Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

Self-Adaptive Double Bootstrapped DDPG

Zhuobin Zheng[12], **Chun Yuan**[2], **Zhihui Lin**[12], **Yangyang Cheng**[12], **Hanghao Wu**[12]
[1] Department of Computer Science and Technologies, Tsinghua University
[2] Graduate School at Shenzhen, Tsinghua University
{zhengzb16, lin-zh14, cheng-yy13, whh16}@mails.tsinghua.edu.cn, yuanc@sz.tsinghua.edu.cn

Abstract

Deep Deterministic Policy Gradient (DDPG) algorithm has been successful for state-of-the-art performance in high-dimensional continuous control tasks. However, due to the complexity and randomness of the environment, DDPG tends to suffer from inefficient exploration and unstable training. In this work, we propose Self-Adaptive Double Bootstrapped DDPG (SOUP), an algorithm that extends DDPG to bootstrapped actor-critic architecture. SOUP improves the efficiency of exploration by multiple actor heads capturing more potential actions and multiple critic heads evaluating more reasonable Q-values collaboratively. The crux of double bootstrapped architecture is to tackle the fluctuations in performance, caused by multiple heads of spotty capacity varying throughout training. To alleviate the instability, a self-adaptive confidence mechanism is introduced to dynamically adjust the weights of bootstrapped heads and enhance the ensemble performance effectively and efficiently. We demonstrate that SOUP achieves faster learning by at least 45% while improving cumulative reward and stability substantially in comparison to vanilla DDPG on OpenAI Gym's MuJoCo environments.

1 Introduction

Reinforcement learning (RL) [Sutton and Barto, 1998] tackles the problem of how agents learn mappings from observations to actions in order to maximize the cumulative reward when interacting with the environment. Deep reinforcement learning adopts large neural network policies and value functions instead of classical linear function approximators for robust generalization capacity to deal with high-dimensional complex tasks. Deep RL has achieved great successes in a wide range of challenging problems, such as Atari games [Mnih *et al.*, 2015], Go game [Silver *et al.*, 2017] and robotic control tasks [Levine *et al.*, 2016].

In particular, model-free reinforcement learning is an appropriate method for dealing with goal-directed decision-making problems only according to the reward signals without extra supervision. Deep Deterministic Policy Gradient (DDPG) [Silver *et al.*, 2014; Lillicrap *et al.*, 2016], as one of the model-free off-policy algorithms, is more sample-efficient by utilizing actor-critic architecture with experience replay and has become increasingly prevalent for state-of-the-art performance in continuous control tasks. However, DDPG is susceptible to the complexity and randomness of the environment, which results in unstable performance and unguaranteed convergence. This issue means extensive hyperparameter tuning is essential for good results [Islam *et al.*, 2017].

In this paper, we aim to improve both sample efficiency and stability of DDPG. Specifically, based on DDPG, we introduce bootstrap which is demonstrated to be advantageous for deep exploration [Osband *et al.*, 2016] and propose an algorithm, Self-Adaptive Double Bootstrapped DDPG (SOUP), to enable efficient exploration together with stability. SOUP extends the single-head actor-critic architecture of DDPG to two bootstrapped networks both with multiple heads branching off independently and trained diversely. When exploring the environment, various possible action candidates are produced by actor heads according to the same state and evaluated weightedly by critic heads to determine the high-potential action collaboratively. To alleviate the inaccurate evaluation problem caused by critic heads of uneven and varying capacity, self-adaptive confidence is introduced to either increase or decrease the weights in ensemble evaluation dynamically according to the feedback. In this case, more justifiable Q-values are estimated by our approach to stabilize training. We evaluate and demonstrate the effectiveness and efficiency of SOUP on OpenAI Gym's [Brockman *et al.*, 2016] MuJoCo continuous control environments, Hopper and Walker2D [Todorov *et al.*, 2012].

The contributions of the paper are summarized below:

- We extend both the actor and the critic of DDPG to bootstrapped neural networks for deep exploration.
- Based on the multi-head architecture, we utilize an ensemble Q-value evaluation to determine a potential action for increasing the efficiency of experience replay.
- To address the inaccurate evaluation problem caused by critic heads of spotty capacity, we propose self-adaptive confidence strategy to calibrate weights automatically.
- We conduct extensive experiments to evaluate the performance of our approach from different perspectives, including comparisons on bootstrapped models, confidence strategies and multiple heads.

