Reinforcement Learning Final Project

Huang Zhemin

May 2022

Abstract

Deep reinforcement learning has revolutionized the field of artificial intelligence and has made impressive achievement in a lot of areas. In this paper, we conclude the development of model-free reinforcement learning algorithms. Then we evaluate DQN-based algorithms on three of OpenAI Gym tasks, and TD3 algorithm on four of MuJoCo tasks. For DQN algorithm, we use some tricks like multi-step learning, prioritized replay for improvement. For TD3 algorithm, we use softmax operator for value function estimation in continuous control, which outperforms vanilla TD3 algorithm.

Keywords: Model-free reinforcement learning, DQN, Rainbow, TD3, DDPG, Atari, MuJoCo

1 Introduction

Reinforcement learning has achieved success in many practical problems, such as robotics, video games, etc. With recent advances in deep learning, it has attracted more attention and been combined as deep reinforcement learning to solve end-to-end learning in sequential decision tasks.

Interactive Reinforcement Learning algorithms in the literature can be divided into two categories: model-based algorithms and model-free algorithms. For model-free algorithms, we can further divided them into value-based algorithms and policy-based algorithms. Besides, actor-critic approaches have grown in popularity, as an effective way of combining the benefits of policy search with learned value functions, such as Deep-Deterministic Policy gradient (DDPG)[1], Asynchronous Advantage Actor-Critic (A3C)[2], Twin Delayed Deep Deterministic Policy Gradient (TD3)[3].

In this paper, we choose DQN[4] and its variants to test the performance of

value-based methods. For policy-based methods, we choose **Twin Delayed Deep Deterministic Policy Gradient algorithm (TD3)**, which is one of the most popular policy-based algorithms. In order to test the performance of TD3, we use DDPG as a baseline. With the performance in these two algorithms, we then use softmax operator to improve TD3 algorithm, and make a comparison. For both value-based and policy-based methods, we use multiple environment for training and evaluation.
Besides, videos of training results are all exported in */video* directory. You can watch the video to see the result.

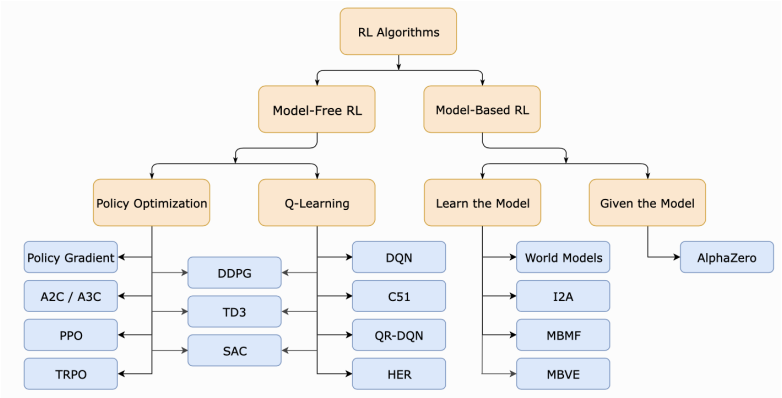


Figure 1: Taxonomy of Reinforcement Learning Algorithms.[5]

2 Value-based methods

Most of the value-based methods are based on **Deep Q Network (DQN)**. Therefore, we will first introduce vanilla DQN, and then propose several improvements.

2.1 Algorithm Description

The DQN (Deep Q-Network) algorithm was developed by DeepMind in 2015.[4] The algorithm was developed by enhancing Q-Learning with deep neural networks and a technique called **experience replay**.

2.1.1 Deep Q-Learning

For most problems, it is impractical to represent Q-function containing each combination of s and a. Therefore, we train a function approximator, a neural network with parameters θ, to estimate Q-values, $Q(s, a; \theta) \approx Q^*(s, a)$. This can be done by minimizing the following loss at each step i:

$$L_i(\theta_i) = E_{s,a,r,s' \sim \rho(\cdot)}[(y_i - Q(s, a; \theta_i))^2 \tag{1}$$

where $y_i = r + \gamma \max_{a'} Q(s', a', ; \theta_{i-1})$. $y_i - Q$ represents TD(temporal difference) error. ρ represents the behaviour distribution.

