Integrated Actor-Critic for Deep Reinforcement Learning

Jiaohao Zheng[1], Mehmet Necip Kurt[2(✉)], and Xiaodong Wang[2]

[1] Shenzhen Institute of Advanced Technology, Shenzhen, China
jh.zheng@siat.ac.cn
[2] Electrical Engineering Department, Columbia University, New York, NY, USA
m.n.kurt@columbia.edu, wangx@ee.columbia.edu

Abstract. We propose a new deep deterministic actor-critic algorithm with an integrated network architecture and an integrated objective function. We address stabilization of the learning procedure via a novel adaptive objective that roughly ensures keeping the actor unchanged while the critic makes large errors. We reduce the number of network parameters and propose an improved exploration strategy over bounded action spaces. Moreover, we incorporate some recent advances in deep learning to our algorithm. Experiments illustrate that our algorithm speeds up the learning process and reduces the sample complexity considerably over the state-of-the-art algorithms including TD3, SAC, PPO, and A2C in continuous control tasks.

Keywords: Deep reinforcement learning · Integrated actor-critic · Adaptive objective · Sample complexity

1 Introduction

Reinforcement learning (RL) is effective to learn and control over complex and uncertain environments [25]. Especially with the combination of deep learning, RL has been to shown to perform well in many fields such as robotics, games, automatic control and cybersecurity [2,3,12]. In RL, an agent interacts with an environment with the goal of learning the reward-maximizing policy. Policy-based RL directly optimizes the policy towards higher rewards. Value-based RL learns the value (i.e., expected future reward) of each environment state or state-action pair, and the optimal policy is implicitly determined as the reward-maximizing action at each state. Actor-critic RL is at the intersection of the policy-based and the value-based RL such that the policy (actor) is optimized in the direction suggested by the value function (critic).

In deep RL, actor and critic strongly interact while they are trained simultaneously towards the same objective (i.e., learning the reward-maximizing policy). We aim to use the interdependency between them more explicitly and propose an integrated actor-critic algorithm. In this framework, actor and critic share more

I. Farkaš et al. (Eds.): ICANN 2021, LNCS 12894, pp. 505–518, 2021.
https://doi.org/10.1007/978-3-030-86380-7_41

knowledge, which leads to saving lots of parameters. However, shared parameters also bring an additional challenge on stabilizing the training procedure. The integrated actor-critic can be motivated from animal brains such that although different regions in a brain are assigned to different tasks, all regions are still interconnected, and a brain can act both as actor (i.e., select an action) and critic (i.e., evaluate an action).

Deep RL algorithms suffer from slow learning and high sample complexity. We propose a new model-free off-policy deep deterministic integrated actor-critic algorithm (IAC)[1]. Our algorithm speeds up learning, and equivalently reduces sample complexity of the training procedure, compared to the state-of-the-art deep RL algorithms including the twin delayed deep deterministic policy gradient algorithm (TD3) [4], soft actor-critic algorithm (SAC) [6], proximal policy optimization algorithm (PPO) [21], and advantage actor-critic algorithm (A2C) [15] in continuous control tasks. We first design a novel integrated actor-critic network architecture. Next, we propose a novel adaptive objective function to stabilize the training procedure of the integrated network. Finally, we propose an improved exploration strategy over bounded action spaces and use a set of recent advances in deep learning to further improve the performance and stability of our algorithm.

2 Background

We consider a standard RL problem where an agent interacts with a stochastic environment in order to maximize its expected total reward. We model the problem as a Markov decision process where at each discrete time t, the environment is in a particular state $s_t \in \mathcal{S}$. Assuming a fully observable environment, the agent observes the state s_t, takes an action $a_t \in \mathcal{A}$, and receives a reward $r(s_t, a_t) \in \mathbb{R}$ in return of its action. At the same time, the environment makes a transition to the next state s_{t+1} with the probability $p(s_{t+1}|s_t, a_t)$. This process is repeated until a terminal state is reached. We assume that state and action spaces are continuous and real-valued. In addition, since the feasible action space is usually bounded, we assume $a_{t,k} \in [a_{\min}, a_{\max}]$ where $a_{t,k}$ denotes the kth element of a_t.