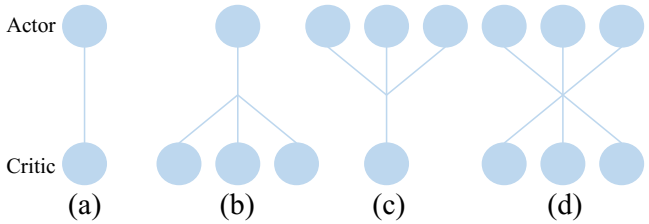


Figure 1: Comparison of different bootstrapped models with three heads: (a) Vanilla DDPG; (b) MA-BDDPG (multi-critic); (c) Multi-DDPG (multi-actor); (d) DBDDPG (ours).

2 Related Work

An intuitive approach to alleviate inefficient exploration issue is parallelism, which means training multiple agents to collect experiences simultaneously. Bootstrapped DQN [Osband *et al.*, 2016] is a first attempt to introduce bootstrap [Efron and Tibshirani, 1994] to the discrete RL method, DQN [Mnih *et al.*, 2015]. It leverages uncertainty estimates of the neural network for deep exploration benefiting from a multi-head architecture, which simulates training different DQNs independently. In recent years, many exploration strategies have been proposed in different directions such as self-supervised curiosity [Pathak *et al.*, 2017], parameter noise [Plappert *et al.*, 2018], unsupervised auxiliary tasks [Jaderberg *et al.*, 2017] and Q-ensemble [Chen *et al.*, 2018].

Though DDPG seeks to ease the instability by target network technique, recent work points out that it sometimes suffers from drastically oscillations in performance on unstable environments [Henderson *et al.*, 2017]. Q-prop uses a control variate to stabilize DDPG by reducing the variance of gradient estimator [Gu *et al.*, 2017]. A3C achieves a stabilizing effect by training parallel agents with accumulated updates asynchronously [Mnih *et al.*, 2016].

Recent work has attempted to improve DDPG with bootstrap. MA-BDDPG [Kalweit and Boedecker, 2017] adopts bootstrapped DQN as the critic in DDPG and extends replay memory by a model-based approach for sample efficiency. Nevertheless, it utilizes an uncertainty variate to limit inaccurate model usage, highly noisy synthesis data still fluctuates the training erratically with insufficient exploration. Multi-DDPG [Yang *et al.*, 2017] employs a multi-actor architecture for multi-task purpose while merely transforming the critic to multiple outputs. If the critic, however, is not sufficiently trained, it cannot capture all informative feedback for training multiple actors of spotty capacity, which induces instability.

In contrast to the inadequate single bootstrapped models, our work enhances both the actor and the critic as bootstrapped networks (see Figure 1 for clear comparison). Benefiting from double bootstrapped DDPG (DBDDPG), more potential action candidates are generated by multiple actor heads and evaluated by multiple critic heads collaboratively, which improves the efficiency of exploration and stability.

3 Background

3.1 Notation

We model a standard reinforcement learning setup including an agent interacting with an environment E and receiving a reward r at every time step t, as a *Markov decision process* (MDP). MDP can be defined as $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, p_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$ is the state transition function, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the reward function, $\gamma \in (0, 1]$ is the discount factor and p_0 is the initial state distribution. Besides, we consider a policy as $\pi : \mathcal{S} \mapsto \mathcal{A}$, which means the agent outputs an action after observing a state. Note that a policy may be stochastic, but in our case, we only consider deterministic policy. After applying a_t in s_t, the agent receives a new state s_{t+1} and a reward r_t, where $s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_t)$ and $r_t = \mathcal{R}(s_t, a_t)$. We denote the return from a state s_t as the cumulative γ-discounted reward $R_t = \sum_{i=t}^{T} \gamma^{i-t} \mathcal{R}(s_i, a_i)$. The goal of reinforcement learning is to learn an optimal policy that maximizes the expected return from the initial state $J = \mathbb{E}_{r_i, s_i \sim E, a_i \sim \pi}[R_0]$. The action-value function $Q^\pi(s_t, a_t)$, also called Q-function, is the expected return starting from state s_t, taking action a_t, and thereafter following policy π:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{r_{i \geqslant t}, s_{i>t} \sim E, a_{i>t} \sim \pi}[R_t | s_t, a_t]. \tag{1}$$

More generally, Equation (1) can be described as a recursive format by *Bellman Equation*:

$$\begin{aligned} Q^\pi(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}[\mathcal{R}(s_t, a_t) \\ + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q^\pi(s_{t+1}, a_{t+1})]]. \end{aligned} \tag{2}$$

3.2 Deep Deterministic Policy Gradient

Model-free reinforcement learning achieves the goal directly without modeling the environment dynamics. Our approach is built upon the DDPG algorithm [Lillicrap *et al.*, 2016], a model-free, off-policy actor-critic [Konda and Tsitsiklis, 2000] approach consisting of a Q-function (the critic) and a policy function (the actor) to tackle high-dimensional continuous action tasks. The actor μ and the critic Q, are estimated by deep function approximators, parameterized by θ^μ and θ^Q. Besides, DDPG uses slowly updated target networks, parameterized by $\theta^{\mu'}$ and $\theta^{Q'}$, to stabilize the training. Experience replay technique stores the experience tuples (s_t, a_t, s_{t+1}, r_t) in the replay memory, from where a minibatch of n samples are generated randomly to break up the temporal correlations within different training episodes for variance reduction.