Note that the parameters from the previous iteration θ_{i-1} are not updated. We use a snapshot of the network parameters from a few iterations ago instead of the alst iteration, which is called **target network**.

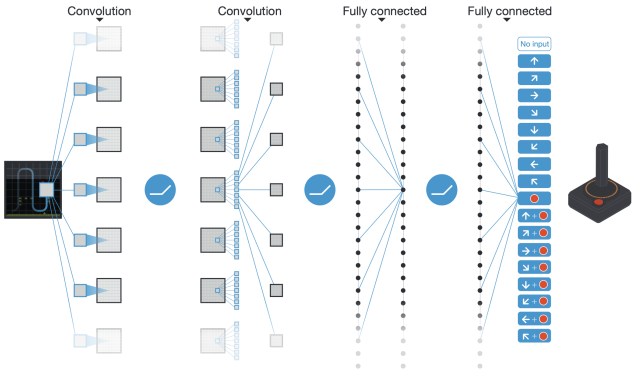


Figure 2: A conceptual visualization of DQN.[4] The input of CNN is state s_t consisting of a few consecutive frames of the video game, from which features are then extracted. These features are then used to compute Q values for every possible action.

2.1.2 Experience Replay

To avoid computing the full expectation in the DQN loss, we can minimize it using stochastic gradient descent. If the loss is computed using just the last transition , this reduces to standard Q-Learning.

DQN introduced a technique called **Experience Replay** to make the network updates more stable. The experience replay algorithm stores the experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ of the agent in a dataset $D_t = e_1, e_2, e_3, ..., e_t$. Then expe-

riences are sampled by randomly drawing

$$e = (s, a, r, s') \sim U(D) \tag{2}$$

and used to perform updates of Q-Learning function.

This has two advantages: **better data efficiency** by reusing each transition in many updates, and **better stability** using uncorrelated transitions in a batch.[6]

2.1.3 Exploration vs. Exploitation with ϵ-greedy

The problem of balance between exploiting the already known and exploring the unknown for reinforcement learning is approached by using the ϵ-greedy algorithm.

The thought of ϵ-greedy is very simple. Exploiting the already known would be taking the action that maximized the Q-learning function, whereas exploring would be taking a random action.

The ϵ-greedy algorithm exploits with probability $1 - \epsilon$ and explores with probability ϵ with $\epsilon \in [0, 1]$.

Based on the details above, we can give pseudo-code of Vanilla DQN. 1

Algorithm 1: Vanilla DQN

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for episode $\leftarrow 1$ **to** M **do**
  Initialize sequence $s_1 = x_1$ and preprocess sequence $\phi_1 = \phi(s_1)$
  for $t \leftarrow 1$ **to** T **do**
    With probability ϵ 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 D
    Sample random mini-batch of transitions $(\phi_t, a_t, r_t, \phi_{t+1})$ from D
    Set

$$y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$$

    Perform gradient descent on $(y_j - Q(\phi_j, a_j; \theta))^2$
  end
end

2.2 Improvement of Algorithm

In this section, we propose several tricks to improvement the effect of vanilla DQN, which have been proved useful. The combination of following tricks is quite similar to Rainbow DQN[7].
However, Noisy Nets are excluded, because they do not have significant improvement on model in my previous experiment.

