Reinforcement Learning Final Project

Shisen Yue

June 8, 2023

Abstract

With the surge of technological advancement and the maturation in theory , The Reinforcement Learning algorithms has marched into a new era. The involvement of deep learning methods allows for the adaptiveness to high dimensionality, and also sparks numerous ideas in developing the classic Reinforcement Learning algorithms. This study selects some ingenious Model-free algorithms to verify the previous research and attempt to make improvements on them. Specifically, We implement and improve the DQN and Dueling DDQN algorithm as the value-based methods and DDPG and TD3 as the policy-based methods to separately train the agents in environments of discrete and continuous space. The agents excel in the selected environments and by which we illustrate our insights into the algorithms and the environments.

1 Introduction

Reinforcement learning has received broad attention over the past few years. It has been applied to many fields, such as robotics, game playing, and natural language processing. The ChatGPT has indeed arouses people's attention to Artificial Intelligence, and the Reinforcement Learning from Human Feedback (RLHF) is an indispensable part in this revolution.

Its success is closely related to the approximation methods with neural networks. By infusing the deep learning methods into the classic reinforcement learning algorithms, researchers are able to train the agents in the environment with continuous action and observation space that highly simulates the real world. In this training process, the process of agents either planning for an optimal policy or interacting with the environment to get the feedback for updating their policy. The former strategy model-based and the latter model-free, which differ in the beforehand knowledge about the transitions and rewards in the environment.

One of the main difference between these two types of algorithms is the sample efficiency. Model-based RL algorithms are crucial in making RL data-efficient and in trading off exploration and exploitation (Kai Arulkumaran, 2017). However, a ground-truth model is usually unavailable to the agents, so their performance in the training might be largely different from that in the real world. Moreover, such an approach requires huge computing resources, making it hard to be conducted. In contrast, the Model-free algorithms are easier to implement and tune, which accounts for the extensive research on it.

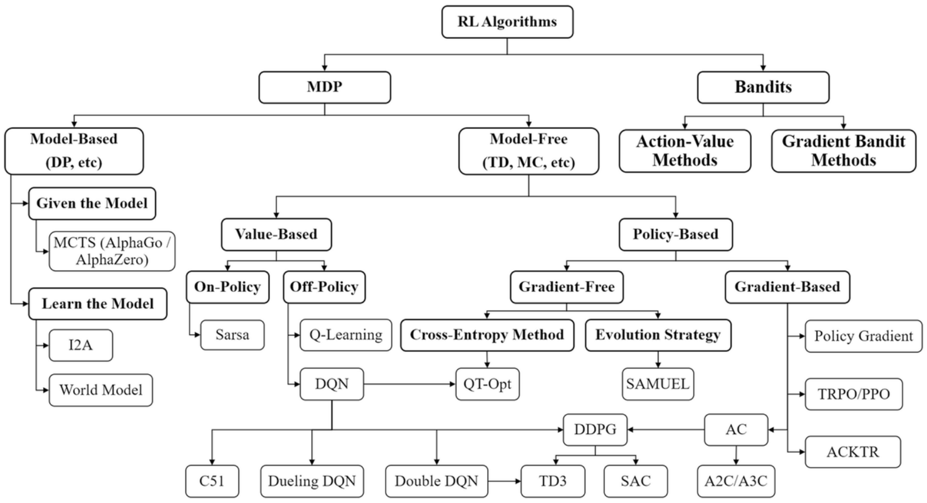


Figure 1: Taxonomy of RL algorithm.

Within the classification of Model-free RL algorithms, two separate methods are applied to predict the policy. The value-based algorithms compute the state values and action values and lead the agent to the direction with high values, and the policy-based algorithms directly evaluate the policies with gradient descent. There are also hybrid methods that combine the advantages of these two methods.

In this study, we focus on the value-based algorithms and policy-based algorithms. We implement the Dueling DQN and DDPG algorithms and test them on the Atari-2600 and Mujoco environments. We also apply the tricks to improve the performance of these algorithms, including the weight initialization, multi-steps, prioritized experience replay, and HER.