DDPG optimizes the critic by minimizing the loss:

$$\mathcal{L}(\theta^Q) = \frac{1}{n} \sum_i (y_i - Q(s_i, a_i | \theta^Q))^2, \tag{3}$$

where

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} | \theta^{\mu'}) | \theta^{Q'}), \tag{4}$$

and the actor by using policy gradient [Silver *et al.*, 2014]:

$$\nabla_{\theta^\mu} \leftarrow \frac{1}{n} \sum_i \nabla_a Q(s_i, a | \theta^Q)|_{a=\mu(s_i|\theta^\mu)} \nabla_{\theta^\mu} \mu(s_i | \theta^\mu). \tag{5}$$

The target networks are slowly updated by:

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'}, \\ \theta^{\mu'} &\leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}, \end{aligned} \tag{6}$$

with $\tau \in (0, 1]$.

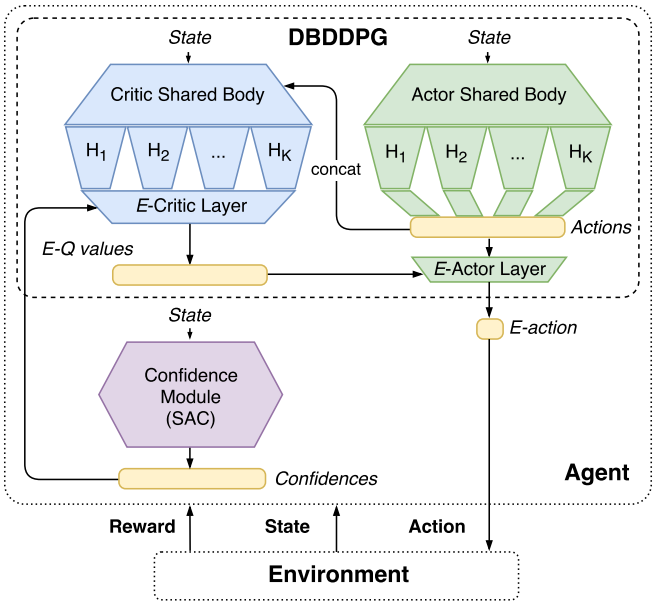


Figure 2: Structure of SOUP. When the actor (green) observes a state, each actor head generates an action vector. Given the same state, the critic (blue) concatenates the vectors in the hidden layer and produces a Q-value matrix while the confidence module (purple) outputs a confidence vector. Combining these two tensors, E-Critic layer generates an E-Q-value vector, followed by E-Actor layer determining the final E-action with the maximum E-Q-value.

4 Self-Adaptive Double Bootstrapped DDPG

In this paper, we propose Self-Adaptive Double Bootstrapped DDPG, abbreviated with SOUP (see Figure 2 for an overview of the approach), to enhance exploration while retaining stability. To explain the full algorithm more clearly, DBDDPG is first described in Section 4.1, as the major framework combining bootstrapped architecture with DDPG. Following that in Section 4.2, we demonstrate how self-adaptive confidence works for balancing the weights of bootstrapped heads.

4.1 Double Bootstrapped DDPG

The core architecture of this work is DBDDPG, which introduces bootstrap [Efron and Tibshirani, 1994] completely to both actor and critic (see Figure 1 for clear comparison with previous work). They both consist of a shared body for feature extraction and K heads respectively, $Q_{1:K}$ and $\mu_{1:K}$, but with different random initialization. The advantage of the shared architecture is that it can capture most of the benefits of bootstrapped architecture with substantially fewer parameters and less computation cost comparing with training multiple models directly. Similar to DDPG, the target networks of the heads, $Q'_{1:K}$ and $\mu'_{1:K}$, are slowly updated for stability.

