Reinforcement Learning algorithms in Acrobot environment (OpenAI's Gym) - Deep Machine Learning

Pongsakorn Chanchaipol
poncha@student.chalmers.se

Leelawadee Sirikul
sirikul@student.chalmers.se

***Abstract*—Reinforcement Learning is one of the important fields of Machine Learning. RL allows computer scientists to train machine learning models to learn how to react to a given environment. One of the most famous algorithms in the Reinforcement Learning field is Q-learning, which based on a concept of learning Q-value that measures how good is the action at a given state.**

In this project, we experiment with multiple Reinforcement learning techniques including Q-learning, Q-learning with modified reward, Double Deep Q-learning, Dueling Deep Q-learning. We train and evaluate these algorithms in the Acrobot environment from OpenAI's gym.

From the experiment, we found that Q-learning performs significantly worse than all of the four Reinforcement algorithms we chose. After the reward modification, Q-learning performs significantly better with more consistent performance. Double DQN and Dueling DQN have very similar performances which are better than both of the previous Q-learning variations.

1. Introduction

Machine Learning is one of the interesting fields of study in Computer Science and it has already been one of the key components that drive world's business to the future. Machine learning can be used to analyze the stock market, consumer behavior, etc.

One of the famous improvement steps in machine learning history is Deep learning. The architecture of Deep learning is implemented based on Human neuron cells. A Neural network is a combination of these neuron-like architectures that are combined together into a network. Deep learning allows computer scientists to analyze a much more complex relationship between the data. In order to step up closer from the data world to the real world, computer scientists tried to develop a Machine/Deep learning technique that can learn the whole environment like how humans learn things. Introducing Reinforcement Learning.

Reinforcement Learning allows machines to learn how to react to a given environment. Games are one of the closest examples to the real world since some games are very complex and can have almost infinite possibilities like the real world. If we can teach a machine to do some job in a game environment and it can perform that job very well, we can step up and use that same algorithm to teach a machine to do something in the real world.

In this project, we experimented with several classic Reinforcement Learning algorithms in the Acrobot environment from OpenAI's gym to see the performance and behaviour of each algorithm in the environment.

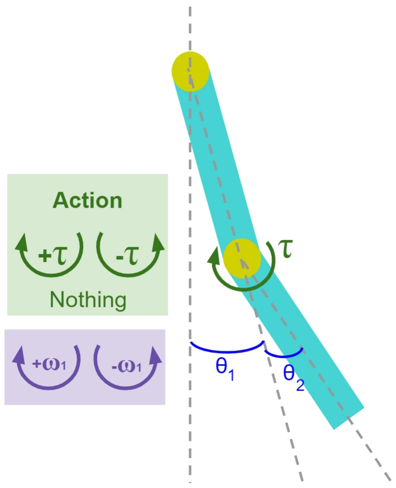

Figure 1: OpenAI's Gym: Acrobot environment

2. Background theory and Related Work

The concept of Reinforcement learning consists of two terms; agent and environment. The agent will try to learn how to react to the environment in order to maximize reward. For each action taken by an agent, the environment responds by returning the information of the next state and the immediate reward of the taken action. An environment usually consists of a large number of states which need different specific action to be taken by the agent in order to maximize the reward, so the agent will need to learn its own policy or how should it play the game to achieve the highest reward as possible.

Q-learning, one of the most fundamental yet powerful reinforcement learning algorithm, measure how good of each state-action pair by its Q-value, Q(s, a). The agent will improve its policy by adjusting these Q-values of each state-action pair and can find the "best" action at any state by choosing the action with the highest Q-value.

This project focuses on model-free reinforcement learning algorithms. We experimented with Q-learning, Q-learning with modified reward, Double Deep Q-learning with experience replay, and Dueling Deep Q-learning algorithms. These algorithms will be implemented on the Acrobot environment from OpenAI's gym.

The Acrobot environment consists of an Acrobot, which is a two-link robot, where the joint between two links is actuated. The goal is to swing up a two-link robot to a given height. The state of this environment is defined by $(\cos\theta_1, \sin\theta_1, \cos\theta_2, \sin\theta_2, \omega_1, \omega_2)$, which θ_1 and θ_2 are the current angles of each link, and ω_1 and ω_2 are the angular velocities of each link, respectively. The actions that can be taken in this environment are applying +1, 0, or -1 torques to the joint in between the two links of the acrobot. This means that the agent can only control the torque of the second link only. The goal height of this environment is 1, which is the length of one of the acrobot's links or half of the acrobot size (acrobot size is 2).

