Behavioral Proximal Policy Optimization with Kolmogorov-Arnold Networks

Katie Guo
Carnegie Mellon University
Pittsburgh, PA 15213
katieguo@andrew.cmu.edu

Ryan Lau
Carnegie Mellon University
Pittsburgh, PA 15213
rwlau@andrew.cmu.edu

Victoria Li
Carnegie Mellon University
Pittsburgh, PA 15213
vli3@andrew.cmu.edu

Abstract

Multi-layer perceptrons (MLPs) are ubiquitous in modern machine learning architectures, including in reinforcement learning (RL). Kolmogorov-Arnold Networks (KANs) are an emerging neural network architecture that potentially offer comparable performance while using fewer parameters and providing greater explainability. In this work, we explore the replacement of MLPs with KANs into the actor and critic for offline RL, specifically for the Behavioral Proximal Policy Optimization (BPPO) algorithm. We evaluate the performance and training efficiency of our models on the Gymnasium MuJoCo Hopper environment. Although KANs show promise, our study shows that they fail to achieve similar performance to the commonly used MLP for BPPO. Our code is available at `https://github.com/rlyss/10-701-Project`.

1 Introduction

Multi-layer perceptions (MLPs), or fully-connected feedforward neural networks, are key components of modern deep learning architectures and are the go-to model in machine learning for approximating nonlinear functions. Recently, however, Kolmogorov-Arnold Networks (KANs) have become a promising alternative for MLPs [5]. Whereas MLPs are inspired by the universal approximation theorem [2], KANs rely on the Kolmogorov-Arnold representation theorem [3]. As such, while MLPs have fixed non-linear activation functions on nodes and learnable weights on edges, KANs have learnable activation functions parameterized as splines on edges and sum operations on nodes. Therefore, KANs have the potential to achieve similar performance to MLPs while using fewer parameters and providing greater explainability.

Reinforcement Learning (RL) is a type of machine learning algorithm in which an intelligent agent learns to make decisions by interacting with an environment. Offline reinforcement learning, or batch reinforcement learning, is a paradigm of RL where the agent learns the optimal policy from a fixed, pre-collected dataset of interactions with the environment rather than actively interacting with the environment during training [4]. This makes offline RL algorithms suitable in situations where real-time exploration is impractical or data collection is costly. Behavior Proximal Policy Optimization (BPPO) is one such offline RL algorithm which monotonically improves behavioral policy in the manner of Proximal Policy Optimization[7].

RL algorithms often rely on feedforward neural networks which can be replaced with KANs. Although KANs face limitations such as suboptimal GPU performance and overfitting due to their more complex activation functions, they may be well-suited for RL, where model sizes tend to be simpler and smaller. In this work, we propose the use of KANs, instead of MLPs, as the actor and/or critic networks used in BPPO. To this end, we 1) implement the code necessary to achieve this and 2) run experiments replacing MLPs with KANs to compare their performance and training efficiency.

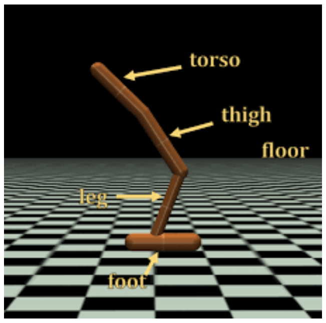

Figure 1: Labeled segments of Hopper [6].

2 Dataset

Gymnasium MuJoCo is a set of robotics-based reinforcement learning environments using the MuJoCo physics simulator. In particular, the Hopper robot is a two-dimensional one-legged robot with four body parts and three joints that connect them. The goal is to apply torque to the joints and so that the hopper moves the greatest possible distance within a certain time window. We use hopper-medium-v2 to evaluate our models.

Action An action represents the torque applied to one of the three joints: the thigh rotor, the leg rotor, and the foot rotor.

Observation Space The observation space consists of eleven elements: five elements quantifying the positions of the hoppers's body parts as well as the angle of the joints and six elements representing the velocities and angular velocities of these parts.

Rewards A positive reward is given based on the distance moved forward, and a negative reward is given based on distance moved backward. There is also a fixed reward for every timestep the hopper is "healthy" and a control cost to penalize the robot for taking actions that are too large.

Episode End The episode terminates when the hopper is unhealthy, which usually means that it has fallen over or its torso is no longer upright. Otherwise, it terminates in 1000 timesteps.

