Experiments of Model-Free Algorithms on Value-Based and Policy-Based RL

Xiao Huang
Department of Computer Science
Shanghai Jiao Tong University
hwangxiao@sjtu.edu

Abstract

In area of model-free control RL, value-based and policy-based methods have commonly classified the problems based on the continuity of action spaces. In this paper, we implement DQN and its improved version Dueling-Double-Deep Q-Network (Dueling DDQN) algorithm as the representation of our advanced value-based methods as well as making detailed comparison and analysis on the performance on DQN series methods. As for continuous action space, we choose to implement we implement two policy based algorithms: PPO (on-policy) and SAC (off-policy) and test them on four of Gym MuJoCo environments.

Keywords: Model-free control, Dueling-DDQN, PPO, SAC

1 Introduction

In area of reinforcement learning, value-based algorithm Deep Q-Network showed excellent performance in model-free control problems with discrete actions. Series of algorithms based on DQN[1] have been proposed, including Double DQN[2], which uses target networks to re-estimate the greedy action value to avoid over-estimation, and Dueling DQN[3], which learns state value and action advantage function separately to promote the representative power. Dueling DDQN[4] algorithm has been proposed to improve the model-free control performance., which combines DDQN with the dueling network architecture. In this paper, we will implement our Dueling DDQN algorithm, make reasonable modifications to better fit the local host environment, and evaluate it on various Gym Atari environments with the performance comparison to original DQN.

Compared to value-based algorithm, policy-based algorithm is proposed to solve continuous control problems. The key idea is to optimize over an actor directly using policy gradient. Actor-Critic algorithms such as proximal policy optimization (PPO) [5] introduce an additional module named critic to estimate state or action value, which can dramatically reduce variance (though introduce bias) and stabilize the training procedure. The biggest drawback for policy gradient methods is sample inefficiency: since policy gradients are estimated from rollouts. Although Actor-Critic methods use value approximation instead of rollouts, its on-policy style remains sample inefficient. Taking the idea from Q-learning, prior works such as deep deterministic policy gradient (DDPG) [6] and soft Actor-Critic (SAC) [7] strive to introduce off-policy mode to policy-based RL. In this paper, we implement and evaluate two traditional policy-based RL algorithms: PPO (on-policy) and SAC (off-policy) on various Gym MuJoCo environments.

2 Value-based methods

Most of the value-based methods are based on Deep Q Network (DQN). DQN algorithm was developed by DeepMind in 2015.The algorithm was developed by enhancing Q-Learning with deep

neural networks and a technique called experience replay. Therefore, we will first introduce DQN, and then propose several improvements.

2.1 Deep Q-Learning

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

$$L_i\left(\theta_i\right) = \mathbb{E}_{(s,a,r,s')\sim \mathrm{U}(D)}\left[\left(r + \gamma \max_{a'} Q\left(s', a'; \theta_i^-\right) - Q\left(s, a; \theta_i\right)\right)^2\right] \tag{1}$$

in which γ is the discount factor determining the agent's horizon. θ_i are the parameters (that is, weights) of the Q-network at iteration i and θ_i^- are the network parameters used to compute the target at iteration i. We parameterize an approximate value function $Q\left(s, a; \theta_i\right)$ using the deep convolutional neural network shown in Fig. 1.

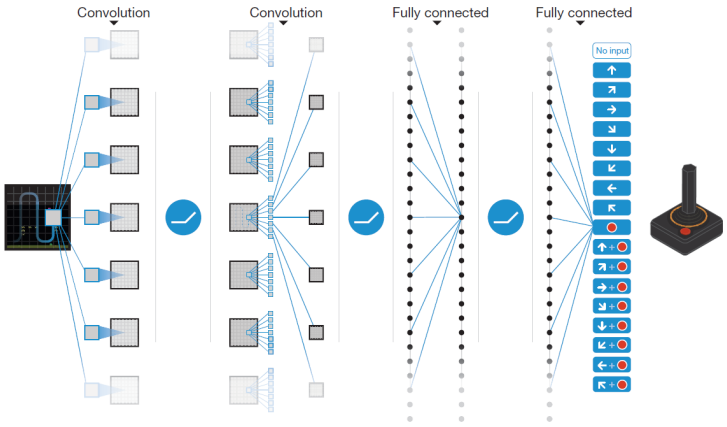

Figure 1: Schematic illustration of the CNN

2.2 Experience Replay

DQN introduced a technique called Experience Replay to make the network updates more stable. To perform experience replay we store the agent's experiences $e_t(s_t, a_t, r_t, s_{t+1})$ at each time-step t in a data set $D_t\{e_1, \ldots, e_t\}$. During learning, we apply Q-learning updates, on samples (or minibatches) of experience $(s, a, r, s') \sim U(D)$, drawn uniformly at random from the pool of stored samples. This has two advantages: better data effciency by reusing each transition in many updates, and better stability using uncorrelated transitions in a batch.