Return from a state is defined as the total discounted future reward, $G_t = \sum_{i=t}^{\infty} \gamma^{i-t} r(s_i, a_i)$, where $\gamma \in [0, 1]$ denotes the discount factor. In RL, the agent's goal is to learn an optimal policy $\pi : \mathcal{S} \rightarrow \mathbb{P}(\mathcal{A})$ to maximize its expected return from the start, written by $J^{\pi} = \mathbb{E}_{s_i \sim p^{\pi}, a_i \sim \pi}[G_1]$, where p^{π} denotes the state visitation distribution under the policy π. The agent's policy can either be stochastic or deterministic. In case the policy is stochastic, $\pi(a_t|s_t)$ denotes a probability density function over the action space given the state s_t.

The expected return from a state and action pair is called the Q value. If policy π is followed after taking action a in state s, the Q value is written by $Q^{\pi}(s, a) = \mathbb{E}_{s_{i>t} \sim p^{\pi}, a_{i>t} \sim \pi}[G_t | s_t = s, a_t = a]$. The Bellman equation provides a recursive relationship between the current and the next Q values:

$$Q^{\pi}(s_t, a_t) = r(s_t, a_t) + \gamma \, \mathbb{E}_{s_{t+1} \sim p^{\pi}, a_{t+1} \sim \pi}[Q^{\pi}(s_{t+1}, a_{t+1})].$$

[1] IAC codes are available at https://github.com/IAC-deepRL/IAC.

If the policy is deterministic, it is denoted by $\mu : \mathcal{S} \rightarrow \mathcal{A}$ and the Bellman equation is written by

$$Q^{\mu}(s_t, a_t) = r(s_t, a_t) + \gamma \, \mathbb{E}_{s_{t+1} \sim p^{\mu}}[Q^{\mu}(s_{t+1}, \mu(s_{t+1}))].$$

2.1 Deterministic Policy Gradient

Policy gradient algorithms are useful to solve the RL problems, especially over continuous action domains, in which the policy is parameterized and updated with the policy gradient. Let a deterministic policy $\mu_\theta(s)$ be parameterized with θ and the expected return be written by

$$J(\theta) = \mathbb{E}_{s \sim p^{\mu}}[Q^{\mu}(s, \mu_\theta(s))]. \tag{1}$$

In the deterministic policy gradient algorithm (DPG) [22], the parameters θ are moved towards maximizing $J(\theta)$ via the deterministic policy gradient, given by

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim p^{\mu}} \left[\nabla_\theta \mu_\theta(s) \, \nabla_a Q^{\mu}(s, a)|_{a=\mu_\theta(s)} \right].$$

2.2 Deep Deterministic Policy Gradient

The deep deterministic policy gradient algorithm (DDPG) [13] is a model-free off-policy actor-critic algorithm that combines DPG [22] with the deep Q network algorithm (DQN) [16]. In DDPG, actor and critic are both neural networks. The critic estimates the Q values $Q_w(s, a)$ parameterized by w and the actor learns a deterministic policy $\mu_\theta(s)$. Moreover, separate target actor and target critic networks are kept with parameters θ' and w', respectively, that are slowly updated. These networks provide stable targets to the critic through the Bellman equation: $y_t = r(s_t, a_t) + \gamma Q_{w'}(s_{t+1}, \mu_{\theta'}(s_{t+1}))$. The critic then updates its parameters w to minimize the difference between its Q value estimates and the given targets. Let $\delta_i = y_i - Q_w(s_i, a_i)$. The critic minimizes the following loss function over a mini-batch of samples chosen uniformly from an experience replay buffer $\mathcal{D}$:

$$L(w) = \mathbb{E}_{(s_i, a_i, r(s_i, a_i), s_{i+1}) \sim \mathcal{D}} \left[\delta_i^2 \right], \tag{2}$$

where the replay buffer stores the tuples $(s_i, a_i, r(s_i, a_i), s_{i+1})$ collected during exploration. Moreover, the policy parameters are updated via the sample deterministic policy gradients, given by

$$\nabla_\theta J(\theta) = \mathbb{E}_{\mathcal{D}} \left[\nabla_\theta \mu_\theta(s)|_{s=s_i} \, \nabla_a Q_w(s, a)|_{s=s_i, a=\mu_\theta(s_i)} \right].$$

In DDPG, actor and critic network parameters, θ and w, are disjoint and updated simultaneously in turn. For exploration, actor follows a stochastic behavior policy via additive random noise $\mathcal{N}$ on the deterministic policy: $\mu'(s) = \mu_\theta(s) + \mathcal{N}$.