D4RL provides an offline reinforcement learning dataset for the hopper environment consisting of 1M samples of observation, action, reward tuples from a policy trained to approximately 1/3 the performance of an expert.

3 Methods

3.1 Background

3.1.1 Kolmogorov-Arnold Networks

Kolmogorov-Arnold Networks rely on the Kolmogorov-Arnold representation theorem, which states that any multivariate continuous function f with a bounded domain can be expressed as the sum of a finite number of compositions of single variable continuous functions. More specifically for continuous $f : [0, 1] \to \mathbb{R}$,

$$f(\mathbf{x}) = f(x_1, \ldots, x_n) = \sum_{q=0}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}\left(x_p\right) \right) \tag{1}$$

where $\phi_{q,p} : [0, 1] \to \mathbb{R}$ and $\Phi : \mathbb{R} \to \mathbb{R}$.

To take advantage of the Kolmogorov-Arnold representation theorem, a network must learn Φ_q and $\phi_{p,q}$. We can parameterize these univariate functions as B-spline curves, with learnable coefficients of local B-spline basis functions. Furthermore, we can generalize the network to arbitrary width and depth by simply specifying the input and output dimensions of each KAN layer and stacking multiple layers [5].

In particular, each edge in the computation graph can be represented by the 1D function applied to the input

$$\phi(x) = w_b b(x) + w_s \text{spline}(x) \tag{2}$$

where $b(x)$ is a basis function that can be set as a hyperparameter and spline(x) is a linear combination of basis splines. $b(x)$ is typically taken to be the SILU activation

$$b(x) = \text{silu}(x) = x/(1 + e^{-x}), \qquad \text{spline}(x) = \sum_i c_i B_i(x)$$

i ranges over the spline order hyperparameter k. Here, each c_i, w_b, w_s represents parameters that the model learns.

Each B_i is defined by G knot points specifying the interval of the spline and the locations where its polynomial components meet. The hyperparameter G is also known as the grid size and controls the coarseness of the function approximation over the input interval. Thus, for a KAN network with L layers and an input/output dimension of N for all layers, the total number of parameters will be $O(N^2 L(G + k))$.

Since KANs usually require a much smaller hidden dimension N, and each edge is a 1D function that can be easily plotted and visualized, they are much more interpretable and memory efficient than deep MLPs.

3.1.2 Behavior Proximal Policy Optimization (BPPO)

Offline RL represents the task of learning a behavior policy π given access to a fixed dataset of state, action, reward transitions $D = \{(s_t, a_t, s_{t+1}, r_t)_{t=1}^N\}$. However, offline RL algorithms are susceptible to the out-of-distribution (OOD) problem, which arises when the agent encounters state-action pairs not well represented in the dataset. If the model overestimates the Q-values for unseen actions, the policy may favor suboptimal or risky actions leading to poor generalizability.

Algorithms like Conservative Q-Learning[1] constrain the policy to actions close to the dataset by introducing a loss term that penalizes high-value actions outside of the dataset.

BPPO instead adapts online RL methods by constraining the closeness between the learned policy π and the behavior cloned (BC) policy $\hat{\pi}_\beta$ trained directly on the dataset.

$$\hat{\pi}_\beta = \text{argmax}_{\pi_\beta} \underset{(s,a)\sim D}{\mathbb{E}} [\log \pi(a \mid s)] \tag{3}$$

The BPPO algorithm is as follows [7].

Algorithm 1 Behavior **P**roximal **P**olicy **O**ptimization (BPPO)

```
1: Estimate behavior policy $\hat{\pi}_\beta$ by behavior cloning;
2: Calculate $Q$-function $Q_{\pi_\beta}$ by SARSA;
3: Calculate value function $V_{\pi_\beta}$ by fitting returns;
4: Initialize $k = 0$ and set $\pi_k \leftarrow \pi_\beta$ & $Q_{\pi_k} = Q_{\pi_\beta}$;
5: for $i = 0, 1, 2, \cdots, I$ do
6:     $A_{\pi_k} = Q_{\pi_k} - V_{\pi_\beta}$
7:     Update the policy $\pi$ by maximizing $L_k(\pi)$;
8:     if $J(\pi) > J(\pi_k)$ then
9:         Set $k = k + 1$ & $\pi_k \leftarrow \pi$;
10:        $Q_{\pi_k} = Q_{\pi_\beta}$;
11:    end if
12: end for
```