2.3 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 DQN.

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For t = 1,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    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 minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

2.4 Dueling networks

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 $A(s; a)$. From the figure below, compared to the convolutional layers with a single sequence of fully connected layers, we instead use two streams of fully connected layers. The streams are constructed such that they have the capability of providing separate estimates of the value and advantage functions. Finally, the two streams are combined to produce a single output Q function:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha) \tag{2}$$

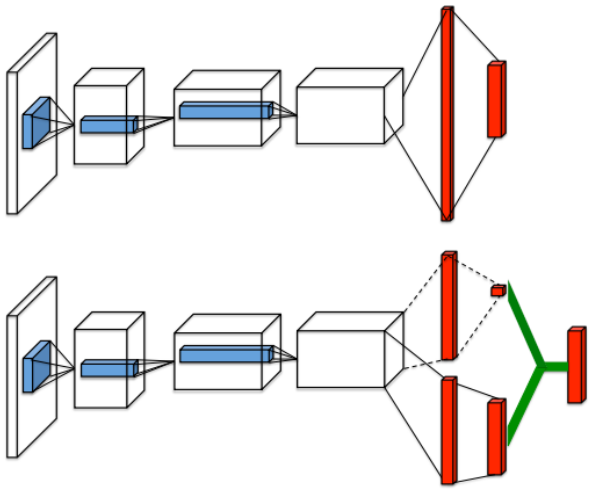

Figure 2: Difference of Network architecture between DQN and Dueling DQN

2.5 Double networks

The target Q function $r + \gamma \max_{a'} Q(s', a'; \theta_i^-)$ tends to select the action that is over-estimated. To mitigate this issue, DDQN proposes to select the target action by the policy net, while evaluate its value by the target net:

$$r + Q'(s', \arg\max_{a'} Q\left(s', a'; \theta\right); \theta_i^-) \tag{3}$$

We decouple the process of action choosing and value estimation, which can significantly eliminate the over-estimation issue. For example, if Q over-estimate a', so it is selected. Then Q' will give it proper value. Besides, the action overestimated by Q' will not be selected by Q.

2.6 Environment: Gym Atari

Atari 2600 has been a challenging testbed due to its high-dimensional video input and the discrepancy of tasks between games. Gym provides a Wrapper class[1] for better adaptation to Atari 2600 games, which is inherited from the Env base class. In this way, we can customize the configurations and interaction method when using different Atari environments by passing environment-specific parameters to methods in the Wrapper class.

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.

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

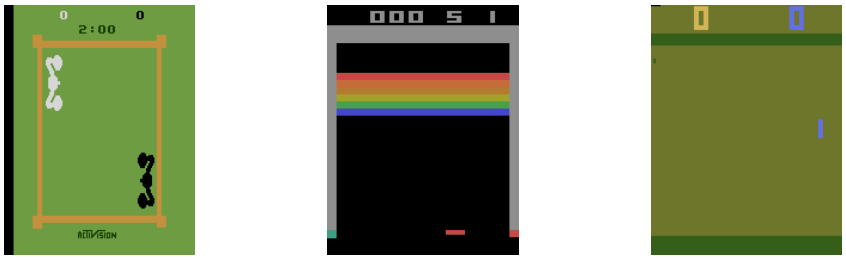

Figure 3: Atari Environment

The reward for each action is divided into {-1,0,+1} to prevent the influence of different reward settings across different environments. Besides, the original observed image is converted from 210×160 to 84×84, and the RGB image is converted to a gray scale image, which would be stored into the replay buffer after transformation. Using only one frame of image to represent current state may not provide enough information for the agent. Frame staking technology combines the previous k frames to form the state space. Typically, this can provide context for the observation and lead the agent to take correct actions. Specifically, we take $k = 4$ in our experiment.