2.2.1 Prioritized Replay

Prioritized replay was first introduced in ICLR 2016.[8] DQN samples uniformly from the replay buffer. However, we want to sample more frequently those transitions, from which there is much to learn. As a proxy for learning potential, prioritized experience replay samples transitions with probability p_t relative to the last encountered absolute *TD error*:

$$p_t \propto |R_{t+1} + \gamma_{t+1} \max_{a'} q_\theta^-(S_{t+1}, a') - q_\theta(S_t, A_t)|^\omega \tag{3}$$

where ω is a hyper-parameter that determines the shape of the distribution. New transitions are inserted into the replay buffer with maximum priority, providing a bias towards recent transitions.

2.2.2 Multi-step Learning

Multi-step learning is not a new trick, but it is powerful.[9] In Q-learning, we accumulate a single reward and then use the greedy action at the next step to bootstrap. Furthermore, forward-view *multi-step* targets can be used. We can define the truncated n-step return from a given state S_t as

$$R_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^{(k)} R_{t+k+1} \tag{4}$$

Therefore, such variant of DQN can be defined by minimizing the loss below:

$$R_t^{(n)} + \gamma_t^{(n)} \max_{a'} q_\theta^-(S_{t+n}, a') - q_\theta(S_t, A_t))^2 \tag{5}$$

2.2.3 Dueling networks

In previous assignment, we have found that dueling network is a powerful weapon to improve DQN's performance. The thought of Dueling network is quite simple. The algorithm splits the Q-values in two different parts, the value function $V(s)$ and the advantage function $A(s, a)$. This is particularly useful for states where their actions do not affect the environment in a relevant way.

Based on these three methods, we can improve the performance of vanilla DQN to a large scale.

2.3 Environment: Gym Atari

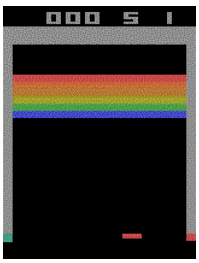
Atari 2600 is a video game console from Atari that was released in 1977. It has been a challenging testbed due to its high-dimensional video input and the discrepancy of tasks between games.

The Atari 2600 environments was originally provided through the Arcade Learning Environment (ALE). The environments have been wrapped by OpenAI Gym to create a more standardized interface. The OpenAI Gym provides 59 Atari 2600 games as environments.

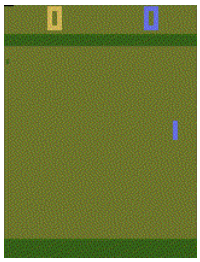
I have trained and tested DQN in environment below, except **VideoPinball** environment. The reason is that the observations are returned with the 128 bytes of RAM in that environment, which is totally different.

- BreakoutNoFrameskip-v4
- PongNoFrameskip-v4
- BoxingNoFrameskip-v4

(a) BoxingNoFrameskip-v4



(b) BreakoutNoFrameskip-v4



(c) PongNoFrameskip-v4

Figure 3: Atari Environment

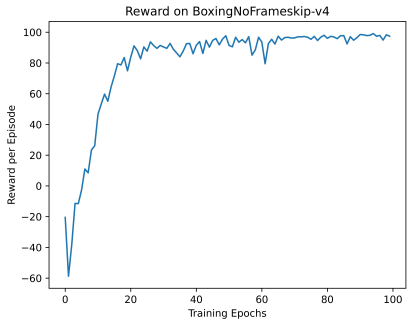
2.4 Experiment Setting

Table 1 shows the hyperparameters of the experiment. Due to the limited computing power, we only experiment on the improved version of DQN, which includes multi-step learning ,prioritized replay and dueling networks. For prioritized replay, we choose prioritization exponent $\omega = 0.5$, linearly increased the importance sampling exponent β from 0.4 to 1. For multi-step learning, n is a very critical parameter. We choose $n = 3$, which the author said performs the best in paper[7].