2 Value-Based Algorithms

2.1 Background

2.1.1 Value Functions and Q-learning

The core idea of the value-based methods is to evaluate the value of states so as to give instructions to the action choices. According to the value function,

$$V^{\pi}(s) = \mathbb{E}[R|s, \pi]$$

and once we have the maximum value of state V^* among all the states, the optimal policy is computed by choosing the action a available in s_t that maximizes

$$\mathbb{E}_{s_{t+1} \sim \tau(s_{t+1}|s_t, a)}[V^* s_{t+1}]$$

However, since the transition dynamic *tau* and the rewards are unknown in the model-free environment, we rely on the *state-action-value* or *Q-value* to evaluate the policies.

$$Q^{\pi}(s,a) = \mathbb{E}[R|s,a,\pi]$$

2.1.2 Deep-Q Learning

In solving high dimensional problem, it's nearly impossible to maintain a table for each $Q(s,a)$ value. Hence the neural networks plays it role as the estimator of V^*, Q^* and A^* in finding the optimal policy π^*. In DQN[1], the estimation of $Q(s,a)$ is through the network $Q(s,a;\theta)$ and the update of the network by minimising the loss function $L_i(\theta_i)$ that changes at each iteration i,

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}[(y_i - Q(s,a;\theta_i))^2]$$

where $y_i = \mathbb{E}_{s'\sim\epsilon}[r + max_{a'}Q(s',a';\theta_{i-1}|s,a]$ is the target and the difference between y and $Q(s,a;\theta_i)$ is called temporal difference error (TD error).

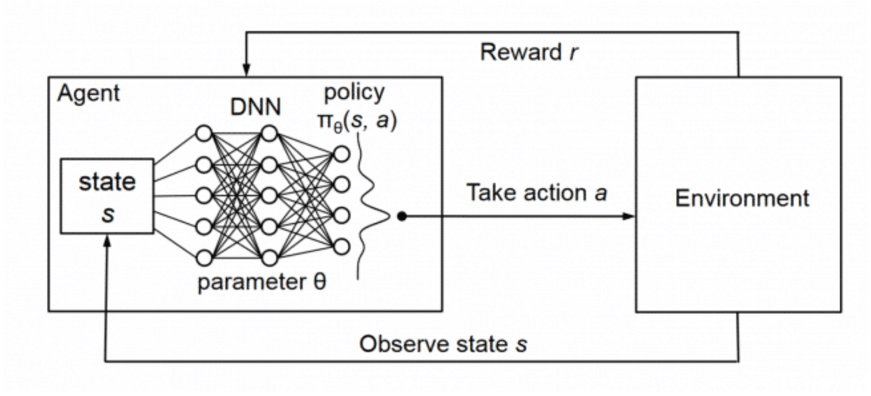


Figure 2: A demonstration of using neural network to evaluate action values and take actions accordingly.

The DQN addresses the fundamental instability problem of using function approximation[2] by the use of two techniques: the target network and the experience replay[3].

The use of target network was introduced by Minh et al.[1]. In updating the policy network, the weights in the target network which predicts y are frozen and is synchronized to the policy network after a certain number of steps. This avoid calculating te TD error based on the rapidly fluctuating estimates of the Q-values from the policy network, and this largely increase the stability in training.

The experience replay indicates the technique that the agent store the transition of each step $(s_t, s_{t+1}, a_t, rt+1)$ in a cyclic buffer, and every time learns from a batch of experience retrieved from this queue. It massively reduces the amount of interaction needed with the environment, controls the variance in learning and breaks the temporal correlations that may adversely influence RL algorithms. While it's widely acknowledged as a technique applied on a model-free algorithm, some researchers regard it as a model[4]. Both of these two techniques increase the stability of the algorithm and are continued to be used in the variants of DQN and other subsequent algorithms.

2.2 Methodology

2.2.1 Algorithm

In this study, we use DQN and one of its variant, Dueling-DQN[5] to train the agent in an environment with discrete observation and action space.

The DQN algorithm is implemented based on the Pseudo-code Algorithm 1 from the study[6].

