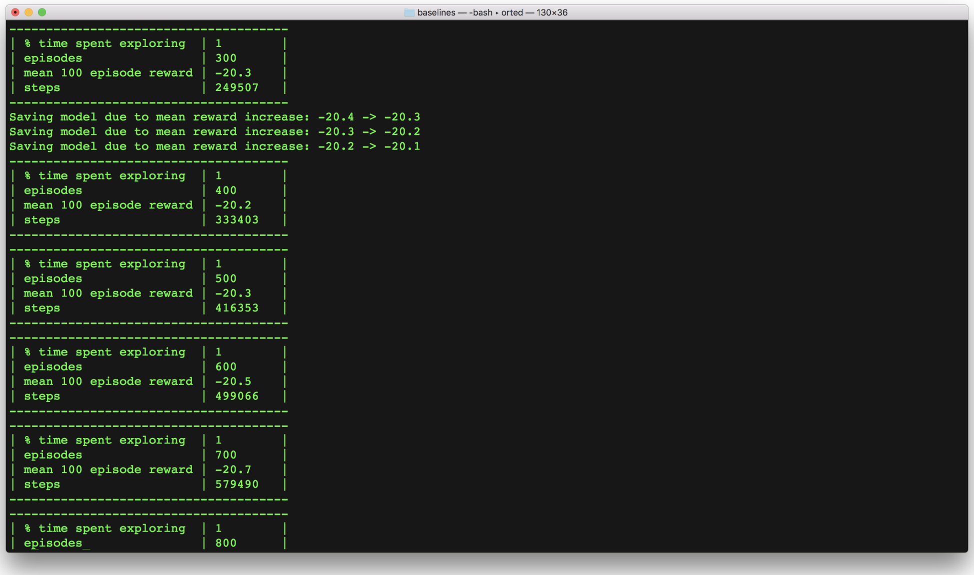
4.2 OpenAI Baselines

OpenAI Baselines [10] is a GitHub repository which provides the implementations of well known reinforcement learning algorithms.

I tried to learn in using there implementation of DQN and Proximal policy Optimization (PPO2)[11] but i did not have inadequate GPU resources for computing I was not able to train there implementation model as it was taking very long time on laptop. Below are the screen-shots in attempt to train their implementation.

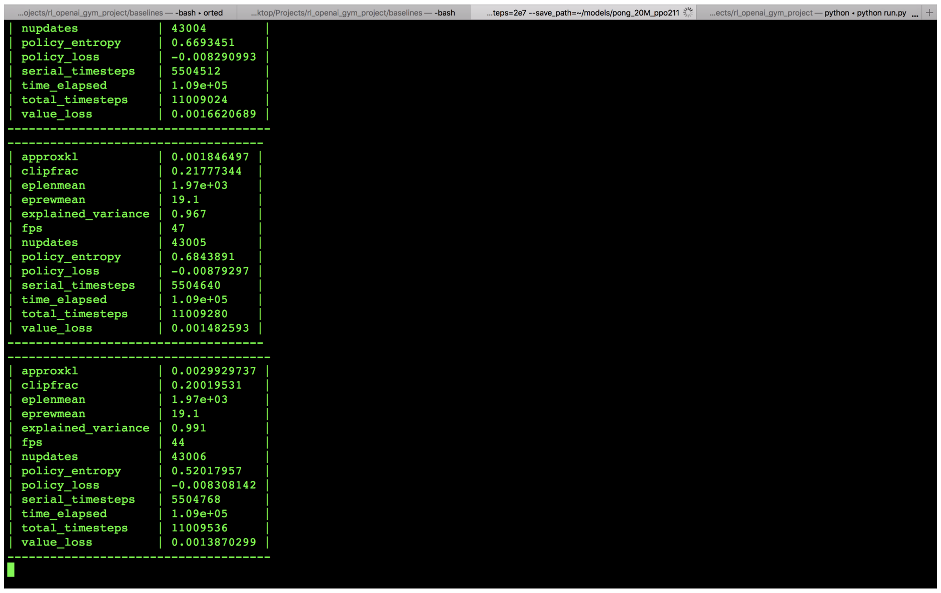
The first screen shot of the terminal (Figure 3) showed was to run the implementation of dqn algorithm on PongNoFrameskip-v4 for 1M time steps which provides the details of the process including time spent, number of episodes, mean of 100 episodes reward and time steps. As i could not complete this process as it was taking very long time, i could not view the agent playing.