2.7 Experiment Setting

Here are settings and some hyperparameters of the experiment.

- We train the Q network for 1e7 steps, optimize the model for one batch after every 4 environment steps, and update the target network for every 1e5 steps.
- The size of the replay buffer is 200,000, and the optimization procedure starts when the replay buffer owns 500 pieces of experiences.

[1] https://github.com/openai/baselines/blob/edb52c22a5e14324304a491edc0f91b6cc07453b/baselines/common/atari_wrappers.py

- We use the ϵ-greedy algorithm for exploration. Typically, ϵ is initialized as 1.0, and linearly drop to 0.1 after 1e6 steps.
- We evaluate the training agent for every 1e6 environment steps. Specifically, we greedily roll out 10 episodes to calculate the mean and std of scores.
- The learning rate is set as 1e-4, and the discounted factor γ is set as 0.99. Our batch size equals to 32 and Adam optimizer is adopted for optimization..

2.8 Results and Discussion

The following figure shows the result for our DQN and DuelingDDQN. For results, we use Gaussian Filter (σ = 1) to smooth the curve. One epoch is of 100000 environment steps.

Table 1: Best Score for Value-based Methods

Environment	Best Score
BoxingNoFrameskip-v4	94.35
BreakoutNoFrameskip-v4	368.3
PongNoFrameskip-v4	20.45

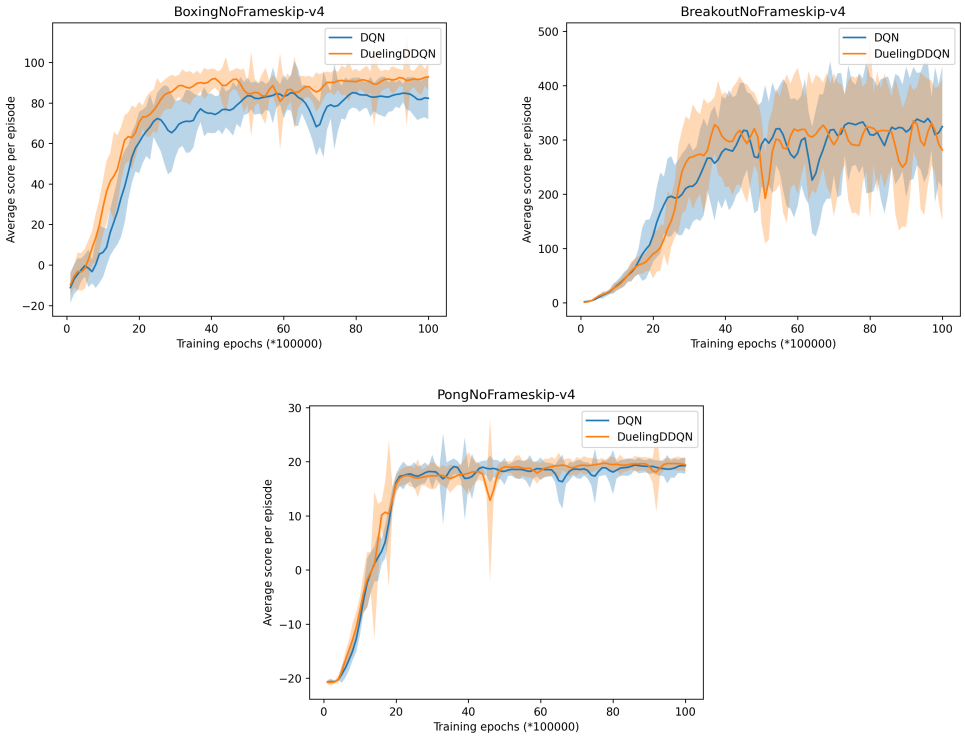

Figure 4: Different Experiment Results for Value-based Methods

Generally, two algorithms both achieve satisfying scores on the three testing environments. As we can see, the final performance of two algorithms are almost on a par with each other. In comparison, Dueling DDQN shows the better sample efficiency, which learns and converges faster than original DQN algorithms, especially on BoxingNoFrameskip-v4. Therefore, we can conclude that double and dueling architecture can promote the sample efficiency of DQN.

3 Policy-based methods

Deep Deterministic Policy Gradient(DDPG)[8] is based on the deterministic policy gradient(DPG) algorithm, which concurrently learns a Q-function and a policy. 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)[9] is an successor to DDPG. 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.