Ensemble Q-value Evaluation

When the actor interacts with the environment given a state s_t, K actor heads generate multiple action candidates $\boldsymbol{A}_t = \{a_t^k | a_t^k \in \mathbb{R}^{d_{\mathcal{A}}}, d_{\mathcal{A}} = \dim \mathcal{A}\}_{k=1}^K$, which are embedded to the critic [Lillicrap *et al.*, 2016]. Multiple critic heads output a Q-value matrix $\boldsymbol{V}_t \in \mathbb{R}^{K \times K}$ according to s_t and $\boldsymbol{A}_t$. Ensemble critic (E-Critic) layer performs a weighted sum operation on the Q-value matrix to transform it into an ensemble-Q-value (E-Q-value) vector $\boldsymbol{v}_t \in \mathbb{R}^K$, which represents the potential values of the action candidates in $\boldsymbol{A}_t$. The E-action a_t with the maximum E-Q-value determined by ensemble actor (E-Actor) layer according to Equation (7), is chosen to execute receiving a new state s_{t+1} and a reward r_t. Moreover, a random mask $\boldsymbol{m}_t = (m^1, m^2, ..., m^K)_t$ is generated with Bernoulli distribution concurrently. A new transition $(s_t, a_t, s_{t+1}, r_t, \boldsymbol{m}_t)$ is stored in replay memory. In this case, only the experiences with high-potential actions are saved instead of the spotty ones that have the uneven potential to get rewards, which improves the efficiency of experience replay.

$$a_t = \arg\max_a \{\sum_{i=1}^K c_t^i Q_i(s_t, a|\theta_i^Q)|_{a=\mu_k(s_t|\theta_k^\mu)}\}_{k=1}^K, \quad (7)$$

where $c_t^i \in (0, 1]$ is the confidence of critic head Q_i for ensemble Q-value evaluation. Normally, we set $c_t^i = 1.0$.

In the training phase, the k-th pair of actor head and critic head is randomly activated to learn during every episode. The selected heads together with their respective shared bodies and target networks are trained as a vanilla DDPG given a minibatch of samples. The i-th experience with mask $m_i^k = 0$ is ignored or negated for the purpose of bootstrap. A more detailed procedure can be viewed in Algorithm 1.

Bootstrapped Architecture Drives Deep Exploration

Bootstrap [Efron and Tibshirani, 1994] generally works relying on the random sampling with replacement process, which is demonstrated to be helpful for variance reduction in ensemble training such as Bootstrap Aggregating algorithm. [Osband *et al.*, 2016] analyzed and empirically suggested that this requirement can be simply attained by different random initialization of the network weights of different heads. We adopt this technique as a prior to induce initial diversity. In this case, the bootstrapped architecture of DBDDPG is a simulation of training multiple models in parallel on different sub-dataset by stochastic minibatch independently. Furthermore, actor heads can generalize diversely in the action space, leading to more potential directions for deep exploration.

Double Bootstrapped Architecture Enhances Stability

Intuitively, a couple of actor head and critic head in double bootstrapped architecture are always bound together as a single DDPG. Note that though such multiple "DDPGs" share the same replay memory, they are trained on different experiences according to the random mask. Moreover, "DDPGs" not only generate diverse available action candidates, which is more sample-efficient than single-actor bootstrapped model, but also determine the outputs by multiple critic heads' weighted evaluation collaboratively. In contrast to single-critic bootstrapped model, the potential of an action is estimated more reasonably and accurately by various critic heads, instead of spotty estimations from a single critic. In other words, double bootstrapped architecture improves stability by alleviating the uncertainty of evaluation and averting irreversible degradation of a single critic in unstable systems.

4.2 Self-Adaptive Confidence

Though DBDDPG achieves more cumulative reward than DDPG, we find that it sometimes suffers from oscillations

during the preliminary phase of training. We ascribe this problem to the extremely difficult part of double bootstrapped architecture, which is caused by critic heads of spotty capacity varying throughout training. In this case, Q-values evaluated by critic heads that are not well trained or trapped into local optima, are inaccurate for ensemble decision. This uninformative and defective supervision leads to oscillations since the model is trained in the wrong direction.

Confidence Strategies

The main motivation of this idea is to counteract the adverse effects caused by critic heads of spotty capacity when estimating the E-Q-values as Equation (7). c_t^i can be described as the confidence, which means how confident the i-th critic head is in its evaluation, Q-value, according to the action. It is straightforward to balance the spotty evaluation by adjusting the confidence. We try strategies such as no confidence, fixed confidence and decayed confidence. However, none of these approaches achieve promising stability against oscillations. Notably, they tend to suffer from the over-confidence problem during certain periods. In other words, the critic heads always retain high confidence in their evaluation whether reasonable and accurate or not. In this case, spotty ensemble Q-values of uneven estimations may easily mislead the supervision and result in instability.

Self-Adaptive Strategy

To tackle this problem, DBDDPG requires an external "controller" taking full advantage of the informative feedback to calibrate the confidence automatically. We further propose Self-Adaptive Confidence (SAC), a new strategy designed for bootstrapped architecture. Different from the previous methods, SAC adjusts the confidence dynamically according to the reward signals (see Figure 2). The confidence module is constructed similarly to single-head actor network with K outputs $\boldsymbol{c}_t = \{c_t^k | c_t^k \in (0, 1]\}_{k=1}^K$ as the bootstrapped heads.