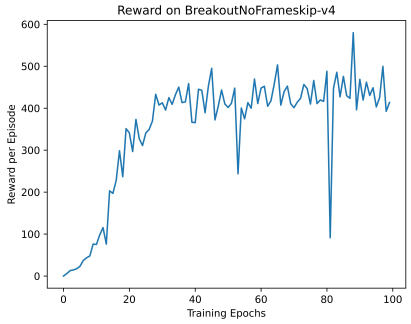
For other hyperparameters, our batch size equals to 32, and the learning rate equals to 10^{-5}, which is smaller than vanilla DQN.[4]

2.5 Result

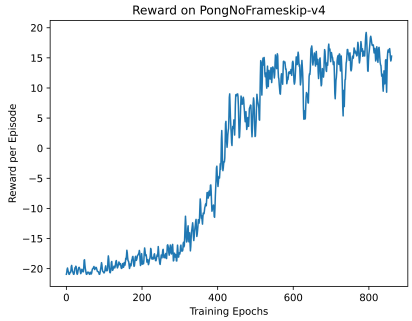
The following figure shows the result for our improved DQN version, with prioritized replay, multi-step learning and dueling networks. All of them get satisfactory results, comparing with the vanilla DQN. For result of PongNoFrameskip-v4, we use Gaussian Filter ($\sigma = 1$) to smooth the curve. One epoch is of 100000 environment steps.



(a) BoxingNoFrameskip-v4



(b) BreakoutNoFrameskip-v4



(c) PongNoFrameskip-v4

Figure 4: Evaluation Rewards for Atari tasks.

Besides, we have recorded the best return for value-based algorithms. Table 1 shows the result.

Table 1: Best Return for Improved-DQN Algorithm

Environment	Score
BoxingNoFrameskip-v4	99.10
BreakoutNoFrameskip-v4	579.90
PongNoFrameskip-v4	21

3 Policy-based methods

Before we have thorough discussion, we should take a brief review and make a comparison between several mainstream policy-based methods.

Deep Deterministic Policy Gradient(DDPG) is based on the deterministic policy gradient(DPG) algorithm, which concurrently learns a Q-function and a policy.[1] It uses off-policy data and the Bellman equation to learn the Q-function, and uses the Q-function to learn the policy. Like DQN, DDPG use experience replay buffer to approximate $Q^*(s, a)$. Besides, it use **soft target update** to ensure convergence.

Soft Actor Critic (SAC) is an successor to DDPG.[10] It optimizes a stochastic policy in an off-policy way, forming a bridge between stochastic policy optimization and DDPG-style approaches. A key factor of SAC is **entropy regularization**. The policy is trained to maximize a trade-off between expected return and entropy, a measure of randomness in the policy. It can also prevent the policy from prematurely converging to a bad local optimum.

In original paper of TD3, the authors offer better result to SAC on most task. However, this comparison is against a prior version of SAC. The latest version of SAC includes Clipped Double Q-Learning, which also produces competitive result.[10]

Proximal Policy Optimization (PPO) is an on-policy algorithm[11], unlike DDPG and SAC. It simplifies the computationally expensive constraint calculations and second-order approximations in the Trust Region Policy Optimization (TRPO) algorithm.[12] By simply clipping the importance sampling term in the surrogate Advantage function to be close to unity, PPO prevents the policy update from diverging far from the previous policy.

Alternatively, another variant of PPO applies dual gradient descent, adjusting the Lagrangian multiplier for the KL-divergence in reaction to a breach of the constraint.

3.1 Algorithm Description

In this section, we will focus on the main policy-based algorithm used in this paper: **Twin-Delayed Deep Deterministic Policy Gradient (TD3)**, which has similar performance to SAC, and higher sampling efficiency than PPO.

Firstly, we would introduce a crucial drawback of DDPG. In some environment, DDPG can achieve a relatively high score. However, it often needs a find tuning of hyperparameters. A fatal flaw of DDPG is that **the learned Q-function may dramatically overestimate Q-values**, which then leads to suboptimal policies.