Algorithm 1 Deep Q-learning with experience replay

```
Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
for t = 1, T do
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi = \phi(s_1)$
    for t=1, T do
        With probability $\epsilon$ select a random action $a_t$.
        otherwise select $a_t$ $\max_a Q^*(\phi(s_t+, a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$.
        Store transition $(\phi_t, a_t, r_t, \phi t + 1)$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        set $y_j = \begin{cases} r_j, & \text{for terminated } \phi_{t+1}. \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi t + 1. \end{cases}$
        Perform a gradient descent step on $(y_i - Q(\phi_j, a_j; \theta))^2$
    end for
end for
```

The implementation of Dueling-DQN follows the same structure with DQN, with the only modification on the network structure and the TD error computation.

2.2.2 Implementation details

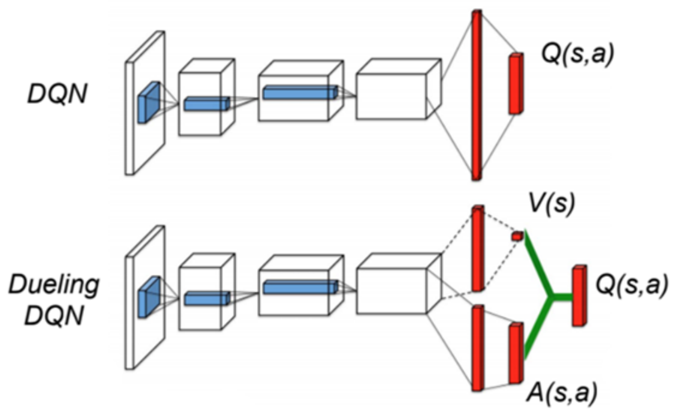


Figure 3: The neural network structures in DQN and Dueling-DQN. This demonstration is from the original study by Wang et al.[5].

We apply the same hyperparameters to DQN and Dueling DQN. In particular, we linearly schedule the decay of ϵ to allow the agent gradually relies more on its exploitation. In our pilot experiment, the agent with linear scheduled ϵ outperforms that with exponential scheduled ϵ, so we choose the former one. A warm-up process with 10000 steps is added before the training process, in which agents choose action randomly and the networks keep unchanged.

2.2.3 Performance Enhancement Techniques

In training the agents, we apply the following techniques to accelerate the convergence and enhance the stability.

1. **Neural network weights initialization**

   This is a general method for neural network training. We apply He initialization[7] to our network and see significant change in convergence. It initializes the weights of a layer by sampling from a normal distribution with zero mean and a variance calculated based on the size of the input and output dimensions of that layer. It takes into account the characteristics of the ReLU activation function and scales the variance appropriately.

2. **The Dueling network and TD error computation**

   The neural network in Dueling DQN provides an effective approach to enhance the performance of DQN agents. As the Figure 3 demonstrates, the action values output from Dueling network is composed of value function (V) and the advantage function. The formula for Q-values is

   $$Q(s,a,;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,A';\theta,\alpha)$$

   This helps agent ignore the irrelevant actions in some states and effectively addresses the overestimation problem.

   The computation of TD error is the same as that in Double DQN, proposed by van Hasselt et al.[8], which is written as

   $$y_i^{DDQN} = r + \gamma Q(s', \arg\max_{a'} Q(s',a';\theta_i);\theta^-)$$

   This also helps in mitigating the overestimation of Q-values.

3. **Prioritized experience replay**

   The prioritized experience replay assigns a weight to each experience to order their importance for the training of agents. This helps us draw information more frequently from those more valuable experience that are more likely to drive the agent to behave better.