3.2 Model

Here, L_k represents a PPO-like loss with the BC divergence constraing

$$L_k(\pi) = \mathbb{E}_{s\sim\rho_D(\cdot),a\sim\pi_k(\cdot|s)}\left[\min\left(\frac{\pi(a \mid s)}{\pi_k(a \mid s)}A_{\pi_k}(s,a), \left(\frac{\pi(a \mid s)}{\pi_k(a \mid s)}, 1-2\epsilon, 1+2\epsilon\right)A_{\pi_k}(s,a)\right)\right] \tag{4}$$

and $J_\Delta(\pi', \pi) = J(\pi) - J(\pi')$ is the performance difference of two policies calculated by the Advantages

$$J_\Delta(\pi', \pi) = \mathbb{E}_{s\sim\rho_{\pi'}(\cdot),a\sim\pi'(\cdot|s)}[A_\pi(s,a)] \tag{5}$$

Since the main hurdle of offline RL lies in its ability to generalize, it would be meaningful to evaluate KAN's susceptibility to overfitting within the framework of offline RL due to their increased representational power.

3.2.1 Model architecture

KAN implementation We use an existing, unofficial implementation of KAN in PyTorch found here `https://github.com/Blealtan/efficient-kan`. It makes two main modifications to the implementation described in the original paper [5]. Firstly, to increase efficiency, L1 regularization is applied to the weights instead of the input samples. The official implementation also allows for this. Secondly, it uses Kaiming uniform instead of Xavier initialization because Kaiming uniform performed significantly better than Xavier initialization in preliminary experiments (∼97% to ∼20% accuracy on MNIST).

BPPO implementation We build on the existing implementation of BPPO found here `https://github.com/Dragon-Zhuang/BPPO`. The original implementation consists of four MLP networks trained sequentially.

1. `value`, V_{π_β} is a two hidden layer MLP with input dimension 11, hidden dimensions 512, output dimension 1, and ReLU activation function.
2. `Q`, Q_{π_β} is a one hidden layer MLP with input dimension 14, hidden dimension 1024, output dimension 1, and ReLU activation function.
3. `bc`, $\hat{\pi}_\beta$ is a one hidden layer MLP with input dimension 11, hidden dimension 1024 output dimension 6, and tanh activation function.
4. `BPPO`, π_k is a one hidden layer MLP with input dimension 11, hidden dimension 1024, output dimension 6, and tanh activation function.

The existing code uses an MLP with parameters `input_dim`, `hidden_dim`, `depth`, `output_dim`, and `final_activation`. The expected BPPO score on the hopper-medium-v2 environment using the above model is 93.9 ± 3.9 [7]. We recreate this baseline in Table 1.

Base models We modify the BPPO implementation by including a `use_kan` parameter to specify whether to use a KAN network instead for each model.[1] If so, we instead initialize a KAN using the same depth and input and output dimensions as the original MLP.

Since the each edge of a KAN layer is represented by more parameters than an MLP, to make the number of parameters of both networks relatively similar for a better comparison of model performance, we use half the hidden dimensions for each KAN model by default, so our hidden dimensions in order are: 256, 512, 512, and 512. Further hyperparameter studies on the hidden dimensions of select networks were subsequently conducted. Comparisons of the actual total parameter sizes are included in the discussion section. A full list of the hyperparameters for KANs is available in Table 5

To investigate the incremental differences in MLPs and KANs, we explore four network types.

1. **MLP** is the default MLP implementation from above
2. **BC-KAN** uses the pretrained value and Q networks from **MLP** and KANs for the bc and BPPO networks. In other words, our critic networks are MLPs and actor networks are KANs.

[1] `BPPO` and `bc` must be the same type of network.