3. Method/proposed solution

Below this, you will find a description that explains the principal idea of each reinforcement learning method that we chose. The main task of this project to compare the performance and evaluate the result of each algorithm in the Acrobot environment.

Q-learning. Q-learning algorithm approximates and updates the action-values on the table of the state by action after training in order to approach the optimal action-value-function Q^*.

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_a Q(s',a) - Q(s,a))$$

where r is the reward get from moving from state s to state s', γ is the discount factor,$\gamma = 0$ means consider only immediate reward, $\gamma = 1$ means the agent considers future rewards are stronger, α is the learning rate $(0 < \alpha \leq 1)$

Double Deep Q-learning. The problem of the Q-learning algorithm is to overestimate action values and have an upward bias in $\max_a Q(s,a;\theta)$. To address these issues, Hado van Hasselt, Arthur Guez and David Silver [2] offered Double Deep Q-learning algorithm that uses online Q-network to select the best action for next state, and offline Q-network (target network) to evaluates the state-action values. Double DQN helps us train faster, obtain better policies, and have more stable learning.

$$Q(s,a) = r + \gamma Q(s', \arg\max_a Q(s',a;\theta); \theta^-)$$

where θ (online Q-network) for selecting the best action. θ^- (offline Q-network) for evaluating the best action.

Modified reward. We modify the reward by adding a normalized magnitude of the angular velocity (ω_1) into the reward. The reason for this modification is to encourage the model to increase the magnitude of angular velocity so that it reaches the goal height more frequently and possibly learning things faster.

$$reward_{modified} = reward + constant \times \frac{|\omega_1|}{(4\pi)}$$

Dueling Deep Q-learning. The dueling network consists of the state value function and the state-dependent action advantage function. The advantage is to generalize learning across actions without any modification to the underlying reinforcement learning algorithm.

$$Q_\theta(s,a) = v_\eta(f_\xi(s)) + A_\psi(f_\xi(s),a) - \frac{\sum_{a'} A_\psi(f_\xi(s),a')}{N_{action}}$$

where ξ, η, and ψ are, respectively, the parameters of the shared encoder f_ξ, of the value stream v_η, and of the advantage stream A_ψ; and $\theta = \xi, \eta, \psi$ is their concatenation. [2]

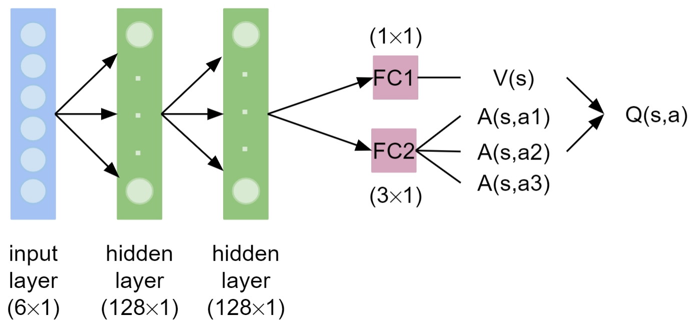

Figure 2: Dueling Deep Q-learning Network

Calculating the height. Acrobot is an environment consisting of 2 links, the length of each is equal to 1. The goal is to reach a height of 1. The original formula to calculate the height from the OpenAI gym's acrobat.py is: $-\cos\theta_1 - \cos(\theta_1 + \theta_2)$

But the state we received is in the form of $(\cos\theta_1, \sin\theta_1, \cos\theta_2, \sin\theta_2, \omega_1, \omega_2)$, Since we don't have $\cos(\theta_1 + \theta_2)$, so we need to compute it by using this formula:

$$\cos(\theta_1 + \theta_2) = \cos\theta_1 \cos\theta_2 - \sin\theta_1 \sin\theta_2$$

Applying the previous formula, we can now calculate the height of the given state like below:

$$height = -\cos\theta_1 - (\cos\theta_1 \cos\theta_2 - \sin\theta_1 \sin\theta_2)$$

4. Results/discussion

4.1. Performance analysis

Model	Average successful steps(percentage)	Average first successful steps
Random policy	0	-
Q-learning	64.5 (12.90%)	178.3
Q-learning (modified reward)	80.6 (16.12%)	101.95
Double DQN	83.5 (16.70%)	106.55
Dueling DQN	85.65 (17.13%)	75.2

TABLE 1: Average successful steps comparison

We will compare the performance of each approach by their Average number of successful steps, and the average first successful step (how many steps does it take to reach the goal height for the first time).