4. **Multi-step learning** In Q-learning, we calculate the TD error based on the rewards and the Q-values of the next state-action pair and the current state-action pair. The multi-step learning adopts the Q-value of several steps after as the target, to allow a further forward view

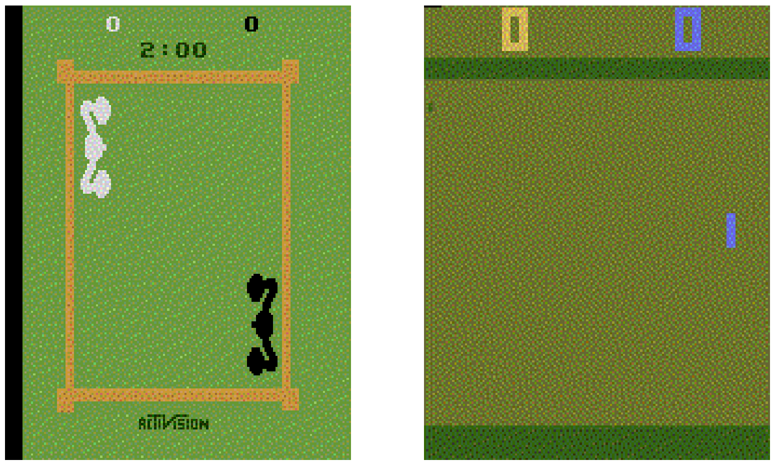
for the selection of action in the current state. The optimization object is thus changed to

$$(R_t + \gamma_t \max_{a'} Q_\theta^-(S_{t+n}, a') - Q_\theta(S_t, A_t))^2$$

2.2.4 Environment settings

Atari games and the OpenAI Gym Atari environment provide a popular and challenging platform for training reinforcement learning agents. The Atari game library consists of a collection of classic arcade-style games originally released on the Atari 2600 console. These games include well-known titles like Space Invaders, Breakout, Pac-Man, and many more.

In this study, we select the following two games to test our algorithms.



(a) BoxingNoFrameskip-v4 (b) PongNoFrameskip-v4

Figure 4: The rendering of OpenAI atari games

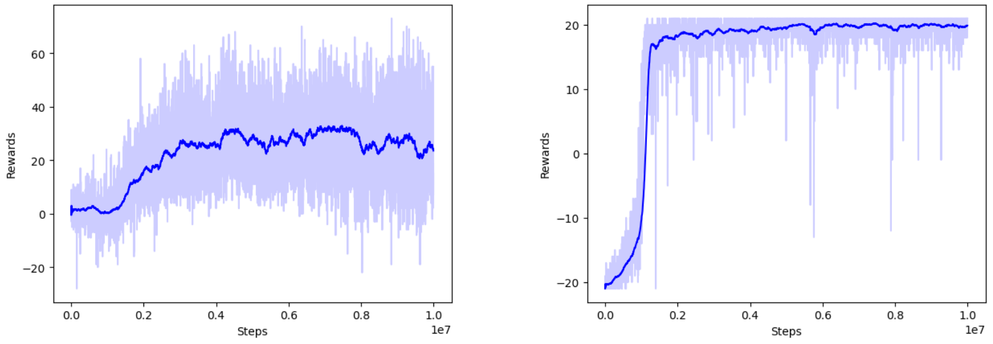
We conduct a series of preprocessing to the testing . In particular, the original size of $(210, 160)$ is downsampled to $(84, 84)$ and the reward is clipped into $\{-1, 0, 1\}$. Besides, we add the deepmind-style configurations to the games. These operations make the training process more time and memory efficient.

2.3 Results and Discussion

We train the DQN agent and the improved Dueling-DDQN agent in the **BoxingNoFrameskip-v4** and **PongNoFrameskip-v4** and collect their scores, displayed in the following table and figures.

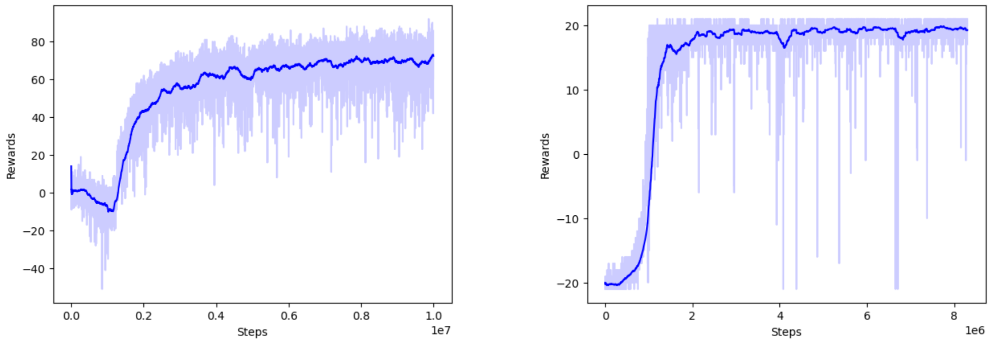
	Dueling-DQN score	**DQN score**
PongNoFrameskip-v4	21.0	20.2
BoxingNoFrameskip-v4	79.3	73.0