Table 1: BPPO scores with default hyperparameters within 1000 time steps

Network Type	Best	End
MLP	92.7985	89.9154
BC-KAN	72.7264	69.8888
Q-KAN	78.0745	72.7212
KAN	82.8148	76.5817

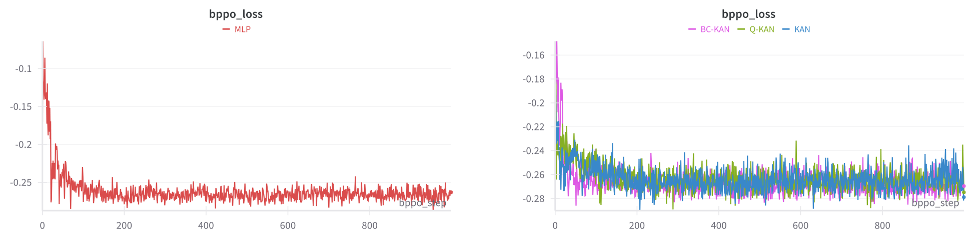

Figure 2: Losses of BPPO network for default models.

3. **Q-KAN** uses the pretrained value network from **MLP** and KANs for the Q, bc, and BPPO networks.
4. **KAN** uses KANs for all networks.

Since the results of each network may be dependent on the performance of the previously trained networks, analyzing the performance of these sequences of models with KAN's progressively replacing more of each network will allow us to better isolate and evaluate the effect of KANs over MLPs.

The hidden dimension of each network is dependent on the network itself as well as the type (MLP v.s. KAN). All other hyperparameters remain the same. Training was conducted on a $L4$ gpu instance with 8 vcpus.

3.2.2 Hyperparameters

All networks use the default parameters unless otherwise specified.

A supposed benefit of KANs is that they require fewer parameters than MLPs to learn a complex function. Since our proposed models have a similar number of parameters as shown in Table 3, we also experiment with the hidden dimension of the BC-KAN model to see if we can perform well despite fewer parameters.

4 Results

4.1 Base model experiments

4.1.1 Performance

The models, from best to worst performance, are: MLP, KAN, Q-KAN, and BC-KAN (Table 1). We also notice that although the losses of the models are similar (Figure 2), the scores are very different (Figure 3, 4). The final score for our MLP model matched the literature reference of 93.9 ± 3.9.

4.1.2 Training time

Training time (Table 2) is similar across all models .

4.1.3 Number of parameters

As part of our design, the number of parameters (Table 3) across our networks is relatively similar.

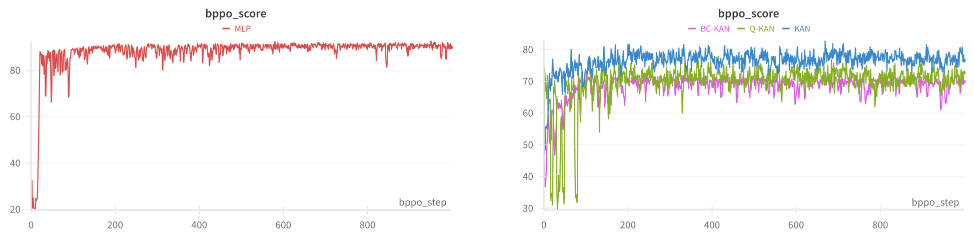

Figure 3: Scores at each time step for default models.

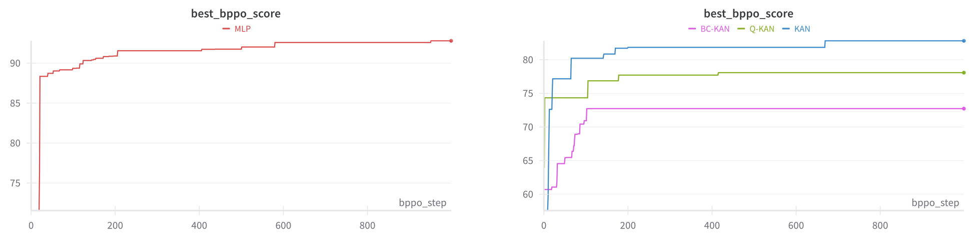

Figure 4: Best scores at each time step for default models.