3 Integrated Actor-Critic

3.1 Network Architecture

The proposed integrated network (see Fig. 1) consists of five main building blocks: state encoder, action encoder, action decoder, Q value decoder, and an internal network connected to all encoders and decoders. The integrated network acts as actor when the green area is activated, and critic when the pink area is activated. The actor and critic share the state encoder and the internal network. The whole network is kept active during training procedure and only the green area (i.e., actor) is activated after training is done.

In the integrated network, each building block is a multilayer neural network (see Fig. 2). The encoder outputs can either be concatenated or added to obtain the internal network's input. According to our experiments, the addition operation works better to reduce the network size and speed up learning without performance loss. In this case, the encoder outputs have the same width, say m.

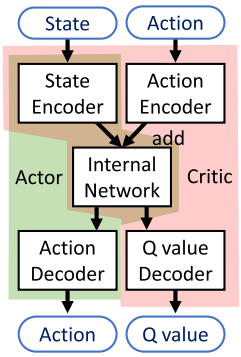

Fig. 1. Integrated actor-critic network. (Color figure online)

We design the internal network by modifying the Dense Convolutional Network (DenseNet) [9] such that all convolutional layers in the DenseNet are replaced with dense (i.e., fully connected) layers. Shortcut connections in the DenseNet architecture enable us training with fewer parameters and improve the learning performance. The internal network takes an input tensor with width m and outputs a tensor of width $4m$, which is input to both decoders.

Thanks to the shared internal network and the state encoder, the integrated network has fewer parameters compared to overall parameters of separate actor and critic networks, especially in high-dimensional tasks (e.g., when video frames form the state input). This can speed up learning. However, the shared parameters also bring an additional challenge on the training stability. The next section addresses this challenge via an adaptive objective function designed for the integrated network.

3.2 Adaptive Objective Function

Let ϕ denote parameters of the integrated network, which is the union of actor and critic parameters: $\phi = \theta \cup w$. In our algorithm, similar to DDPG, we also keep a separate target network with parameters ϕ' to provide stable targets to the critic during training. For convenience, let the policy and the value function be written in terms of ϕ by $\mu_\phi(s)$ and $Q_\phi(s, a)$, respectively. Moreover, let the expected return and the critic's loss be written by $J(\phi)$ (see Eq. (1)) and $L(\phi)$ (see Eq. (2)), respectively, additionally with the following ℓ_1 smoothing [10] on the critic's loss: $L(\phi) = E_{\mathcal{D}}\left[f(\delta_i)\right]$, where

$$f(x) = \begin{cases} 0.5\,x^2, & \text{if } |x| < 1, \\ |x| - 0.5, & \text{if } |x| \geq 1. \end{cases}$$

The ℓ_1 smoothing enables a more stable training, as it provides steady gradients for large δ_i and hence helps to avoid exploding gradients.

We aim to design an objective function to train the parameter-sharing integrated network in a stable manner. In the policy gradient algorithms, the policy cannot be improved if the value function estimation is wrong [4]. Hence, we introduce an adaptive variable $\lambda \in [0, 1]$ that reflects the critic's reliability level. After an initialization, we propose to update λ depending on the critic's loss (over a batch of samples) such that $\lambda \leftarrow \tau e^{-L(\phi)^2} + (1 - \tau)\lambda$, where $\tau \in (0, 1)$ is a hyperparameter. Notice that as the critic's loss $L(\phi)$ gets larger, λ gets closer to 0, and as the critic's loss gets smaller, λ gets closer to 1. A larger λ implies a more reliable critic.

Using the adaptive variable λ, we integrate $J(\phi)$ and $L(\phi)$ as well as an additional regularization term $G(\phi)$ on the policy into the following objective function:

$$Z(\phi) = L(\phi) - \lambda J(\phi) + (1 - \lambda)G(\phi), \tag{3}$$

where $G(\phi) = E_{\mathcal{D}}\left[f(\mu_\phi(s_i) - \mu_{\phi'}(s_i))\right]$ is a measure of how different the policy is from the target policy.