Proximal Policy Optimization (PPO)[10] is an on-policy algorithm, unlike DDPG and SAC. It simplifies the computationally expensive constraint calculations and second-order approximations in the Trust Region Policy Optimization (TRPO) algorithm. 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 PPO algorithm

As the original policy gradient suffers from high variance due to the rollout estimates, an additional critic is introduced in Actor-Critic (AC) for value estimation. Specifically, the critic uses the approximation $Q^\pi(s_t^n, a_t^n) = \mathbb{E}[r_t^n + V^\pi(s_{t+1}^n)] \approx r_t^n + V^\pi(s_{t+1}^n)$ and only estimates the state value. In this sense, the policy gradient can be modified as:

$$\nabla \bar{R}_\theta = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} (r_t^n + V^\pi(s_{t+1}^n) - V^\pi(s_t^n)) \nabla \log \pi_\theta(a_t^n | s_t^n) \tag{4}$$

which can reduce the variance drastically (though introduce some bias). Typically, the actor is optimized using policy gradient, while the critic is optimized by Monte-Carlo or the TD error.

PPO uses the importance sampling to modify the policy gradient. We can use experience sampling from policy $\pi_{\theta'}$ to optimize π_θ , which can promote the sample efficiency significantly. We adopt generalized advantage estimation (GAE) to calculate $A(s_t, a_t)$ for its promising performance. Its key idea is similar to TD(λ), but we omit it here since it's not our main concern in this paper. However, we need to constraint the difference between $\pi_{\theta'}$ and π_θ for practical use. Specifically, in PPO1, we introduce an additional KL-loss between $\pi_{\theta'}$ and π_θ . Here we adopt PPO2, which is easier to implement while achieving competitive performance:

$$J_\pi^{\theta^k}(\phi) = \sum_{(s_t, a_t)} \min \left(\frac{\pi(a_t|s_t)}{\pi(a_t|s_t)} A(s_t, a_t), \text{clip} \left(\frac{\pi(a_t|s_t)}{\pi(a_t|s_t)}, 1-\epsilon, 1+\epsilon \right) A(s_t, a_t) \right) \tag{5}$$

As we can see, it clips the probability ratio to prevent unreasonable gradient steps. In practical use, PPO will use a policy net to interact with the environment and fill the replay buffer. When the replay buffer is full, it will start optimizing the agent for a number of epochs. Then the replay buffer will be cleared and we will use the new policy net to sample experience for the next iteration. Thus, the experienced transitions are only used in one iteration, and the size of the replay buffer is rather small to ensure that π_θ will not go too far away from π'_θ.

3.2 Maximum Entropy Reinforcement Learning

The objective for standard reinforcement learning is to find the policy that can maximize the cumulative reward. While the objective for MERL is:

$$\pi^*_{MaxEnt} = \arg\max_\pi \sum_t \mathbb{E}_{(s_t, a_t) \sim \pi} [r(s_t, a_t) + \alpha H(\pi(\cdot|s_t))] \tag{6}$$

where α is a hyperparameter named temperature to control the significance of the policy entropy. Different from in PPO, where we introduce an entropy loss to encourage the actor for more exploration, in MERL we directly put the entropy into the objective, which will change the Bellman equation completely. Experiments have proved that MERL can perform well in many continuous control scenarios. It also performs more stable and owns excellent immunity to interference compared to other off-policy algorithms such as DDPG.

3.3 Energy Based Policy

In SAC, energy-based models (EBMs) are adopted to represent policy:

$$\pi(a_t|s_t) \propto \exp(-\mathcal{E}(s_t, a_t)) \tag{7}$$

different from the previously used Gaussian distribution, EBM owns the promising power to approximate any probability model. Therefore, the policy model in SAC is more general and can learn more useful information during the training procedure. Specifically, the energy function in SAC is connected with the Q function:

$$\pi(a_t|s_t) \propto \exp(-\frac{1}{\alpha}Q_{soft}(s_t, a_t)) \tag{8}$$

3.4 Soft Policy Iteration

Although we can use policy gradient to solve the equation, here we have more clever methods to simplify the process. Typically, the objective can be re-written as:

$$J(\pi) = \sum_{t=1}^{T} \gamma^{t-1} \mathbb{E}_{(s_t,a_t)\sim\pi}[r(s_t, a_t) + \alpha H(\pi(\cdot|s_t))] \tag{9}$$