Table 2: Training time on L4 GPU per network with default hyperparameters

	Training time			
Network	MLP	BC-KAN	Q-KAN	KAN
`value`	31:37	-	-	41:11
`Q`	40:39	-	46:11	46:41
`bc`	36:31	32:14	33:50	34:13
`BPPO`	3:05:44	2:50:24	2:51:04	3:13:30

Table 3: Number of parameters with default hyperparameters

	Size	
Network	MLP	KAN
`value`	531969	536576
`Q`	1065985	1079296
`bc`	1068038	1083392
`BPPO`	1068038	1083392

Table 4: Number of parameters of BC-KANs with different hidden dimensions

Hidden	Size of `BC/BPPO`	MLP size diff	BC Time (s)	Best Score
64	20736	-2094604	1827	47.55
128	74240	-1987596	1874	50.82
256	279552	-1576972	1942	76.65
512	1083392	+30708	2031	72.69
1024	3211252	+6391796	2270	63.46

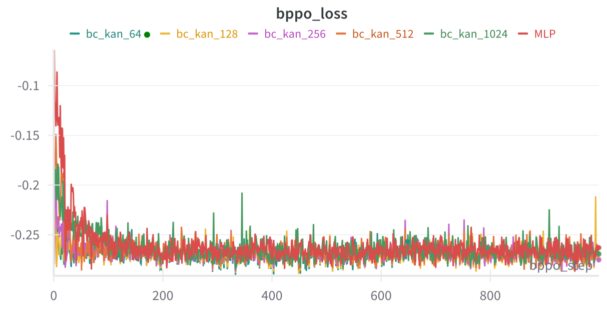

Figure 5: Losses of BPPO network for different hidden sizes.

4.2 Hidden dimension hyperparameter experiments

To investigate the parameter efficiency of KAN networks in an offline RL setting, we varied the hidden dimension of the KAN networks in **BC-KAN** and evaluated the final BPPO score after training. The hidden dimensions used are shown in Table 4.

Despite similar loss curves for all models (Figure 5), MLP vastly outperforms the BC-KAN models (Figure 7). Furthermore, BC-KAN models that are too large (hidden dimension = 1024) or too small (hidden dimension < 256) perform noticeably worse (Figure 6) than the medium-sized models with hidden dimensions of 256 and 512.

5 Discussion and Analysis

In this work, we experimented with using KANs as an alternative for MLPs in reinforcement learning, specifically in the BPPO algorithm in the Hopper MuJoCo environment, by testing four network variants that systematically replace MLPs with KANs: an MLP-only network, BC-KAN, Q-KAN, and a KAN-only network. To test the hypothesis that KANs perform better than MLPs in complex spaces with fewer parameters and to ensure a fair comparison, we controlled the number of hidden

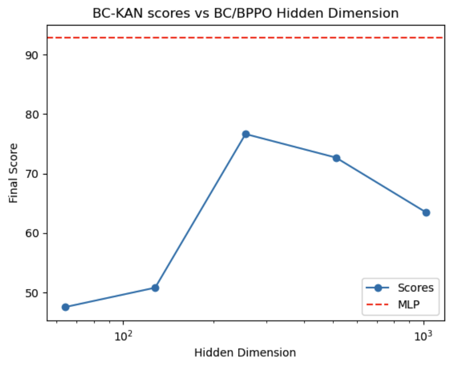

Figure 6: BC-KAN scores

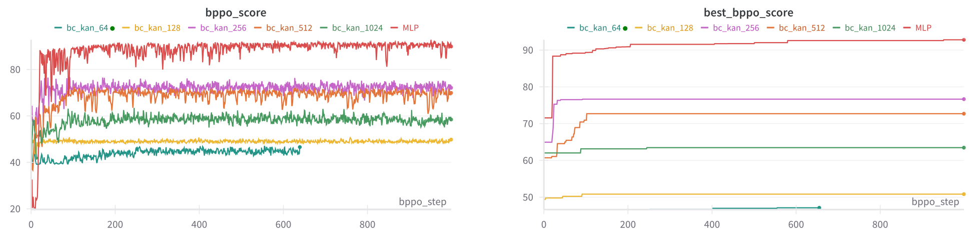

Figure 7: Scores at each time step for hopper-medium-v2.

layers to equalize total parameter count. We then modified the hidden dimensions of the KAN layers to investigate the effects of model sizes on performance.

Ultimately, none of the KAN models were able to match the performance of the MLP, even after tuning of the hidden dimensions. At the same time performance significantly varied across the different KAN configurations. Notably, the pure KAN model outperformed both BC-KAN and Q-KAN while maintaining similar training times, which suggests that the effectiveness of KANs may be more task dependent than MLPs. With proper hyperparameter tuning, KANs might be able to outperform MLPs. However, given that KANs have much more hyperparameters than MLPs, the increased search space could make hyperparameter tuning highly resource intensive and impractical as a simple out-of-the-box implementation like MLPs. It is also possible that the increased model complexity and representational power of KANs has lead to the models overfitting on the offline dataset, leading to poor generalizability as discussed previously.