Twin Delayed Deep Deterministic Policy Gradient (TD3) is an algorithm that

addresses this problem by introducing three critical tricks[3]:

Trick One: Clipped Double Q-Learning. It is the key component in TD3. TD3 learns two Q-functions instead of one, and uses the smaller of the two Q-values to form the targets in the Bellman error loss functions:

$$y_1 = r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \pi_{\phi_1}(s')) \tag{6}$$

The first benefit is that with Clipped Double Q-Learning, the value target cannot introduce additional overestimation over using the standard Q-Learning target; A secondary benefit is that the minimization in Equation 6 will lead to a preference for states with low-variance value estimates, leading to safer policy updates with stable learning targets.

Trick Two: Delayed Policy Updates. TD3 updates the policy (and target networks) less frequently than the Q-function. The less frequent policy updates that do occur will use a value estimate with lower variance, and should result in higher quality policy updates. The paper recommends one policy update for every two Q-function updates.[3]

Trick Three: Target Policy Smoothing. TD3 adds noise to the target action, to make it harder for the policy to exploit Q-function errors by smoothing out Q along changes in action. This makes the modified target update:

$$y = r + \gamma Q_{\theta'}(s', \pi_{\phi'}(s') + \epsilon) \tag{7}$$

$$\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -\text{c}, \text{c}) \tag{8}$$

where the added noise is clipped to keep the target close to the original action. Based on description above, we would give the pseudo-code of TD3 algorithm.2

Algorithm 2: TD3 Algorithm

Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$, and actor network π_θ with random parameters θ_1, θ_2, ϕ
Initialize target networks ${\theta_1}' \leftarrow \theta_1, {\theta_2}' \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t \leftarrow 1$ **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 transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}, -c, c))$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \tilde{a})$
  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 ϕ by deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    ${\theta_i}' \leftarrow \tau\theta_i + (1 - \tau){\theta_i}'$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end
end

3.2 Improvement of Algorithm

TD3 significantly improves the performance of DDPG by reducing the overestimation. Specifically, TD3 estimates the value function by taking the minimum of value estimates from the two critics according to $y_1, y_2 = r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \tilde{a})$ Nevertheless, it can also incur **a large underestimation bias**, which impacts performance.[13]

In this section, we use **softmax operator** in updating value functions in actor-critic methods for continuous control. It can smooth the optimization landscape and thus helps learning empirically.[14]

In continuous action space, the softmax operator is defined by $softmax_\beta(Q(s, \cdot)) = \int_{\alpha \in A} \frac{exp(\beta Q(s,a))}{\int_{\alpha' \in A} exp(\beta Q(s,a'))da'} Q(s, a)da$, where β is the parameter of the softmax operator.

We use the softmax operator based on double estimators to relieve the problem. We estimate the target value for critic Q_i by $y_i = r + \gamma\tau(s')$, where

$$\tau(s') = softmax_\beta(Q_i(s', \cdot)) \tag{9}$$

$$\hat{Q}_i(s', a') = \min(Q_i(s', a'; \theta_i^-), Q_{-i}(s', a'; \theta_{-i}^-)) \tag{10}$$

For proof of why the softmax operator can reduce overestimation bias, please refer to [14].

In order to clarify the algorithm details, we give the pseudo-code of improved version of algorithm. 3

Algorithm 3: TD3 with softmax operator (SD3)

Initialize critic networks Q_1, Q_2, and actor network π_1, π_2 with random parameters $\theta_1, \theta_2, \phi_1, \phi_2$
Initialize target networks $\theta_1{}' \leftarrow \theta_1, \theta_2{}' \leftarrow \theta_2, \phi_1{}' \leftarrow \phi_1, \phi_2{}' \leftarrow \phi_2$
Initialize replay buffer $\mathcal{B}$
for $t \leftarrow 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma)$
  Execute action a, observe reward r, new state s' and done d
  Store transition tuple (s, a, r, s', d) in $\mathcal{B}$
  for $i = 1, 2$ **do**
    Sample mini-batch of N transitions (s, a, r, s', d) from $\mathcal{B}$
    Sample K noises $\epsilon \sim \mathcal{N}(0, \sigma)$
    $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}, -c, c)$
    $\tilde{Q}(s', \tilde{a}') \leftarrow \min_{j=1,2}(Q_j(s', \tilde{a}'; \theta_j{}^-))$

