Self-Guided and Self-Regularized Actor-Critic

Anonymous Author(s)
Affiliation
Address
email

Abstract

Deep reinforcement learning (DRL) algorithms have successfully been demonstrated on a range of challenging decision making and control tasks. One dominant component of recent deep reinforcement learning algorithms is the target network which mitigates the divergence when learning the Q function. However, target networks can slow down the learning process due to delayed function updates. Another dominant component especially in continuous domains is the policy gradient method which models and optimizes the policy directly. However, when Q functions are approximated with neural networks, their landscapes can be complex and therefore mislead the local gradient. In this work, we propose a self-regularized and self-guided actor-critic method. We introduce a self-regularization term within the TD-error minimization and remove the need for the target network. In addition, we propose a self-guided policy improvement method by combining policy-gradient with zero-order optimization such as the Cross Entropy Method. It helps to search for actions associated with higher Q-values in a broad neighborhood and is robust to local noise in the Q function approximation. These actions help to guide the updates of our actor network. We evaluate our method on the suite of OpenAI gym tasks, achieving or outperforming state of the art in every environment tested.

1 Introduction

Reinforcement learning (RL) studies decision-making with the goal of maximizing total discounted reward when interacting with an environment. Leveraging high-capacity function approximators such as neural networks, Deep reinforcement learning (DRL) algorithms have been successfully applied to a range of challenging domains, from video games [12] to robotic control [16].

Actor-critic algorithms are among the most popular approaches in DRL, e.g. *DDPG* [11], *TRPO* [16], *TD3* [6] and *SAC* [7]. These methods are based on policy iteration, which alternates between policy evaluation and policy improvement [17]. Actor-critic methods jointly optimize the value function (critic) and the policy (actor) as it is often impractical to run either of these to convergence [7].

In DRL, both the actor and critic use deep neural networks as the function approximator. However, DRL is known to assign unrealistically high values to state-action pairs represented by the Q-function. This is detrimental to the quality of the greedy control policy derived from Q [21]. Mnih *et al.* [13] proposed to use a *target network* to mitigate divergence. A target network is a copy of the current Q function that is held fixed to serve as a stable target within the TD error update. The parameters of the target network are either infrequently copied [13] or obtained by Polyak averaging [11]. A limitation of using a target network is that it can slow down learning due to delayed function updates. We propose an approach that reduces the need for a target network in DRL while still ensuring stable learning and good performance in high-dimensional domains. We add a self-regularization term to encourage small changes to the target value while minimizing the Temporal Difference (TD)-error [17].

Evolution Strategies (ES) are a family of black-box optimization algorithms which are typically very stable, but scale poorly in high-dimensional search spaces, (e.g. neural networks) [14]. Gradient-

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

based DRL methods are often sample efficient, particularly in the off-policy setting when, unlike evolutionary search methods, they can continue to sample previous experiences to improve value estimation. But these approaches can also be unstable and highly sensitive to hyper-parameter tuning [14]. We propose a novel policy improvement method which combines both approaches to get the best of both worlds. Specifically, after the actor network first outputs an initial action, we apply the *Cross Entropy Method* (CEM) [15] to search the neighborhood of the initial action to find a second action associated with a higher Q value. Then we leverage the second action in the policy improvement stage to speed up the learning process.

To mitigate the overestimation issue in Q learning [18], Fujimoto *et al.* [6] proposed Clipped Double Q-Learning in which the authors learn two Q-functions and use the smaller one to form the targets in the TD-Learning process. This method may suffer from under-estimation. In practice, we also observe that the discrepancy between the two Q-functions can increase dramatically which hinders the learning process. We propose Max-min Double Q-Learning to address this discrepancy. Our method also provides a better approximation of the Bellman optimality operator [17].

We propose a novel self-**G**uided and self-**R**egularized **A**ctor **C**ritic (*GRAC*) algorithm. *GRAC* uses **self-regularized TD-Learning** removing the need for a target network and utilizes a novel **policy improvement method which combines policy-gradients and zero-order optimization** to speed up learning. Following Clipped Double Q-Learning, we propose **Max-min Double Q-learning** to address underestimation and the discrepancy between the two Q functions. We evaluate *GRAC* on six continuous control domains from OpenAI gym [3], where we achieve or outperform state of the art result in every environment tested. We run our experiments across a large number of seeds with fair evaluation metrics [4], perform extensive ablation studies, and open source both our code and learning curves.

2 Related Work

The proposed algorithm incorporates three key ingredients within the actor-critic method: a self-regularized TD update, self-guided policy improvements based on evolution strategies, and Max-min double Q-Learning. In this section, we review prior work related to these ideas.