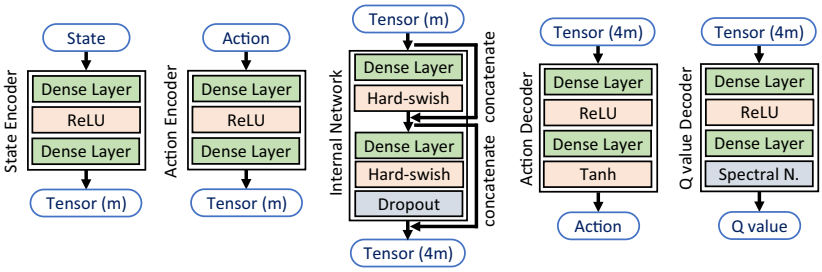

Fig. 2. Building blocks of the integrated network.

The integrated network parameters are updated towards minimizing $Z(\phi)$ via the stochastic gradient descent as follows:

$$\phi \leftarrow \phi - \alpha \nabla_{\phi} Z(\phi) = \phi - \frac{\alpha}{N} \sum_{i=1}^{N} \nabla_{\phi} Z_i(\phi),$$

where α is the learning rate, N is the batch size, and

$$Z_i(\phi) = f(y_i - Q_{\phi}(s_i, a_i)) - \lambda\, Q_{\phi}(s_i, \mu_{\phi}(s_i)) + (1 - \lambda)\, f(\mu_{\phi}(s_i) - \mu_{\phi'}(s_i)).$$

We use the deterministic policy gradient theorem [22] to compute sample policy gradients as in [13].

According to the adaptive objective in Eq. (3), when the critic is less reliable (i.e., smaller λ), the actor gets a smaller learning rate. Specifically, as $\lambda \to 0$, the objective function approximates to

$$Z(\phi) \approx L(\phi) + (1 - \lambda) G(\phi),$$

including only the critic's loss and the regularization term on the actor that roughly ensures keeping the policy unchanged (near the target policy) while the critic makes large errors. In this case, effectively only the critic is updated towards minimizing its loss. On the other hand, when the critic is more reliable (i.e., larger λ), actor gets a larger learning rate such that as $\lambda \to 1$, the objective function approximates to $Z(\phi) \approx L(\phi) - \lambda J(\phi)$ without including the regularization term $G(\phi)$. In this case, actor and critic are updated together.

The two-time-scale update rule (TTUR) [7,11] was shown to be useful for the convergence of the actor-critic algorithms. The TTUR suggests updating the policy with a smaller learning rate and less frequently than the value function. Notice that with the proposed objective function $Z(\phi)$, we update the policy less frequently than the value function, and moreover, we update the policy with a smaller learning rate as $\lambda \leq 1$. Hence, the proposed objective enables an adaptive version of the TTUR.

Finally, depending on the critic's reliability level, we perform adaptive periodic updates on the target network. In particular, we perform hard target updates $\phi' \leftarrow \phi$ at certain periods $p > 1$ only if the critic is sufficiently reliable: $\lambda > \beta$, where $\beta \in (0, 1)$ is a predetermined threshold. This provides an adaptive version of the delayed target updates in TD3 [4].

4 Further Techniques on Improving Performance and Stability

In this section, we first propose an improved exploration strategy and then a modified version of the target policy smoothing technique in TD3. Next, we discuss utility of a set of recent deep learning techniques that have not been commonly used in deep RL.

4.1 Exploration over Bounded Action Spaces

In deep RL, improving exploration is critical to increase data diversity, mitigate overfitting, and speed up learning. Moreover, in off-policy deterministic policy gradient algorithms, exploration can be treated independently from the learning problem [13]. Existing algorithms such as DDPG [13] use random exploration noise $\mathcal{N}_1$ such that $\mu'(s) = \mu_\theta(s) + \mathcal{N}_1$ is the behavior policy. However, since feasible actions are usually bounded to a certain interval such that $a \in [a_{\min}, a_{\max}]$, a clipping operation needs to be employed after noise addition: $\mu'(s) = \min\{\max\{a_{\min}, \mu_\theta(s) + \mathcal{N}_1\}, a_{\max}\}$. We argue that the clipping degrades the exploration efficiency since all actions exceeding the limits are set to the boundary actions, which may then be repeatedly explored by the RL agent.