Table 1: Best scores of the Dueling-DQN agent and the DQN agent in the two games



(a) BoxingNoFrameskip-v4 (b) PongNoFrameskip-v4

Figure 5: Scores of the DQN agent in the two games. The dark and light blue lines separately correspond to the average rewards of the past 100 episodes and the current episode rewards



(a) PongNoFrameskip-v4 (b) BoxingNoFrameskip-v4

Figure 6: Scores of the Dueling DDQN agent in the two games

We evaluate the DQN and Dueling DDQN agents by their scores. In order to depict the tendency of score change more clearly, we compute at each episode the average rewards of its past 100 and make differentiation with the alpha value of colors. Table 1 suggests that the two agents obtain similar best scores in both the games. Figure 4 shows that the Dueling DDQN agent converges well in the two games with low variance, and the DQN agent obtains comparable performance in the PongNoFrameskip-v4 and less satisfactory in BoxingNoFrameskip-v4 with low scores and high variance.

We prefer to explain their similarity in the game PongNoFrameskip-v4 as due to the attribute of the environment, while the difference in the game BoxingNoFrameskip-v4 demonstrates the difference in stability between the two agents. This is consistent with the results in the paper[5] and justifies the rationality of the improvement we expect to see in the improved Dueling DDQN agent.

We also verified the fact that initialization matters in training the Reinforcement Learning agent, with the comparison displayed in Figure 7.

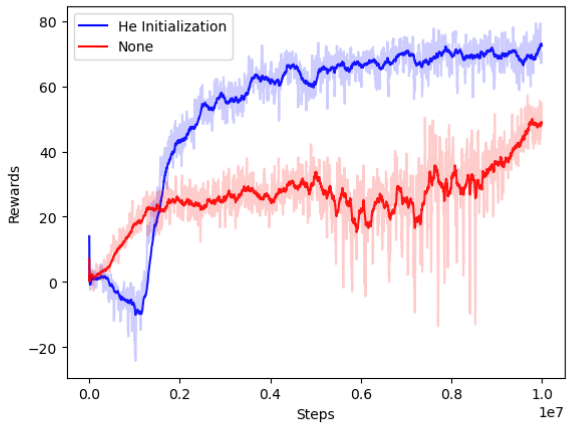


Figure 7: Scores of Dueling DDQN agent with and without He initialization[7], The Dark and light color separately represents the average rewards of the past 100 and 10 episodes.

3 Policy-Based Algorithms

3.1 Background

The Policy Gradient theorem[9] is a fundamental result in reinforcement learning that provides a way to update the parameters of a policy to maximize the expected return. The theorem states that the gradient of the expected return with respect to the policy parameters can be computed as an expectation over the states and actions, weighted by the advantage function.

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}[\nabla_\theta \log_{\pi_\theta}(a|s)Q^\pi(s,a)]$$

However, the Policy Gradient theorem is originally formulated for stochastic policies, where actions are chosen according to a probability distribution. In continuous action spaces, it is challenging to directly apply the theorem to optimize deterministic policies, which directly output specific actions.

The DPG algorithm[10] addresses this challenge by introducing the concept of action-value gradients. Instead of directly computing the gradient of the expected return with respect to the policy parameters, DPG computes the gradient of the expected action-value with respect to the policy parameters.

$$\theta^{k+1} = \theta^k + \alpha[E]_{s\sim\rho^{\mu^k}}[\nabla_\theta\mu_\theta(s)\nabla_a Q^{\mu^k}(s,a)|_a = \mu_\theta(s)]$$

By applying the chain rule, this gradient can be expressed in terms of the policy gradient.