Perception Phase When interacting with the environment, the confidence module perceives a state s_t and generates a confidence vector $\boldsymbol{c}_t$ concurrently for assisting critic heads' evaluation. Any confidence c_t^k is created based on the performance history of the k-th critic head, including the feedback and confidence. Following that, the E-Critic layer estimates the E-Q-values for action selection, by performing a product operation on the confidence vector $\boldsymbol{c}_t$ and Q-value matrix $\boldsymbol{V}_t$.

Calibration Phase After perception phase, reward signal is returned that represents whether the previous action and confidence are reasonable or not. This feedback supervises the training of both DBDDPG and the confidence module. During the training phase, the confidence network C is updated by policy gradient [Sutton *et al.*, 2000] as Equation,

$$\theta^C \leftarrow \theta^C + \alpha \nabla_{\theta^C} log \pi_{\theta^C}(s_t, a_t) Q^\pi(s_t, a_t), \qquad (8)$$

where α is the learning rate. Note that in the experiments, we use reward r_t as an unbiased sample of $Q^\pi(s_t, a_t)$, as the REINFORCE algorithm [Williams, 1992]. Following this policy, SAC eliminates the over-confidence problem by calibrating the weights dynamically and more reasonably. Intuitively, the confidence of critic head increases due to positive reward while decreases due to negative reward for penalization. Therefore, self-adaptive confidence stabilizes DBDDPG by more justifiable E-Q-value evaluation throughout any period of training, which leads to robust generalization.

Algorithm 1 Self-Adaptive Double Bootstrapped DDPG

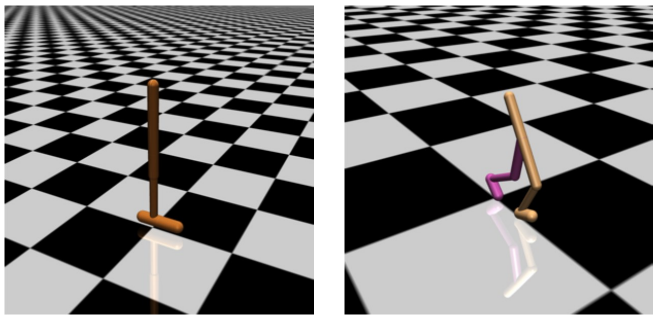
Input: number of heads K, maximum training episode E, masking distribution M and mini-batch size n.
Initialize: Randomly initialize critic and actor networks both with K heads $\{\theta_k^Q, \theta_k^\mu\}_{k=1}^K$, assigning copies to target networks $\{\theta_k^{Q'}, \theta_k^{\mu'}\}_{k=1}^K$. Initialize confidence network θ^C and replay buffer R.
for episode $e = 1, E$ **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_0
  Randomly select the k-th pair of critic and actor heads
  for step $t = 1, T$ **do**
    K actor heads generate candidates
    Select action a_t according to (7) and apply $\mathcal{N}$
    Execute a_t then observe reward r_t and state s_{t+1}
    Sample bootstrapped mask $\boldsymbol{m}_t \sim M$
    Store transition $(s_t, a_t, s_{t+1}, r_t, \boldsymbol{m}_t)$ in R
    Sample a random minibatch of n transitions
    Update critic head Q^k according to (3) and (4)
    Update actor head μ^k according to (5)
    Update the k-th target networks according to (6)
    Update the confidence network according to (8)
  end for
end for

5 Experiments

We evaluate our algorithm on following continuous robotic environments implemented in MuJoCo simulator [Todorov *et al.*, 2012] from OpenAI Gym [Brockman *et al.*, 2016] (see Figure 3 for a visualization).

Hopper-v1 In this environment, a two-dimensional one-legged robot is rewarded by hopping forward as fast as possible ($\mathcal{S} \subseteq \mathbb{R}^{11}, \mathcal{A} \subseteq \mathbb{R}^3$).

Walker2d-v1 This environment extends Hopper to a bipedal robot in 2D-space, rewarded by walking forward as fast as possible ($\mathcal{S} \subseteq \mathbb{R}^{17}, \mathcal{A} \subseteq \mathbb{R}^6$).



(a) (b)

Figure 3: Illustration of locomotion tasks on MuJoCo continuous control environment: (a) Hopper and (b) Walker2D.