We address this issue via an easy modification: whenever action exceeds the limits, choose a uniformly random action from the feasible space: $\mu'(s) = g(\mu_\phi(s) + \mathcal{N}_1)$, where

$$g(a) = a\,\mathbb{1}\{a \in [a_{\min}, a_{\max}]\} + \mathcal{U}[a_{\min}, a_{\max}]\,\mathbb{1}\{a \notin [a_{\min}, a_{\max}]\}, \tag{4}$$

$\mathbb{1}\{\cdot\}$ is an indicator function, and $\mathcal{U}[a_{\min}, a_{\max}]$ is a uniform random variable. If the action space is multidimensional, $g(\cdot)$ performs the same elementwise operation at each dimension. We choose the exploration noise $\mathcal{N}_1$ as an independent and identically distributed (iid) zero-mean Gaussian process with variance σ_1^2 at each dimension.

4.2 Target Policy Smoothing

In TD3 [4], target policy smoothing regularization forces similar actions to have similar values and for this purpose, a small random noise is clipped and added on the target policy when computing the target Q values. We find it useful to apply a modified target policy smoothing technique by computing the target y_i for $Q_\phi(s_i, a_i)$ as follows:

$$y_i = r(s_i, a_i) + \frac{\gamma}{2}\big(Q_{\phi'}(s_{i+1}, \mu_{\phi'}(s_{i+1})) + Q_{\phi'}(s_{i+1}, g(\mu_{\phi'}(s_{i+1}) + \mathcal{N}_2))\big),$$

where $g(\cdot)$ is as given in Eq. (4) and the smoothing noise $\mathcal{N}_2$ is chosen as an iid zero-mean Gaussian process with variance σ_2^2 at each action dimension.

4.3 Spectral Normalization

Since the generative adversarial networks (GANs) [5] and the actor-critic RL [11] are both bi-level optimization problems, where one model is optimized with respect to the optimum of another model, and there are many similarities in their information structures [17], techniques for improving the stability of GANs are potentially useful to stabilize the actor-critic algorithms as well. In GANs, the spectral normalization, that normalizes the spectral norm of the weight matrices of the discriminator network, is shown to improve the stability of training the

discriminator [14]. Since the discriminator in GANs corresponds to the critic in actor-critic RL [17], we employ the spectral normalization on the critic network, particularly on the Q value decoder (see Fig. 2), with the goal of improving the stability of our algorithm.

4.4 Hard-Swish

In neural networks, nonlinear activation functions enable learning complex mappings from inputs to outputs, which is useful to deal with complex and high-dimensional data. Hard-swish [8] is a computationally simplified version of the swish nonlinearity [18] and it achieves a good performance especially in deep neural networks [8]. In our network design, we use the hard-swish as the activation function in the internal network and the decoders (see Fig. 2). Moreover, we use the rectified linear unit (ReLU) activation in the encoders and the hyperbolic tangent (Tanh) activation at the output layer of the action decoder.

4.5 Dropout

Dropout [24] is randomly zeroing out a certain fraction of neurons at a layer, that reduces the network capacity and forces the network to learn the most important patterns in the data. It is a widely used technique to mitigate overfitting in deep learning. We use the dropout at the last layer of the internal network (see Fig. 2) with the purpose of reducing the overfitting and improving the generalization ability of the network.

4.6 Adjusting Batch Size and Number of Iterations During Training

In [23], it is shown that increasing the batch size enables training a model with fewer parameter updates compared to reducing the learning rate in the stochastic gradient descent optimization. Based on this principle, we increase the batch size and the number of iterations during training as new samples are collected and stored in the experience replay buffer, until the buffer is full.

In our algorithm (see Algorithm 1), at each training episode, first the actor interacts with the environment, collects new samples, and saves them into the buffer. Next, the network parameters are updated via the stochastic gradient descent with a mini-batch of samples chosen uniformly from the buffer. In this process, let the parameters be updated over K iterations and the batch size be N. Moreover, let $K_0 \geq 1$ and $N_0 \geq 1$ be the initial number of iterations and the initial batch size, respectively. Furthermore, let the buffer capacity be $M \gg 1$ and the current size of the buffer be $0 \leq R \leq M$. We keep and update a parameter ρ while the buffer size gradually increases as more samples are collected: $\rho = 1 + R/M$. We then update the number of iterations and the batch size as $K = \rho K_0$ and $N = \rho N_0$, respectively.