The DPG algorithm leverages this relationship to optimize deterministic policies. It learns a critic network, also known as an action-value function or Q-function, to estimate the expected action-value given a state and an action. The critic network provides the gradient information necessary to update the policy parameters through the policy gradient. It consists of two main steps: the actor update and the critic update. In the actor update, the policy parameters are updated in the direction of the estimated action-value gradient. In the critic update, the critic network is updated using temporal difference learning to minimize the mean squared error between the estimated action-value and the observed returns. By iteratively updating the policy and the critic network, the DPG algorithm seeks to find the optimal deterministic policy that maximizes the expected action-value. The Figure 8 demonstrates this process.

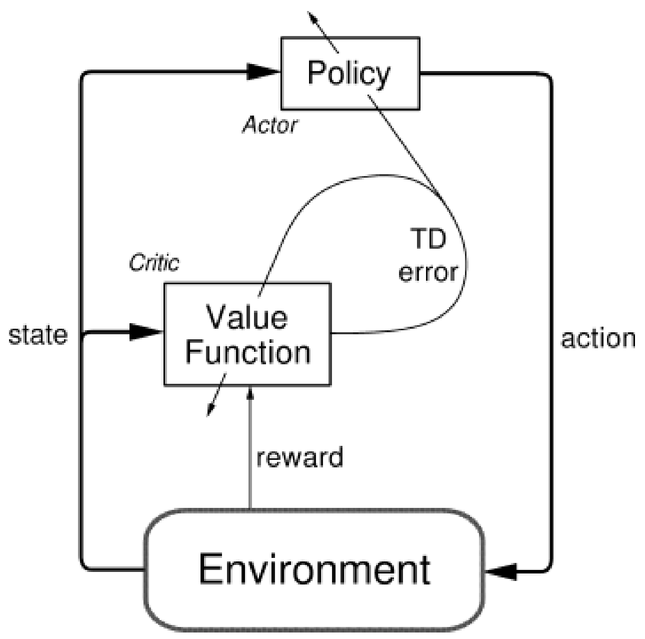


Figure 8: Demonstration of Deterministic Policy Gradient (DPG) algorithm. The DPG agent takes actor through the actor network and receive observation and rewards from the environment to update its critic network

3.2 Methodology

3.2.1 Algorithm

In this study, we use Deep Determinsitc Policy Gradient (DDPG)[11] and its improved version Twin Delayed DDPG (TD3)[12] to train the agents to play in the environment with continuous observation and action space.

DDPG algorithm builds upon the Deterministic Policy Algorithm (DPG), integrating the experience replay and a noise in selecting action. This combination of off-policy and actor-critic method, inheriting both the stability from actor-critic methods and the efficient use of data from off-policy methods.

While the DDPG algorithm reached a well combination of the off-policy and actor critic methods, it left the overestimation problem unsolved. As noted by Fujimoto et al.[12], the overestimation of Q-values exist in a list of powerful algorithms including DDPG. The researchers made the following modifications on the DDPG algorithm to address the overestimation problem.

1. **Clipped Double Q-Learning**

It uses two separate critic networks instead of one in DDPG. During training, the minimum value between the two critics is used to determine the target value for updating the actor and critics:

$$y_1 = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \pi_{\phi_1}(s'))$$

This prevents the value target from introducing any overestimation over using the standard Q-learning target.

2. **Delayed Policy Updates**

   While the critics are updated at each time step, the actor's policy update is delayed. The actor's policy is updated less frequently, with a fixed interval, reducing the likelihood of updating the policy based on noisy or inaccurate value estimates. This delayed update improves stability and prevents policy oscillations.

3. **Target Policy Smoothing**

   TD3 adds noise to the target policy during the action selection process:

$$y = r +_{\theta} (s', \pi_{\phi'}(s') + \epsilon)$$
$$\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c)$$

   This smoothing technique helps to regularize the learning process and prevents the actor from exploiting potential errors or inconsistencies in the critics' value estimates. By adding noise to the target policy, TD3 encourages exploration and enhances robustness.

The pseudo-code for TD3 algorithm is consistent with that in the original study[12].