    $\text{softmax}_\beta(\tilde{\text{Q}}(\text{s}', \cdot)) \leftarrow$
    $\mathcal{E}_{\hat{\text{a}}' \sim \text{p}}[\frac{\exp(\beta \tilde{\text{Q}}(\text{s}', \text{a}'))\tilde{\text{Q}}(\text{s}', \tilde{\text{a}}')}{\text{p}(\tilde{\text{a}}')}] / \mathcal{E}_{\hat{\text{a}}' \sim \text{p}}[\frac{\exp(\beta \tilde{\text{Q}}(\text{s}', \text{a}'))}{\text{p}(\tilde{\text{a}}')}]$
    $y_i \leftarrow r + \gamma(1 - d)\text{softmax}_\beta(\tilde{(\text{Q})}(\text{s}', \cdot))$

    Update critics $\theta_i \leftarrow \arg\min_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
    Update ϕ by deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta_i{}' \leftarrow \tau\theta_i + (1 - \tau)\theta_i'$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end
end

3.3 Environment: MuJoCo

MuJoCo stands for **Multi-Joint dynamics with Contact**. It is a general purpose physics engine that aims to facilitate research and development, written in C/C++. The user defines models in the native MJCF scene description language, which is an **XML file format**. The library includes interactive visualization with a native GUI, rendered in OpenGL.

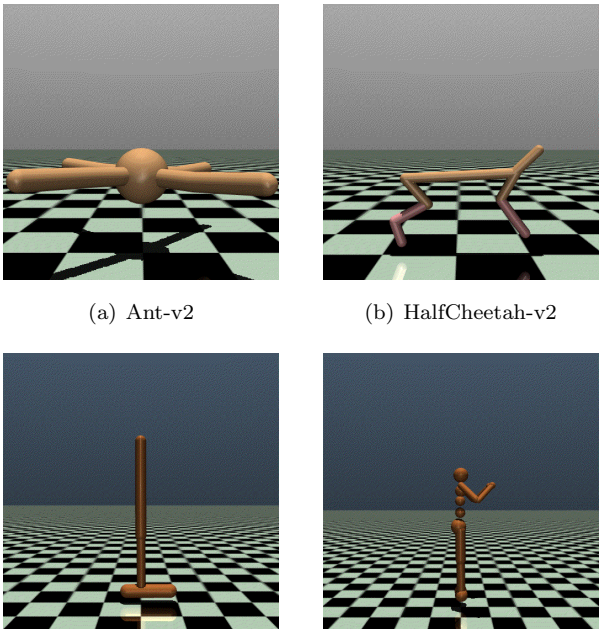
MuJoCo can be used to implement model-based computations such as control synthesis, state estimation, system identification, mechanism design, data

analysis through inverse dynamics, and parallel sampling for machine learning applications.

It is recommended that we should use MuJoCo under **Linux** environment, because it is hard to configure the environment on Windows. Besides, since Mujoco was acquired and made freely available by DeepMind in October 2021, and open sourced in May 2022, we do not need a license after Version 210.

In my experiment, I have trained and tested TD3 with hyperparameters given above in all of the given MuJoCo environments.

- Ant-v2
- HalfCheetah-v2
- Hopper-v2
- Humanoid-v2



(a) Ant-v2 (b) HalfCheetah-v2

(c) Hopper-v2 (d) Humanoid-v2

Figure 5: Example MoJoCo Environment

3.4 Experiment Settings

Table 3 shows the hyperparameter settings we used in policy-based algorithms. Most of them are similar to the original paper. For TD3, we set the learning

rate of critic and actor networks to 3×10^{-4}, instead of 10^{-3}. For improved version of TD3, most hyperparameters are the same as TD3. The parameter β is 10^{-3} for Ant-v2.