Algorithm 1. Integrated Actor-Critic (IAC)

Initialize the integrated network with random parameters ϕ and the target network with $\phi' \leftarrow \phi$
Initialize the replay buffer $\mathcal{D}$ with size $R \leftarrow 0$ and the adaptive variable with $\lambda \leftarrow 0.5$
for episode $= 1 : E$ **do**
    I. Interact with environment
    Observe the initial state s_1
    for $t = 1 : T$ **do**
        Select action $a_t \leftarrow g(\mu_\phi(s_t) + \mathcal{N}_1)$, receive reward $r(s_t, a_t)$, and observe the next state s_{t+1}
        Save the tuple $(s_t, a_t, r(s_t, a_t), s_{t+1})$ into $\mathcal{D}$ and update the buffer size: $R \leftarrow \min\{R + 1, M\}$
    II. Update network parameters
    Update the number of iterations and the batch size: $\rho \leftarrow 1 + R/M$, $K \leftarrow \rho K_0$, $N \leftarrow \rho N_0$
    Initialize the total loss of critic: $L \leftarrow 0$
    for $k = 1 : K$ **do**
        Sample uniformly a mini-batch of N tuples $(s_i, a_i, r(s_i, a_i), s_{i+1})$ from $\mathcal{D}$
        $y_i \leftarrow r(s_i, a_i) + \frac{\gamma}{2}\left(Q_{\phi'}(s_{i+1}, \mu_{\phi'}(s_{i+1})) + Q_{\phi'}(s_{i+1}, g(\mu_{\phi'}(s_{i+1}) + \mathcal{N}_2))\right)$
        Update the integrated network: $\phi \leftarrow \phi - \frac{\alpha}{N}\sum_{i=1}^{N} \nabla_\phi Z_i(\phi)$
        Update the total loss of critic: $L \leftarrow L + \sum_{i=1}^{N} f(y_i - Q_\phi(s_i, a_i))$
        if k mod p **then**
            Compute the average loss of critic: $\bar{L} \leftarrow L/pN$ and reset the total loss: $L \leftarrow 0$
            Update the adaptive variable: $\lambda \leftarrow \tau e^{-\bar{L}^2} + (1 - \tau)\lambda$
            if $\lambda > \beta$ **then**
                Update the target network: $\phi' \leftarrow \phi$

5 Experiments

5.1 Comparisons with Benchmark Algorithms

We evaluate IAC (see Algorithm 1) over five continuous control tasks in the OpenAI Gym [1]. In all the tasks, feasible actions are limited to $[-1, 1]$. For comparisons, we use TD3 [4], SAC [6], A2C [15] (both with separate and shared actor-critic networks), and PPO [21] algorithms. Figure 3 illustrates the learning curves of all algorithms. We measure the sample complexity (equivalently, learning speed) of each algorithm until reaching the default target reward set at each environment. Figure 4 illustrates the average training steps until achieving the target rewards. Note that in Fig. 4, we do not present the bar charts for algorithms that could not reach the target rewards within a reasonable training period. We obtain both the learning curves and the bar charts by averaging the results over 50 random seeds. The experiments show that IAC outperforms all the benchmark algorithms in terms of learning speed.

5.2 Self-comparisons

We evaluate contributions of various IAC components on the overall algorithm performance. We specify four algorithm levels such that new components are added at each level and the level 4 corresponds to the full algorithm (see Table 1). Figure 5 illustrates the average training steps until achieving the target rewards (over 50 random seeds) for all IAC levels and TD3 as a benchmark. Figure 5 shows that the learning speed progressively improves from level 1 to level 4, which implies that all IAC components are useful. Note that in the level 1, conventional clipping operation is performed for the exploration, different from the level 2. Furthermore, the ReLU activation is used instead of hard-swish at all levels except for the level 4. Notice that the level 1 and level 2 include the novel components of IAC whereas the level 3 and level 4 incorporates some existing

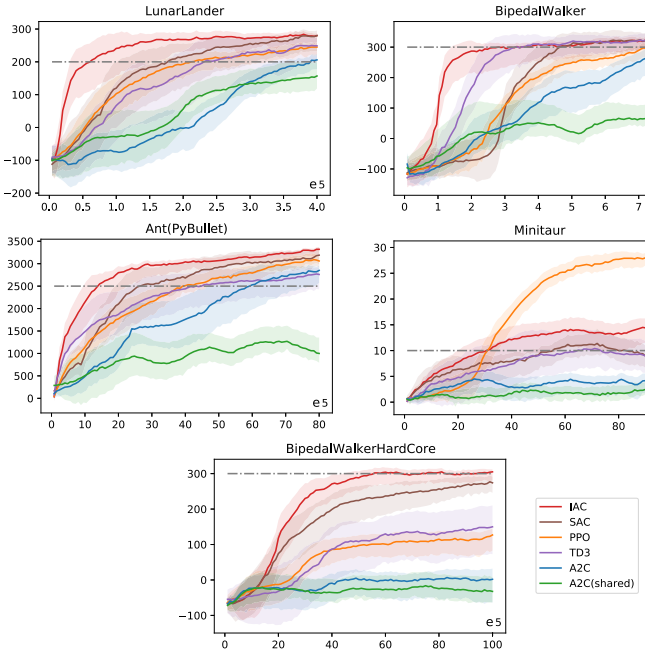

Fig. 3. Learning curves: expected return vs. number of training steps.