In soft Q-learning (SQL), the soft V function is defined as:

$$V_{soft}(s) = \alpha \log \int \exp(-\frac{1}{\alpha}Q_{soft}(s_t, a_t)) \tag{10}$$

To find the optimal policy π that maximizes this equation, we can calculate the derivative (set to zero) of it w.r.t. $\pi(a_t|s_t)$, which yields that:

$$\pi(a_t|s_t) = \exp(\frac{1}{\alpha}(Q_{soft}(s_t, a_t) - V_{soft}(s_t))) \tag{11}$$

where $Q_{soft}(s_t) = r(s_t, a_t) + \gamma \mathbb{E}[V_{soft}(s_{t+1})]$. The results implies that the optimal policy can be derived if we have known the soft value functions. Inspired by this, we can use a similar policy iteration training procedure as in Q-learning to optimize the agent, which enables off-policy training mode in SAC. SAC proposes a soft bellman equation to conduct value iteration:

$$\begin{aligned} Q^{\pi}_{soft}(s, a) &= \mathbb{E}_{s'\sim p(s'|s,a)}[r(s, a) + \gamma V^{\pi}_{soft}(s')] \\ &= \mathbb{E}_{s'\sim p(s'|s,a), a'\sim\pi}[r(s, a) + \gamma \left(Q^{\pi}_{soft}(s', a') + \alpha H(\pi(\cdot|s'))\right)] \end{aligned} \tag{12}$$

Though we can calculate the policy probability directly, however, we cannot directly sample actions from the EBP. Therefore, we use an Gaussian distribution instead to interact with the environment. Afterwards, we will optimize the Gaussian distribution towards the EBP. Specifically, we try to minimize the KL divergence:

$$\pi_{new} = \arg\min_{\pi\in\mathcal{N}} D_{KL}\left(\pi(\cdot|s_t)||\frac{\exp(\frac{1}{\alpha}Q^{\pi_{old}}_{soft}(s_t, \cdot))}{Z^{\pi_{old}}_{soft}(s_t)}\right) \tag{13}$$

where $\mathcal{N}$ denotes the set of parameterized Gaussian distribution, $Z_{soft}^{\pi_{old}}(\cdot)$ is the function to normalize the probability distribution. The action a_t in the expectation is sampled from the current policy but not from the replay buffer. This is the key difference between SAC and those algorithms based on policy gradient, which enables off-policy training and promotes the sample efficiency greatly. Since the action at is sampled from $\pi(\cdot|s_t)$, we can introduce the reparameterization technique to make the backpropagation process differentiable. Specifically, we have:

$$a_t = f\left(\epsilon_t, s_t; \phi\right) = f^{\mu}\left(s_t; \phi\right) + \epsilon_t \odot f^{\sigma}\left(s_t; \phi\right) \tag{14}$$

where ϵ_t is sampled from the standard Gaussian distribution. Therefore, the final actor loss is transformed as:

$$J_{\pi}(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}, \epsilon \sim \mathcal{N}_{\text{std}}}\left[\alpha \log \pi\left(f\left(\epsilon_t, s_t; \phi\right) \mid s_t; \phi\right) - Q\left(s_t, f\left(\epsilon_t, s_t; \phi\right)\right)\right] \tag{15}$$

3.5 Automating Entropy Adjustment

The temperature α controls the significance of entropy in the objective. However, the optimal may differ for different RL tasks, or in different stages of a same problem. As this is an important hyperparameter that will determine the performance of SAC. An algorithm has been proposed to tune the temperature automatically. Specifically, the authors formulate it as a constrained optimization problem (maximize the expected return while maintain the entropy to be larger than a threshold) and induce the loss function for the temperature:

$$J(\alpha) = \mathbb{E}_{a_t \sim \pi\left(a_t | s_t; \phi\right)}\left[-\alpha \log \pi\left(a_t \mid s_t; \phi\right) - \alpha H_0\right] \tag{16}$$