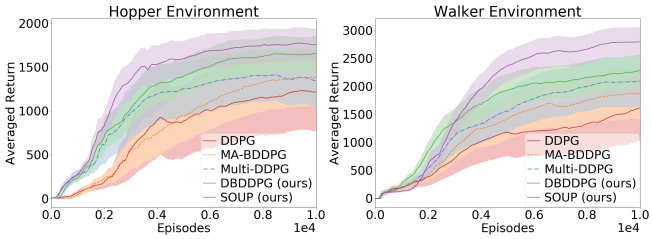


Figure 4: Performance of different bootstrapped models with five heads in different environments, Hopper (left) and Walker2D (right). The shaded area depicts the mean $\pm$ the standard deviation. SOUP outperforms other models in both reward and stability.

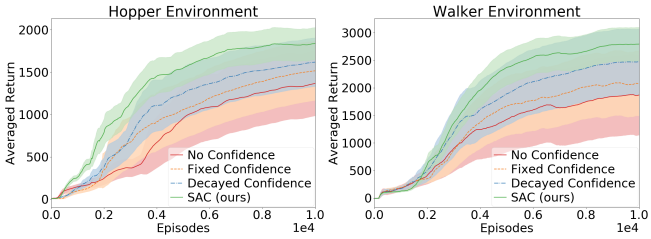


Figure 5: Performance of DBDDPG with five heads and different confidence strategies in different environments, Hopper (left) and Walker2D (right). Self-adaptive confidence outperforms common confidence methods in cumulative reward and learning speed.

We conduct the following experiments to evaluate and analyze the performance of SOUP:

1. we compare our model with and without self-adaptive confidence against other bootstrapped models.
2. we compare the impacts of different confidence strategies mentioned in Section 4.2.
3. we compare the performance achieved by a different number of bootstrapped heads.

To ensure comparability, unless otherwise stated, we keep the common hyperparameters and the network architecture the same in the experiments. We denote the hidden layer sizes as $(N, \boldsymbol{M})$ where the bold number indicates the head layer size. For double bootstrapped DDPG, we use (256, 256, **128**) for the critic and (256, **128**) for the actor. Adam [Kingma and Ba, 2015] is adopted for training actor and critic networks with a learning rate of $1e^{-4}$ and $3e^{-4}$ respectively. We use a discount factor $\gamma = 0.99$, a soft update rate $\tau = 1e^{-3}$, a minibatch size $n = 1024$ and a replay memory size $R = 1e^{6}$. All activation layers use Leaky ReLU [Maas *et al.*, 2013] and the output layer of the actor uses TanH followed by the scale and shift operations. In order to emphasize improvements in exploration efficiency brought by the bootstrapped architecture, exploration at action level is governed by simple Gaussian noise with a decayed rate. We measure the performance by averaged return, maximum return, speedup, episode that first exceeds a threshold and evaluate the stability by standard deviation over 10k episodes with different random seeds varying throughout training. Figure 4, 5, 7 represent the mean return by lines and std return by shaded areas.

5.1 Testing with Bootstrapped Models

Though our work is inspired by Bootstrapped DQN [Osband *et al.*, 2016], it is not taken into comparison since DQN works for discrete action space while DDPG works on continuous tasks. For simplicity, MA-BDDPG [Kalweit and Boedecker, 2017] and Multi-DDPG [Yang *et al.*, 2017] can be regarded as multi-critic and multi-actor single bootstrapped models. Our approach DBDDPG extends both actor and critic networks to bootstrapped architectures.

Figure 4 shows the performance of DBDDPG with and without self-adaptive confidence against other models. We equip all bootstrapped networks with $K = 5$ heads. In Hopper, MA-BDDPG performs similarly to DDPG in the beginning due to the insufficient single-actor exploration and noisy synthetic rollouts. In contrast, Multi-DDPG achieves higher reward benefiting from multi-actor exploration. Nonetheless, it doesn't keep reward raising throughout the training since single critic cannot always evaluate Q-values accurately for multiple actor heads of various capacity. However, with the same multi-actor architecture, DBDDPG explores analogously to Multi-DDPG at first but significantly outperforms it after training, since multi-critic ensemble estimations are much more accurate and reliable for multi-actor potential evaluation of actions. After combining with self-adaptive confidence, SOUP scores the highest averaged reward faster than DBDDPG while retaining stability. In Walker, though SOUP explores less than DBDDPG in the initial phase which may due to the slightly poor initialization, it exceeds soon because of more justifiable evaluation enhanced by self-adaptive confidence. This self-correction of ensemble weights effectively supervises the training.

Table 1 tabulates detailed results of the bootstrapped models. In table 1, DBDDPG with five heads consistently outperforms other models in terms of sample efficiency. Comparing with vanilla DDPG, it achieves higher averaged rewards by at least 36% in Hopper and 42% in Walker, with the same computation cost for training. Moreover, after adopting self-adaptive confidence, SOUP substantially accelerates the learning with speedup of factor $\alpha \in [0.45, 0.65]$. It improves mean return by at least 44% in Hopper and 73% in Walker in comparison to DDPG without inducing instability.