Table 1. Levels of IAC for self-comparisons

Level	Description
L1	Integrated network + Adaptive objective
L2	L1 + Modified exploration strategy
L3	L2 + Target policy smoothing + Spectral normalization
L4	L3 + Hard-swish + Dropout + Adjusting batch size and iteration number

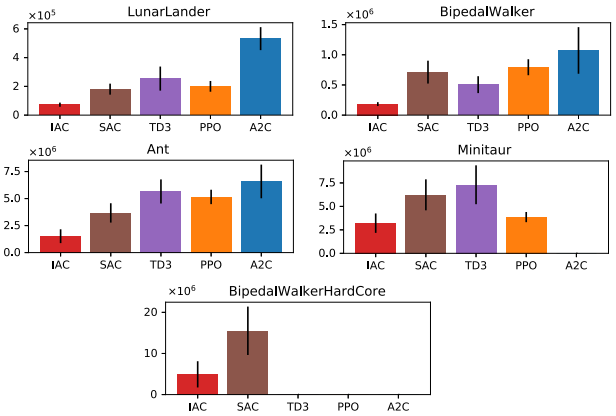

Fig. 4. Comparisons with benchmark algorithms. The bar charts illustrate the mean and standard deviation of the number of training steps until achieving the target rewards.

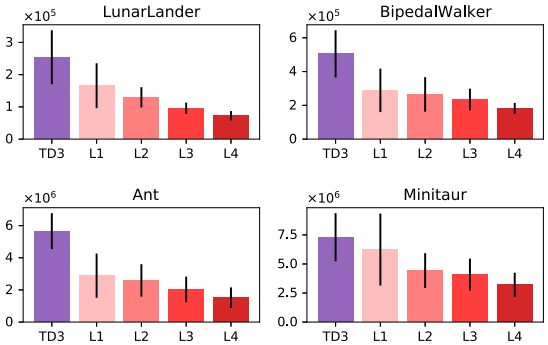

Fig. 5. Self-comparisons. The bar charts illustrate the mean and standard deviation of the number of training steps until achieving the target rewards.

deep learning techniques. Figure 5 shows that only with the novel components, IAC still significantly outperforms TD3.

6 Related Work

Parameter sharing is used for multi-task learning in neural networks [19]. In deep RL, sharing parameters between actor and critic have been discussed for A3C/A2C [15] and PPO [21] (although the PPO implementation does not share parameters) such that the network is shared except for the output layers and the objective function directly adds the actor's and critic's objectives, possibly with an additional entropy term to enable sufficient exploration. In the shared versions of both A3C/A2C and PPO, actor and critic have the same state input. In IAC, we propose a new shared network architecture where actor and critic

have different inputs. Moreover, the adaptive objective of IAC is specifically designed for stable training of the integrated actor-critic network.

Policy gradient algorithms have high sample complexity [6,13]. Deterministic policy gradient algorithms scale better over high-dimensional action spaces since the deterministic policy gradient integrates only over the state space whereas the stochastic policy gradient integrates over both state and action spaces [22]. Moreover, off-policy learning with experience replay enables reusing the past experience and reduces the sample complexity [13]. IAC further reduces the sample complexity via reducing the number of parameters, improving the exploration strategy, and using some recent advances in deep learning, namely the dropout [24], DenseNet [9], and increasing the batch size during training [23].

TD3 [4] addresses the overestimation of the value function via the clipped double Q learning, delayed updates on the policy and target networks, and the target policy smoothing regularization. Our objective function enables an adaptive version of the delayed policy updates in addition to TTUR [7,11] (see Sect. 3.2). Moreover, we delay updates on the target network adaptively depending on the critic's loss (see Algorithm 1). Further, we use a modified target policy smoothing technique for performance improvement (see Sect. 4.2).