We note from Figure 6 that peak model performance occurs at a KAN BC/BPPO hidden dimension of 256, which results in a model size of only 15% of the MLP, suggesting that KANs do require much fewer parameters and memory usage to achieve optimal performance.

Due to resource constraints, we were unable to fully optimize and determine the effects of other KAN hyperparameters such as the G and k values. Thus we cannot definitively conclude if MLPs are superior to KANs for offline RL. We also used Kaiming uniform instead of Xavier initialization due to its higher performance in preliminary experiments on the MNIST dataset, but a different initialization function could similarly drastically improve our KAN model performance on the MuJoCo environments.

Potential improvements to these limitations could be to refine initialization or regularization techniques that are more suitable to KANs, to fine-tune hyperparameters for MuJoCo environments, or explore a wider variety of MuJoCo ranging in complexity to see which scenarios KANs would excel in. An interesting factor to experiment with could be incorporating adaptive spline complexity to improve spline parameterization and to optimize parameter count versus performance.

Further experiments Based on our hyperparameter experiments, it seems that hyperparameters can vastly affect the overall performance of the network. Of particular interest is that the best performing BC-KAN network had a hidden size of 256. Although its performance was notably worse than both the MLP and KAN models, it is possible that using a pure KAN model where the last two networks use a hidden size of 256 could perform comparably to the pure MLP model while needing fewer parameters.

Additionally, our results seem to imply that KANs struggle to build off of pre-existing MLPs. However, the opposite may not necessarily be true. One such experiment to test this would be to use KANs for the value and Q models but MLPs for the bc and BPPO models.

Beyond that, while our initial results imply that KANs fail to match MLPs in performance, an important next step is further experimentation and generalization in more complex state-action spaces and in varying RL environments and tasks, since on the MNIST dataset, both models achieved similar performance, and it is possible that KANs are most useful only in certain specialized environments or tasks.

References

[1] George Tucker Sergey Levine Aviral Kumar, Aurick Zhou. Conservative q-learning for offline reinforcement learning. *ArXiv*, arXiv:2006.04779, 2020.

[2] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[3] A. K. Kolmogorov. On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSSR*, 114:369–373, 1957.

[4] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems, 2020.

[5] Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y. Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks. 2024.

[6] Eugene Vinitsky, Yuqing Du, Kanaad Parvate, Kathy Jang, Pieter Abbeel, and Alexandre Bayen. Robust reinforcement learning using adversarial populations, 2020.

[7] Zifeng Zhuang, Kun Lei, Jinxin Liu, Donglin Wang, and Yilang Guo. Behavior proximal policy optimization. *arXiv preprint arXiv:2302.11312*, 2023.

Appendix

Hyperparameters

Table 5: Default hyperparameters for KAN

Hyperparameters	Default
`grid_size`	1
`spline_order`	1
`scale_noise`	0.1
`scale_base`	1.0
`scale_spline`	1.0
`base_activation`	SiLU
`grid_eps`	0.02
`grid_range`	[-1, 1]

Table 6: Default hyperparameters for MLP BPPO

Network	Hyperparameters	Default
value	v_steps	7e5
	v_hidden_dim	512
	v_depth	3
	v_lr	1e-4
	v_batch_size	512
Q	q_bc_steps	7e5
	q_hidden_dim	1024
	q_depth	2
	q_lr	1e-4
	q_batch_size	512
	target_update_freq	2
	tau	0.005
	gamma	0.99
	is_offpolicy_update	False
bc	bc_steps	2e5
	bc_hidden_dim	1024
	bc_depth	2
	bc_lr	1e-4
	bc_batch_size	1024
BPPO	BPPO_steps	1e3
	BPPO_hidden_dim	1024
	BPPO_depth	2
	BPPO_lr	1e-4
	BPPO_batch_size	512
	clip_ratio	0.25
	entropy_weight	0.0
	decay	0.96
	omega	0.9
	is_clip_decay	True
	is_BPPO_lr_decay	True
	is_update_old_policy	True
	is_state_norm	False