5.2 Testing with Confidence Strategies

In this experiment, we evaluate the performance of following confidence strategies including:

- **No Confidence** $c^i = 1.0$ as original DBDDPG.
- **Fixed Confidence** $c^i = \alpha$, $c^{j(j\neq i)} = \frac{1-\alpha}{K-1}$ when estimating action a_t^i and vice versa. We fix $\alpha = 0.5$.
- **Decayed Confidence** c^i diminishes from 1.0 to $\frac{1}{K}$ continuously with a decayed factor $\tau = 0.9995$.
- **Self-Adaptive Confidence** c^i is automatically adjusted by confidence network according to the reward signal.

Figure 5 shows the performance of DBDDPG with different confidence methods, especially in stability denoted by the shaded area. The larger area means more unstable training. In Figure 5, except self-adaptive confidence, other methods perform with more fluctuations especially during either preliminary or middle phase. As the aforementioned explanation, this phenomenon is caused by inaccurate Q-values evaluated from critic heads of spotty capacity such as in premature or

Env.	Metric	DDPG	MA-BDDPG	Multi-DDPG	**Ours (*K*=5)**	**Ours (*K*=5*)**	**Ours (*K*=10)**	**Ours (*K*=10*)**
Hopper	Avg Return	1217 ± 762	1380 ± 622	1346 ± 597	1656 ± 418	1758 ± 349	1982 ± 269	$\mathbf{2154 \pm 236}$
	Max Return	3920	3892	3860	4000	4269	4365	**4399**
	Episode	2486	2460	1637	1466	**1114**	1155	1366
	Speed Up	-	0.07	0.41	0.48	0.61	0.56	**0.65**
Walker	Avg Return	1611 ± 683	1869 ± 520	2095 ± 553	2286 ± 428	2795 ± 403	2835 ± 367	$\mathbf{3218 \pm 315}$
	Max Return	5883	5626	5942	6072	6186	6309	**6406**
	Episode	2307	2111	2001	1465	1385	1212	**900**
	Speed Up	-	0.42	0.49	0.63	0.65	0.65	**0.74**

Table 1: Results of different models based on DDPG in different environments (Hopper and Walker) in the 10k episodes. MA-BDDPG and Multi-DDPG are equipped with the same number of heads ($K = 5$). We denote DBDDPG with self-adaptive confidence by adding *. **Episode** represents the first episode to cross specific reward threshold (Hopper-2000; Walker-3000) according to [Gu *et al.*, 2017]. **Speedup** is measured by averaged return crossing specific averaged reward threshold (Hopper-1000; Walker-1500) in comparison to DDPG.

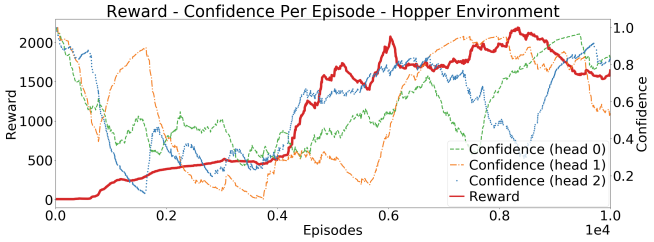


Figure 6: The dynamic changes of confidence in Hopper environment. Note that, the curves have been smoothed for more clear visualization. SOUP with three heads fine-tunes the confidence of critic heads according to the reward, which stabilizes the training.

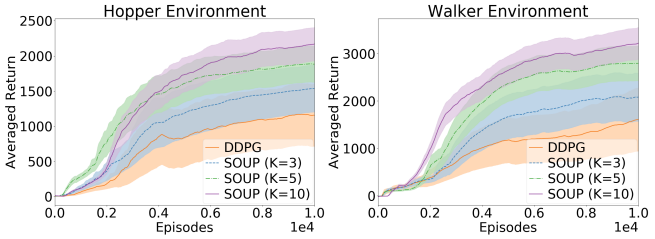


Figure 7: Performance of SOUP with a different number of bootstrap heads K in different environments, Hopper (left) and Walker2D (right). SOUP with ten heads outperforms models with fewer heads in cumulative reward and stability.

sub-optimal convergence. The defective supervision misleads the training which results in drastically oscillations.

Self-adaptive confidence substantially improves the stability of DBDDPG, denoted by narrower shaded area in Figure 5. Different from other strategies with inflexible confidence, our approach achieves both dynamically adaptable adjustment and reasonable self-correction, according to the reward and the capacity of the corresponding head. Figure 6 exactly demonstrates the flexibility and adaptability of our method during training. At first, the confidence fluctuates due to the random initialization. After the warm-up, they are dynamically calibrated by policy gradient, such as decreased for penalization against over-confidence and increased for positive reward. In other words, self-adaptive confidence adjusts the ensemble weights to enhance more justifiable estimations, which achieves robust stability effectively and efficiently.