In RL, reward-maximizing actions can be learned more quickly with a better exploration strategy. SAC [6], PPO [21], and the trust region policy optimization algorithm (TRPO) [20] use entropy regularization to encourage more exploration. In A3C/A2C [15], multiple actors explore environment in parallel, each with a possibly different behavior policy for better exploration. In deterministic policy gradient algorithms, a stochastic behavior policy is used to ensure sufficient exploration [4,13]. In IAC, we improve exploration over bounded action spaces (see Sect. 4.1).

Strong connections between the actor-critic RL and the GANs have been discussed in [17]. The implication is that techniques used to stabilize and improve one of them can be useful for the other. In IAC, we use the spectral normalization that was shown to stabilize the training of GANs [14].

7 Conclusions

We have proposed an off-policy deep deterministic integrated actor-critic algorithm (IAC) based on a shared network architecture and an adaptive objective function. Sharing the network between actor and critic reduces the overall number of parameters but brings an additional challenge on the training stability. The adaptive objective enables a stable training via keeping the policy unchanged while the value estimation is wrong. We have presented an improved exploration strategy over bounded action spaces. Moreover, we have incorporated some recent advances in deep learning, namely the DenseNet, spectral normalization, target policy smoothing, dropout, hard-swish activation, and adjustment of the batch size and iteration number during training, to further improve our algorithm. The experiments have shown that IAC speeds up learning and reduces the sample complexity significantly over the state-of-the-art deep RL algorithms.

References

1. OpenAI Gym (2021). https://gym.openai.com/
2. Church, A., Lloyd, J., Hadsell, R., Lepora, N.F.: Deep reinforcement learning for tactile robotics: learning to type on a braille keyboard. IEEE Rob. Autom. Lett. **5**(4), 6145–6152 (2020)
3. François-Lavet, V., Henderson, P., Islam, R., Bellemare, M.G., Pineau, J.: An introduction to deep reinforcement learning. Found. Trends®in Mach. Learn. **11**(3–4), 219–354 (2018)
4. Fujimoto, S., van Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods. arXiv preprint arXiv:1802.09477 (2018)
5. Goodfellow, I., et al.: Generative adversarial nets. In: Advances in Neural Information Processing Systems, pp. 2672–2680 (2014)
6. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. arXiv preprint arXiv:1801.01290 (2018)
7. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In: Advances in Neural Information Processing Systems, pp. 6626–6637 (2017)
8. Howard, A., et al.: Searching for mobilenetv3. arXiv preprint arXiv:1905.02244 (2019)
9. Huang, G., Liu, Z., van der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017
10. Huber, P.J.: Robust estimation of a location parameter. In: Breakthroughs in Statistics, pp. 492–518. Springer (1992). https://doi.org/10.1007/978-1-4612-4380-9_35
11. Konda, V.R., Tsitsiklis, J.N.: On actor-critic algorithms. SIAM J. Control. Optim. **42**(4), 1143–1166 (2003)
12. Kurt, M.N., Ogundijo, O., Li, C., Wang, X.: Online cyber-attack detection in smart grid: a reinforcement learning approach. IEEE Trans. Smart Grid **10**(5), 5174–5185 (2019)
13. Lillicrap, T.P., et al.: Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971 (2015)
14. Miyato, T., Kataoka, T., Koyama, M., Yoshida, Y.: Spectral normalization for generative adversarial networks. arXiv preprint arXiv:1802.05957 (2018)
15. Mnih, V., et al.: Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning, pp. 1928–1937 (2016)
16. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
17. Pfau, D., Vinyals, O.: Connecting generative adversarial networks and actor-critic methods. arXiv preprint arXiv:1610.01945 (2016)
18. Ramachandran, B.Z.P., Le, Q.V.: Searching for activation functions. arXiv preprint arXiv:1710.05941 (2017)
19. Ruder, S.: An overview of multi-task learning in deep neural networks. arXiv preprint arXiv:1706.05098 (2017)
20. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: International Conference on Machine Learning, pp. 1889–1897 (2015)
21. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)

22. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M.: Deterministic policy gradient algorithms. In: Proceedings of The 31st International Conference on Machine Learning, pp. 387–395 (2014)
23. Smith, S.L., Kindermans, P.J., Le, Q.V.: Don't decay the learning rate, increase the batch size. In: International Conference on Learning Representations (2018)
24. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
25. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)