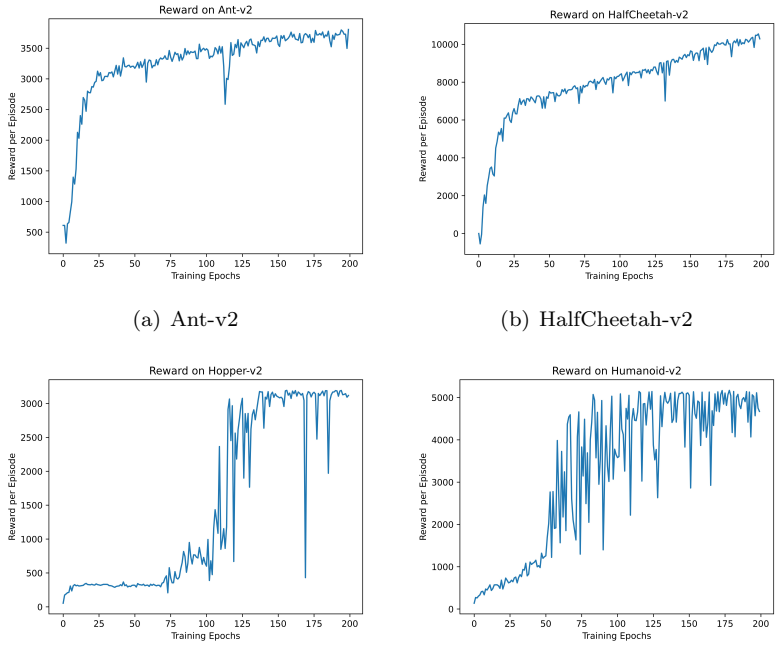
Table 2: Hyperparameters of Policy-based Algorithms

Hyperparameter	TD3	DDPG
Batch Size	100	64
Target Update Rate (τ)	5×10^{-3}	10^{-3}
Critic Learning Rate	3×10^{-4}	10^{-3}
Actor Learning Rate	3×10^{-4}	10^{-4}
Exploration Policy	$\mathcal{N}(0, 0, 1)$	OU Noise
Discount Factor	0.99	
Reward Scaling	1.0	
Optimizer	Adam	

3.5 Result

3.5.1 TD3 Algorithm

We have done experiment on the four MuJoCo environment. The result is in Figure 6. It is shown that TD3 converges well in all of the given environment. Due to the difference of environment, the convergence speed varies. For example, TD3 converges quickly and has low variance in environment of Ant and HalfCheetah, while variance of Humanoid is relatively high in our result.



(a) Ant-v2 (b) HalfCheetah-v2

(c) Hopper-v2 (d) Humanoid-v2

Figure 6: Evaluation Rewards for MuJoCo continuous control tasks. (TD3)

Furthermore, we compare our result with OpenAI Spinning Up Benchmark.[15] Table 3 gives the comparison. Our result has a better performance than the benchmark.

Table 3: Best Return for TD3 Algorithm

Environment	Score	Spinning Up Benchmark Score(TD3, PyTorch)
Ant-v2	5656.32	≈3800
HalfCheetah-v2	10547.39	≈9800
Hopper-v2	3194.02	≈2900
Humanoid-v2	5170.23	None

3.5.2 Comparison with other Algorithms

Due to the limited computing power, we choose **Ant-v2** environment as a representative, to compare TD3 with its improved version and DDPG. The result

is shown in Figure 7.