Algorithm 2 TD3

```
Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$, and actor network $\pi_\phi$ with random parameters $\theta_1$, $\theta_2$, $\phi$
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for t=1 to $T$ do
    Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward $r$ and new state $s'$
    Store transition tuple $(s, a, r, s')$ in $\mathcal{B}$
    Sample mini-batch of $N$ in transitions $(s, a, r, s')$ from $\mathcal{B}$
    $\widetilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon$, $\epsilon \sim (N(0, \widetilde{\epsilon}), -c, c)$
    $y \leftarrow \tau + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \widetilde{a})$
    Update critics $\theta_i \leftarrow \arg\min_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
    if $t$ mod $d$ then
        Update $\phi$ by the deterministic policy gradient:
        $\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|^{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$
        Update target networks:
        $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
        $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
    end if
end for
```

3.3 Environment settings

The MuJoCo (Multi-Joint Dynamics with Contact) environment is a popular physics engine used in the field of reinforcement learning and robotics. It is integrated as an environment within the OpenAI Gym framework, which provides a standardized interface for developing and evaluating reinforcement learning algorithms.

We test the two algorithms on the following four games of continuous space.

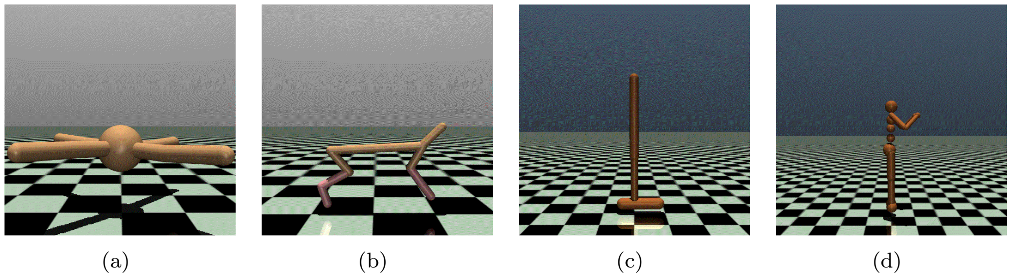


Figure 9: Example MuJoCo environments (a) Ant-v2, (b) Half Cheetah-v2, (c) Hopper-v2, (d) Humanoid-v2.

3.4 Results and Discussion

We train the DDPG agent and the TD3 agent in the four environments separately for 2000000 steps. The recorded scores the two agents obtain during training are as follow.

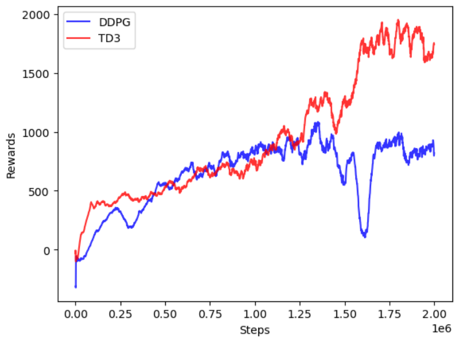
	Ant-v2	**HalfCheetah-v2**	**Hopper-v2**	**Humanoid-v2**
TD3	3823.21	7564.29	3711.63	5999.21
DDPG	3078.36	6822.56	3503.24	5461.99

Table 2: Bests scores of the agents recorded in the training process

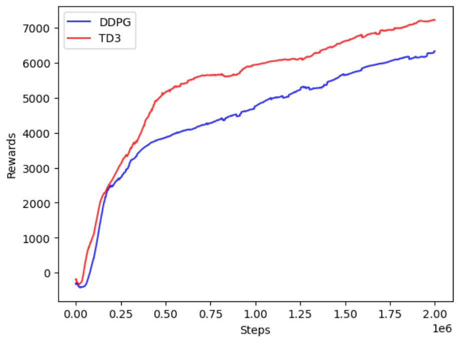
The Figure 10 demonstrates that our agents converge well in the environment Ant-v2, Half Cheetah-v2 and Humanoid-v2, and keeps increasing though fluctuating dramatically in Hopper-v2. TD3 agent perform obviously well than the DDPG agents in all four environments, with the faster increasing rate and higher overall scores.