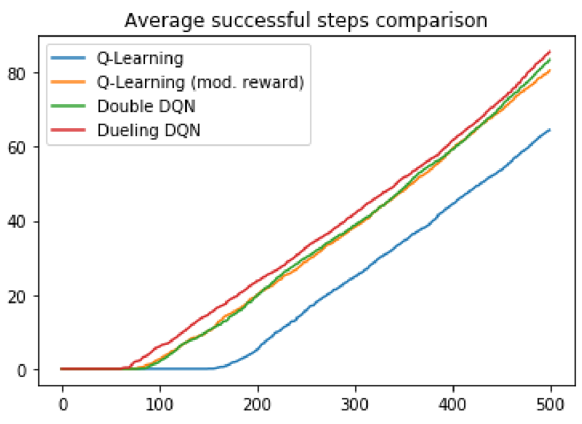

Figure 3: Average successful steps comparison plot

The average number of successful steps will show us how consistent the algorithm reaching the goal height in a finite number of steps. The higher the number of successful steps, the better the algorithm. On the other hand, the average first successful steps will show us the least amount of steps that each algorithm needs in order to reach the goal height for the first time. The lower this measurement is, the faster the algorithm can reach goal height.

These measurements were obtained by running the algorithms in the Acrobat environment for 10 episodes which have 500 steps each. Then, we will track the number of steps that reach the goal height and also keep tracking the lowest steps that each algorithm used in order to reach the goal height. Then, we average the results of these 10 episodes to reduce the effects of the noise of the environment.

From TABLE 1, we can see that the random policy cannot reach the goal height even once. This is understandable since the Acrobot environment really needs a proper way of handling the movement of the two links.

For the Q-learning algorithm, it has the worst performance out of all algorithms we experimented with. It got an average of 12.90 percents of successful steps which is significantly lower than others. Moreover, it needs an average of 178.3 steps in order to reach the goal height for the first time, which is about twice as much of the others. After the reward modification, it performs a lot better but still not as good as the more complex algorithms like Double DQN and Dueling DQN.

For the Double DQN and Dueling DQN, they got the best performance of all the algorithms we try here. Double DQN slightly beat Dueling DQN in the average successful steps category, but the Dueling DQN considerably beat Double DQN in the average first successful step category.

4.2. Policies analysis

In this section, we will analyze the behaviour of each algorithm by looking at their policy plot. In these plots, the chosen actions are distinguish by colors i.e. action +1:Red, 0:Green, and -1:Blue. X-axis and Y-axis represents the coordinate of the first link of the Acrobot, while Z-axis represents angular velocity of the first link.

From figure 4, that is Q-learning policy, we will see that the algorithm has a balanced distribution between +1 and -1 action, while 0-action is randomly appear throughout the plot with no obvious pattern.

For the Figure 5,6,7, which are from Q-learning (modified reward), Double DQN, and Dueling DQN algorithms, the patterns of the policies are quite similar. These models mostly chose action -1 and 0 but don't choose much action +1. Which are totally different from the policy from Q-learning that has balance between action +1 and -1.

Since we got a much better performance in Q-learning (modified reward) than the normal Q-learning while the only difference is the reward, we assume that Q-learning is trying too much to balance between action +1 and -1 without much concern about losing velocity. After the reward modification, the agent knows that it will get more reward if it move faster and it may found that focusing on one action results in faster velocity. This could mean that, in this Acrobot environment, using only action -1 and 0 results in faster angular velocity than using all actions including action +1.