5.3 Testing Multi-Head Architecture

In this experiment, we evaluate the performance of SOUP with a different number of bootstrapped heads. Figure 7 shows that generally with more heads, our approach achieves higher averaged reward faster by more efficient exploration, which further enhances the efficiency of experience replay. Nonetheless, in Hopper, the 10-head explores inefficiently in the beginning, comparing with the 5-head. We ascribe this problem to the fact that five heads are sufficient for exploring this task. In this case, the 10-head is more time-consuming, which requires more episodes to balance the capacities of more heads, than the 5-head. However, the 10-head achieves better performance after training in both tasks.

Note that, SOUP has a bottleneck of performance limited by the original capacity of vanilla DDPG. In our experiments, we find that SOUP with more than ten heads improves the performance slightly than the 10-head with the same hyperparameters, yet more computation cost for interaction. A more reasonable way is to utilize a small number of heads to capture most of the benefits of bootstrapped architecture with fewer parameters and less computation cost.

The consistently better performance highlights the effectiveness and efficiency of double bootstrapped architecture: multiple actor heads generate more action candidates to improve sample efficiency by deep exploration. Multiple critic heads evaluate more justifiable and accurate Q-values for ensemble decisions on high-potential actions to enhance the efficiency of experience replay, which also improves the performance and learning speed. Moreover, in order to counteract the adverse effects caused by critic heads of spotty capacity varying throughout training, self-adaptive confidence adjusts and balances the confidence dynamically for stability.

6 Conclusion

In this paper, we present Self-Adaptive Double Bootstrapped DDPG (SOUP), an algorithm improving both exploration and stability for complex continuous tasks. SOUP extends the actor-critic architecture of DDPG to completely bootstrapped networks for efficient exploration and collaborative decision-

making. Moreover, a self-adaptive confidence mechanism is proposed to dynamically calibrate the weights of the Q-value evaluation from critic heads of spotty and varying capacity, which significantly stabilizes the training.

We demonstrate the effectiveness and performance of SOUP in three experiments on MuJoCo environments by comparisons on bootstrapped models, confidence strategies and multiple heads. We show that SOUP achieves more stable learning with faster speed by at least 45% and improves the performance substantially by efficient exploration.

Acknowledgments

This work is supported by the National High Technology Research and Development Plan (863 Plan) under Grant No. 2015AA015803, the NSFC project under Grant No. U1433112, and the Joint Research Center of Tencent & Tsinghua University.

References

[Brockman *et al.*, 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[Chen *et al.*, 2018] Richard Y. Chen, Szymon Sidor, Pieter Abbeel, and John Schulman. UCB EXPLORATION VIA q-ENSEMBLES, 2018.

[Efron and Tibshirani, 1994] Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

[Gu *et al.*, 2017] Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E. Turner, and Sergey Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. In *Proceedings of International Conference on Learning Representations*, 2017.

[Henderson *et al.*, 2017] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

[Islam *et al.*, 2017] Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017.

[Jaderberg *et al.*, 2017] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *Proceedings of International Conference on Learning Representations*, 2017.

[Kalweit and Boedecker, 2017] Gabriel Kalweit and Joschka Boedecker. Uncertainty-driven imagination for continuous deep reinforcement learning. In *Conference on Robot Learning*, pages 195–206, 2017.

[Kingma and Ba, 2015] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of International Conference on Learning Representations*, 2015.

[Konda and Tsitsiklis, 2000] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[Levine *et al.*, 2016] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[Lillicrap *et al.*, 2016] 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. In *Proceedings of International Conference on Learning Representations*, 2016.

[Maas *et al.*, 2013] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

[Mnih *et al.*, 2015] 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.

[Mnih *et al.*, 2016] 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, 2016.

[Osband *et al.*, 2016] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in Neural Information Processing Systems*, pages 4026–4034, 2016.

[Pathak *et al.*, 2017] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

[Plappert *et al.*, 2018] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y. Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. In *International Conference on Learning Representations*, 2018.

[Silver *et al.*, 2014] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 387–395, 2014.

[Silver *et al.*, 2017] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[Sutton and Barto, 1998] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[Sutton *et al.*, 2000] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

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

[Williams, 1992] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[Yang *et al.*, 2017] Zhaoyang Yang, Kathryn Merrick, Hussein Abbass, and Lianwen Jin. Multi-task deep reinforcement learning for continuous action control. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 3301–3307, 2017.