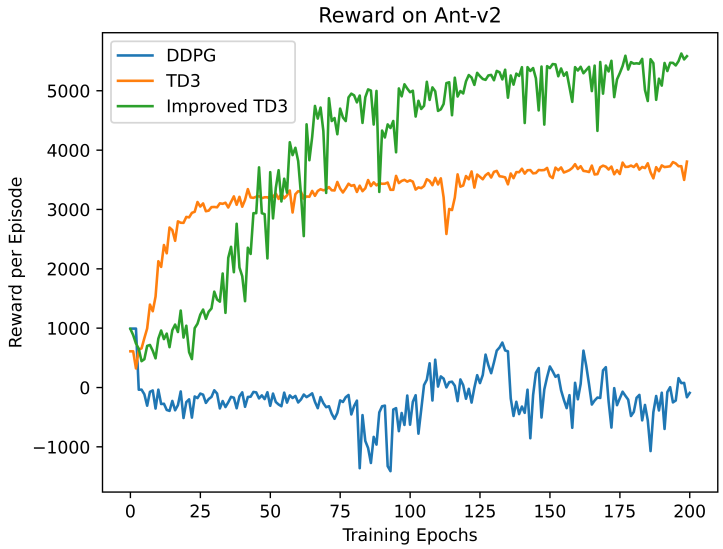


Figure 7: Performance comparison of DDPG, TD3, and TD3 with softmax operator(SD3).

As demonstrated, the improved version of TD3 outperforms the vanilla TD3, where it achieves a higher performance and is more stable, due to the smoothing effect of softmax operator and reduce of overestimation bias. Besides, TD3 significantly outperforms DDPG, which proves the result in the paper.[3]

4 Conclusion

In this paper, we have used several advanced model-free reinforcement learning algorithms for testing.

In conclusion, DQN and its variants are very powerful to control games like Atari. The improvements like prioritized replay, multi-step learning are very useful and can improve scores a lot.

For continuous problems, TD3 performs very well, while DDPG has a relatively weak performance. Using the improvement like softmax operator, we can achieve better results than vanilla TD3.

References

[1] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[2] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[3] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1582–1591, 2018.

[4] 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):529–533, 2015.

[5] OpenAI. Kind of reinforcement learning algorithm. https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html.

[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] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[8] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[9] Richard S Sutton, Andrew G Barto, et al. Introduction to reinforcement learning. 1998.

[10] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with

a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

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

[12] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[13] Kamil Ciosek, Quan Vuong, Robert Loftin, and Katja Hofmann. Better exploration with optimistic actor critic. *Advances in Neural Information Processing Systems*, 32, 2019.

[14] Ling Pan, Qingpeng Cai, and Longbo Huang. Softmax deep double deterministic policy gradients. *Advances in Neural Information Processing Systems*, 33:11767–11777, 2020.

[15] OpenAI. Spinning up benchmark. https://spinningup.openai.com/en/latest/spinningup/bench.html.

[16] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[17] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[18] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

[19] Stephen Dankwa and Wenfeng Zheng. Twin-delayed ddpg: A deep reinforcement learning technique to model a continuous movement of an intelligent robot agent. In *Proceedings of the 3rd International Conference on Vision, Image and Signal Processing*, pages 1–5, 2019.

[20] Jiayi Weng, Huayu Chen, Dong Yan, Kaichao You, Alexis Duburcq, Minghao Zhang, Hang Su, and Jun Zhu. Tianshou: A highly modularized deep reinforcement learning library. *arXiv preprint arXiv:2107.14171*, 2021.

[21] Guillaume Matheron, Nicolas Perrin, and Olivier Sigaud. The problem with ddpg: understanding failures in deterministic environments with sparse rewards. *arXiv preprint arXiv:1911.11679*, 2019.

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

[23] Thomas Nakken Larsen, Halvor Ødegård Teigen, Torkel Laache, Damiano Varagnolo, and Adil Rasheed. Comparing deep reinforcement learning algorithms' ability to safely navigate challenging waters. *Frontiers in Robotics and AI*, 8, 2021.

[24] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.