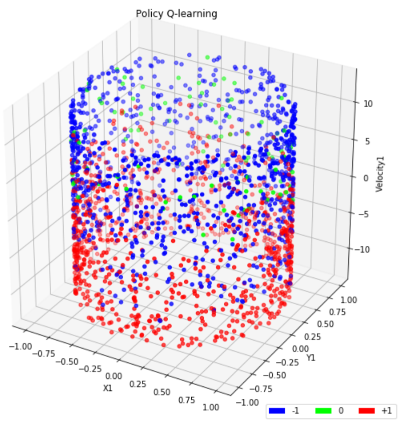

Figure 4: Q-learning policy

5. Conclusion

The results of this experiment turn out like our expectations. The random policy cannot achieve any success in this environment. Q-learning performs okay but can be significantly improved by modifying the reward to encourage the algorithm to increase velocity. For the Double DQN and Dueling DQN, both of them perform really well, but Dueling DQN is slightly better due to its slightly better performance and significantly shorter training time. From the policies analysis, we also found that angular velocity might have some influence to the performance of the model. It is one of the main reasons that make Q-learning with modified reward performs much better than its original version. For further improvement, we could try other reinforcement learning algorithms such as Noisy DQN, Distributional DQN, Prioritized DDQN, and Rainbow.

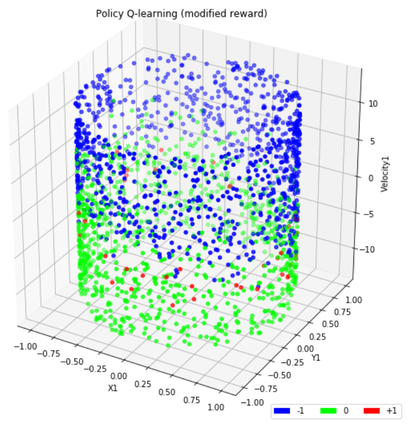

Figure 5: Q-learning policy (modified reward)

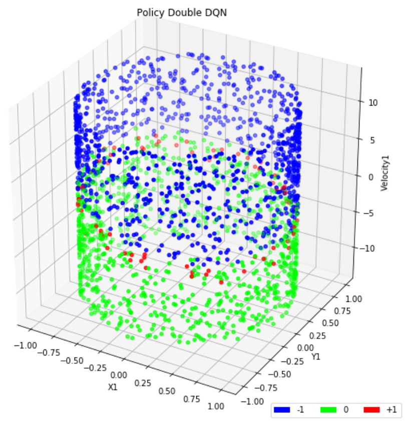

Figure 6: Double DQN policy

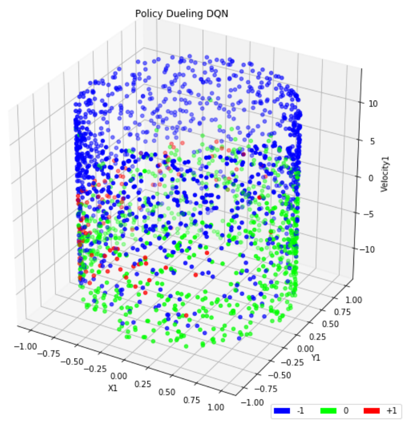

Figure 7: Dueling DQN policy

References

[1] OpenAI's Gym: Acrobot-v1 environment, https://gym.openai.com/envs/Acrobot-v1/

[2] Hessel, Matteo, Modayil, Joseph, van Hasselt, Hado, Schaul, Tom, Ostrovski, Georg, Dabney, Will, Horgan, Dan, Piot, Bilal, Azar, Mohammad, AND Silver, David. "Rainbow: Combining Improvements in Deep Reinforcement Learning" AAAI Conference on Artificial Intelligence (2018): n. pag. Web. 23 Oct. 2019

[3] Philtabor: Solving Mountain Car with Q-Learning, https://github.com/philtabor/Youtube-Code-Repository

[4] Tim: Solving Mountain Car with Q-Learning, https://medium.com/@ts1829/solving-mountain-car-with-q-learning-b77bf71b1de2

[5] Parsa Heidary Moghadam: Deep Reinforcement learning: DQN, Double DQN, Dueling DQN, Noisy DQN and DQN with Prioritized Experience Replay, https://medium.com/@parsa_h_m/deep-reinforcement-learning-dqn-double-dqn-dueling-dqn-noisy-dqn-and-dqn-with-prioritized-551f621a9823

[6] Tavakoli, Arash, Pardo, Fabio, AND Kormushev, Petar. "Action Branching Architectures for Deep Reinforcement Learning" AAAI Conference on Artificial Intelligence 2018. Available at: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17222/16687

[7] Jonathan Hui."RL — DQN Deep Q-network". https://medium.com/@jonathan_hui/rl-dqn-deep-q-network-e207751f7ae4