The second screen shot of the terminal (Figure 4) showed was to run the implementation of ppo2 algorithm on PongNoFrameskip-v4 for 20M time steps and the information provided in the terminal during the process are approxkl, clip fraction, eplenmean, eprewmean, explained variance, frames per second, n updates, policy entropy, policy loss, serial timesteps which update*nsteps, time elapsed,total time steps which is update*nbatch and value loss.



```
baselines — -bash ▸ orted — 130×36
--------------------------------------
| % time spent exploring  | 1        |
| episodes                | 300      |
| mean 100 episode reward | -20.3    |
| steps                   | 249507   |
--------------------------------------
Saving model due to mean reward increase: -20.4 -> -20.3
Saving model due to mean reward increase: -20.3 -> -20.2
Saving model due to mean reward increase: -20.2 -> -20.1
--------------------------------------
| % time spent exploring  | 1        |
| episodes                | 400      |
| mean 100 episode reward | -20.2    |
| steps                   | 333403   |
--------------------------------------
--------------------------------------
| % time spent exploring  | 1        |
| episodes                | 500      |
| mean 100 episode reward | -20.3    |
| steps                   | 416353   |
--------------------------------------
--------------------------------------
| % time spent exploring  | 1        |
| episodes                | 600      |
| mean 100 episode reward | -20.5    |
| steps                   | 499066   |
--------------------------------------
--------------------------------------
| % time spent exploring  | 1        |
| episodes                | 700      |
| mean 100 episode reward | -20.7    |
| steps                   | 579490   |
--------------------------------------
--------------------------------------
| % time spent exploring  | 1        |
| episodes                | 800      |
```

Figure 3: DQN algorithm on environment PongNoFrameskip-v4


...ojects/rl_openai_gym_project/baselines — -bash ▸ orted
...ktop/Projects/rl_openai_gym_project/baselines — -bash
...teps=2e7 --save_path=~/models/pong_20M_ppo211
...ects/rl_openai_gym_project — python ◂ python run.py

```
| nupdates           | 43004        |
| policy_entropy     | 0.6693451    |
| policy_loss        | -0.008290993 |
| serial_timesteps   | 5504512      |
| time_elapsed       | 1.09e+05     |
| total_timesteps    | 11009024     |
| value_loss         | 0.0016620689 |
-------------------------------------
------------------------------------
| approxkl           | 0.001846497 |
| clipfrac           | 0.21777344  |
| eplenmean          | 1.97e+03    |
| eprewmean          | 19.1        |
| explained_variance | 0.967       |
| fps                | 47          |
| nupdates           | 43005       |
| policy_entropy     | 0.6843891   |
| policy_loss        | -0.00879297 |
| serial_timesteps   | 5504640     |
| time_elapsed       | 1.09e+05    |
| total_timesteps    | 11009280    |
| value_loss         | 0.001482593 |
------------------------------------
-------------------------------------
| approxkl           | 0.0029929737 |
| clipfrac           | 0.20019531   |
| eplenmean          | 1.97e+03     |
| eprewmean          | 19.1         |
| explained_variance | 0.991        |
| fps                | 44           |
| nupdates           | 43006        |
| policy_entropy     | 0.52017957   |
| policy_loss        | -0.008308142 |
| serial_timesteps   | 5504768      |
| time_elapsed       | 1.09e+05     |
| total_timesteps    | 11009536     |
| value_loss         | 0.0013870299 |
-------------------------------------
```

Figure 4: PPO2 algorithm on environment PongNoFrameskip-v4

The benchmark results provided in their website explains the comparison between different Atari games in 10 Million time steps, Here I will consider the 3 games and two algorithm which are deep-q and ppo2. in deep-q the mean is 745.27 and the scores of the games breakout,spaceinvaders and pong was 1.93, 483.35, -7.21 and in ppo2 the the mean is 2782.3 the scores of the games breakout,spaceinvaders and pong was 236.9, 959.5, 20.39 and the results obtained is little different compared to the obtained results from the paper due to subtle changes in the implementation.

5 Conclusion and perspectives

Reinforcement Learning is technique of trial and error where the agent program learns about the environment by executing actions in the form trail and error and achieves rewards. By this technique an agent will be able to solve the environment.

This technique is used by humans and animals in learning a particular task, now due to the advancement in artificial intelligence the algorithms can mimic the working of the brain in the form of neural network and with different reinforcement learning techniques, computer programs learns to do a particular task excelling the humans.

Algorithms like DQN, Q-learning which has achieved super human performance in playing games such as Atari, Go. And also Reinforcement Learning techniques is used in different sectors of the industry.

This project was a very good learning phase for me as I was able to understand the concept of reinforcement learning, different RL techniques, and to experiment with OpenAI gym toolkit.

References

[1] Abhijit Gosavi. Reinforcement learning: A tutorial survey and recent advances. *INFORMS Journal on Computing*, 21(2):178–192, 2009.

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

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[4] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[7] Hado V. Hasselt. Double q-learning. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2613–2621. Curran Associates, Inc., 2010.

[8] Karthik bhaskar github repository where the implementations are forked. https://github.com/Karthik-Bhaskar?tab=repositories.

[9] GitHub repository for the implementation of deep q learning. https://github.com/llSourcell/deep_q_learning.

[10] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Openai baselines. https://github.com/openai/baselines, 2017.

[11] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.