3.6 Prioritized Experience Replay

W can use it to sample experienced transitions from the replay buffer w.r.t. their priority. Typically, the priority is defined as the TD error of the transition pair. In our experiment, the priority of the i_{th} sample is calculated as:

$$p_i = \frac{1}{2} \sum_{l=1}^{2} \left| Q\left(s_i, a_i; \theta_l\right) - \left(r\left(s_i, a_i\right) + \gamma\left(Q\left(s_i', a_i'\right) - \alpha \log\left(\pi\left(a_i' \mid s_i'\right)\right)\right)\right)\right| \tag{17}$$

Afterwards, the probability of sampling the i-th data is $P(i) = \frac{p_i^{\beta_1}}{\sum_j p_j^{\beta_1}}$, where β_1 is a hyperparameter that controls how much the priority value affects the sampling probability. As we can see, a transition pair with larger TD error is more likely to be chosen. PER introduces bias because it changes the sample distribution in an uncontrolled fashion. We can correct this bias by using importance sampling weights $w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^{\beta_2}$ that fully compensates for the non-uniform probabilities $P(i)$ if $\beta_2 = 1$

3.7 Environment: MuJoCo

MuJoCo [2] is a general purpose physics engine that aims to facilitate research and development, written in C/C++. 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.

[2] https://mujoco.org/download/mujoco210-linux-x86_64.tar.gz

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

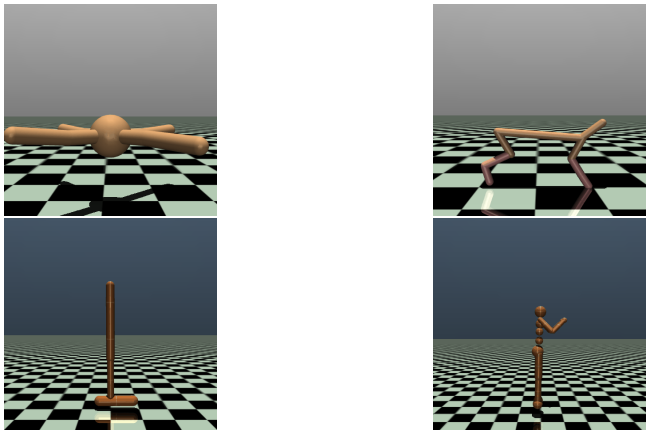

Figure 5: Mujoco Environment

3.8 Experiment Setting

Here are settings and some hyperparameters of the experiment. The number of total training steps depends on the specific environments. Typically, we run 2e6 steps for Humanoid, Ant and Halfcheetah, and 1e6 steps for Hopper.

For PPO:

- The batch size is 64. The Adam optimizer is adopted for optimization.
- We optimize the model for 10 epochs after every 2000 steps.
- The size of the replay buffer is 2000, exactly the same as the period of model optimization.
- We evaluate the training agent for every 2000 steps. Specifically, we rollout 10 episodes to calculate the mean and std of scores.
- The learning rate is set as 1e-4, the discounted factor γ is set as 0.99, the clipping ratio ϵ of PPO is set as 0.1, and the λ for GAE is set as 0.95.

For SAC:

- The batch size is 256. The Adam optimize is adopted for optimization.
- We optimize the model for one batch every step after the initialize 1e5 steps.
- The size of the replay buffer is 1e6.
- We evaluate training agent every 10,000 steps. Specifically, we rollout 10 episodes to calulate the mean and std of scores.
- The learning rate is set as 3e-4, the discounted factor γ is set as 0.99, and the coefficient of entropy is 0.2. The soft updated ratio τ is set as 5e-3. The β_1 for PER is set as 0.6, while β_2 is set as 0.4. Besides, β_2 will increase 0.001 after every environment step until 1.0.

3.9 Results and Discussion

The following figure shows the result for our SAC and PPO. For results, we use Gaussian Filter (σ = 1) to smooth the curve. One epoch is of 10000 environment steps.

Table 2: Best Score for Policy-based Methods

Environment	PPO Best Score	SAC Best Score
Ant-v2	839.44	6518.16
HalfCheetah-v2	1597.31	13969.18
Hopper-v2	3065.44	3439.75
Humanoid-v2	657.82	5863.54