Combining the information in Table 2 and Figure 10, we conclude that TD3 with multiple improvements on DDPG enhances the agent's performance in all selected environment to a large scale. This result is consistent with the comparison done in the study[12] and implies that the tricks in TD3 pay off.

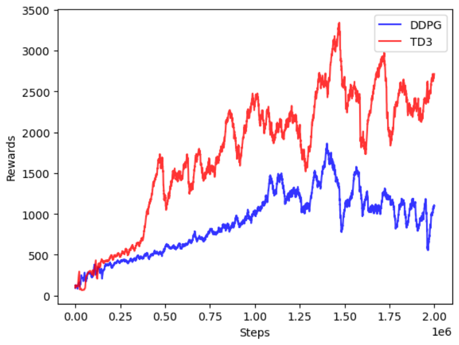
Besides, We observe that In Figure 10, the agent's gain in Half Cheetah-v2 increases more smoothly than it does in the other three environments, which is the expected increasing tendency that suggested in the study[12]. We attribute the dramatic fluctuation to the unsophisticated adjustment of hyperparameters, environment difference and the evaluation approach. For the latter, we hypothesize the results to be more stable if we evaluate the performance by independently run the model on a test environment after each training epoch, which will also address the inconsistency between the highest point on the lines and the values in Table 2.



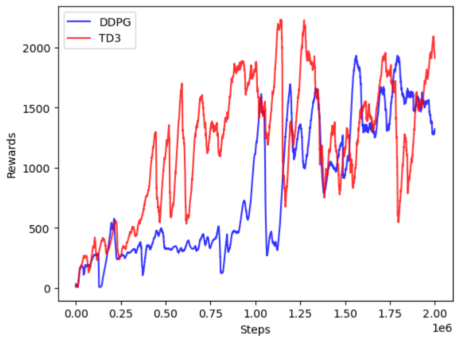
(a) Ant-v2



(b) Half Cheetal-v2



(c) Humanoid-v2



(d) Hopper-v2

Figure 10: Recorded scores of TD3 and DDPG agents while training, with each point on the line demonstrating the average rewards of it's past 100 episodes.

4 Conclusion

In this study, we implement and improve two value-based and two policy-based algorithms and evaluated their performance separately on the Atari-2600 games and the gym environments driven by mujoco. We verify the results reported in the extensively cited papers and improve the performance agents' performance by adjusting the hyperparameters and adding game-specific tricks into algorithm implementation.

To sum up, the DQN family is a series of value-based algorithms that have relatively stable performance, and by adopting prioritized experience replay, multi-step method and apply the reasonable weight initialization to the networks, the agent can reach higher scores and perform more stable.

References

1. Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. nature 2015;518:529–33.
2. Tsitsiklis J and Van Roy B. Analysis of temporal-diffference learning with function approximation. Advances in neural information processing systems 1996;9.
3. Lin LJ. Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine learning 1992;8:293–321.
4. Vanseijen H and Sutton R. A deeper look at planning as learning from replay. In: *International conference on machine learning*. PMLR. 2015:2314–22.
5. Wang Z, Schaul T, Hessel M, Hasselt H, Lanctot M, and Freitas N. Dueling network architectures for deep reinforcement learning. In: *International conference on machine learning*. PMLR. 2016:1995–2003.
6. Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 2013.
7. He K, Zhang X, Ren S, and Sun J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: *Proceedings of the IEEE international conference on computer vision*. 2015:1026–34.
8. Van Hasselt H, Guez A, and Silver D. Deep reinforcement learning with double q-learning. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016.
9. Sutton RS, McAllester D, Singh S, and Mansour Y. Policy gradient methods for reinforcement learning with function approximation. Advances in neural information processing systems 1999;12.
10. Silver D, Lever G, Heess N, Degris T, Wierstra D, and Riedmiller M. Deterministic policy gradient algorithms. In: *International conference on machine learning*. Pmlr. 2014:387–95.
11. Lillicrap TP, Hunt JJ, Pritzel A, et al. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 2015.
12. Fujimoto S, Hoof H, and Meger D. Addressing function approximation error in actor-critic methods. In: *International conference on machine learning*. PMLR. 2018:1587–96.