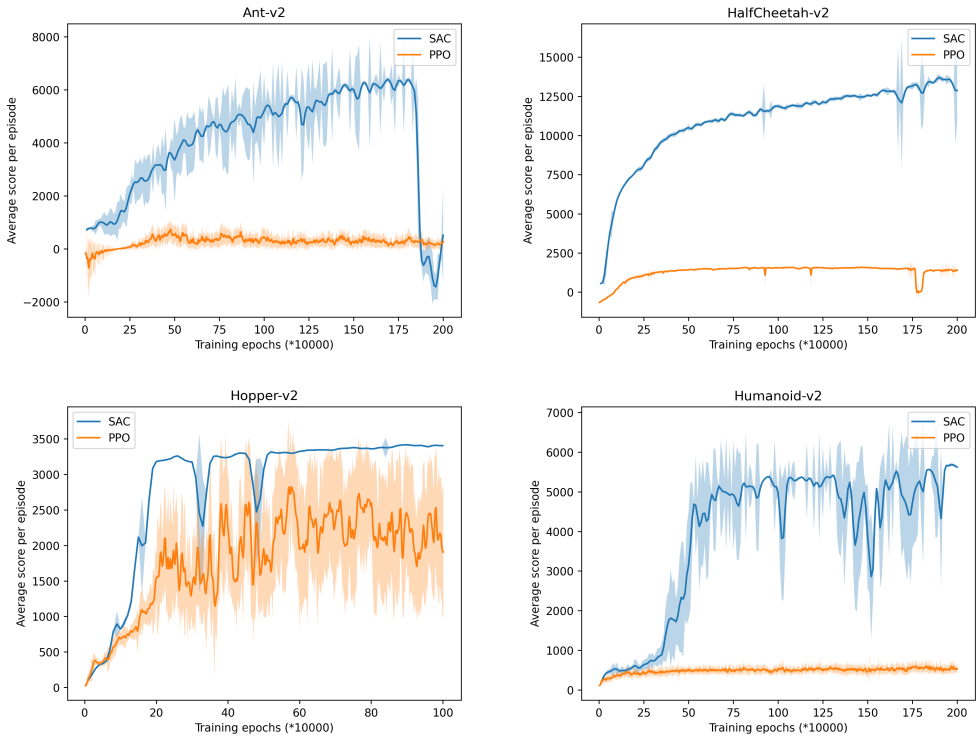

Figure 6: Different environment results for policy-based methods

From the results, we can find that SAC performs comparably to PPO on the easier tasks such as Hopper-v2, and outperforms them on harder tasks with a large margin. For example, PPO fails to make remarkable progress on Humanoid-v2, Ant-v2, and HalfCheetah-v2. In comparison, SAC turns out to learn considerably faster than PPO.

4 Conclusion

In this paper, we have tried model-free reinforcement learning using both value-based and policy-based algorithms. In conclusion, we implement and evaluate DQN and Dueling DDQN on three of Gym Atari environments with discrete action space. Besides, we also implement two representative policy-based algorithms: PPO and SAC and test them on four of continuous control environments in Gym MuJoCo. The experiment results show that our proposed methods can achieve satisfying scores on all these environments. Besides, the analysis results also suggest that both double and dueling structures are necessary to achieve DQN's performance. Morever, the off-policy learning schema and PER can promote the sample efficiency of policy-based algorithms.

References

[1] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. nature, 2015, 518(7540): 529-533.

[2] Van Hasselt H, Guez A, Silver D. Deep reinforcement learning with double q-learning[C]//Proceedings of the AAAI conference on artificial intelligence. 2016, 30(1).

[3] Wang Z, Schaul T, Hessel M, et al. Dueling network architectures for deep reinforcement learning[C]//International conference on machine learning. PMLR, 2016: 1995-2003.

[4] Han B A, Yang J J. Research on adaptive job shop scheduling problems based on dueling double DQN[J]. Ieee Access, 2020, 8: 186474-186495.

[5] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms[J]. arXiv preprint arXiv:1707.06347, 2017.

[6] Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning[J]. arXiv preprint arXiv:1509.02971, 2015.

[7] Haarnoja T, Zhou A, Hartikainen K, et al. Soft actor-critic algorithms and applications[J]. arXiv preprint arXiv:1812.05905, 2018.

[8] Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning[J]. arXiv preprint arXiv:1509.02971, 2015.

[9] Haarnoja T, Zhou A, Abbeel P, et al. Off-policy maximum entropy deep reinforcement learning with a stochastic actor[C]//Proceedings of the 35th International Conference on machine learning (ICML-18). 2018: 1861-1870.

[10] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms[J]. arXiv preprint arXiv:1707.06347, 2017.