Divergence in Deep Q-Learning In Deep Q-Learning, we use a nonlinear function approximator such as a neural network to approximate the Q-function that represents the value of each state-action pair. Learning the Q-function in this way is known to suffer from divergence issues [20] such as assigning unrealistically high values to state-action pairs [21]. This is detrimental to the quality of the greedy control policy derived from Q [21]. To mitigate the divergence issue, Mnih *et al.*[13] introduce a target network which is a copy of the estimated Q-function and is held fixed to serve as a stable target for some number of steps. However, target networks can slow down the learning process due to delayed function updates [10]. Durugkar *et al.* [5] propose Constrained Q-Learning, which uses a constraint to prevent the average target value from changing after an update. Achiam *et al.*[1] give a simple analysis based on a linear approximation of the Q function and develop a stable Deep Q-Learning algorithm for continuous control without target networks. However, their proposed method requires separately calculating backward passes for each state-action pair in the batch, and solving a system of equations at each timestep. The proposed *GRAC* algorithm adds a self-regularization term to the TD-Learning objective to keep the change of the state-action value small.

Evolution Strategies in Deep Reinforcement Learning *Evolution Strategies* (ES) are a family of black-box optimization algorithms which are typically very stable, but scale poorly in high-dimensional search spaces [23]. Gradient-based deep RL methods, such as *DDPG* [11], are often sample efficient, particularly in the off-policy setting. These off-policy methods can continue to reuse previous experience to improve value estimations but can be unstable and highly sensitive to hyper-parameter tuning [14]. Researchers have proposed to combine these approaches to get the best of both worlds. Pourchot *et al.* [14] proposed *CEM-RL* to combine CEM with either *DDPG* [11] or *TD3* [6]. However, *CEM-RL* applies *CEM* within the actor parameter space which is extremely high-dimensional, making the search not efficient. Kalashnikov *et al.* [9] introduce *QT-Opt*, which leverages *CEM* to search the landscape of the Q function, and enables Q-Learning in continuous action spaces without using an actor. However, as shown in [23], *CEM* does not scale well to high-dimensional action spaces, such as in the Humanoid task we used in this paper. A near-optimal actor is needed to initialize the *CEM* process in such tasks. Different from *QT-Opt*, we adopt the actor-critic framework and leverage *CEM* in both Q-Learning and policy improvement. *GRAC* speeds up the learning process compared to popular actor-critic methods.

Double-Q Learning Using function approximation, Q-learning [22] is known to suffer from overestimation [18]. To mitigate this problem, Hasselt *et al.* [8] proposed *Double Q-learning* which uses two Q functions with independent sets of weights. *TD3* [6] proposed *Clipped Double Q-learning* to learn two Q-functions and uses the smaller of the two to form the targets in the TD-Learning process. However, *TD3* [6] may lead to underestimation. Besides, the actor network in *TD3* [6] is trained to select the action to maximize the first Q function throughout the training process which may make it very different from the second Q-function. A large discrepancy results in large TD-errors which in turn results in large gradients during the update of the actor and critic networks. This makes instability of the learning process more likely. We propose *Max-min Double Q-Learning* to balance the differences between the two Q functions and provide a better approximation of the Bellman optimality operator [17].

3 Preliminaries

In this section, we define the notation used in subsequent sections. Consider a *Markov Decision Process* (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathscr{P}, r, \rho_0, \gamma)$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\mathscr{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the transition probability distribution, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\rho_0 : \mathcal{S} \to \mathbb{R}$ is the distribution of the initial state s_0, and $\gamma \in (0, 1)$ is the discount factor. At each discrete time step t, with a given state $s_t \in \mathcal{S}$, the agent selects an action $a_t \in \mathcal{A}$, receiving a reward r and the new state s_{t+1} of the environment.

Let π denote the policy which maps a state to a probability distribution over the actions, $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$. The return from a state is defined as the sum of discounted reward $R_t = \sum_{i=t} \gamma^{i-t} r(s_i, a_i)$. In reinforcement learning, the objective is to find the optimal policy π^*, with parameters ϕ, which maximizes the expected return $J(\phi) = \sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi(s_t,a_t)}[\gamma^t r(s_t, a_t)]$ where $\rho_\pi(s_t)$ and $\rho_\pi(s_t, a_t)$ denote the state and state-action marginals of the trajectory distribution induced by the policy $\pi(a_t|s_t)$.

We use the following standard definitions of the state-action value function Q_π. It describes the expected discounted reward after taking an action a_t in state s_t and thereafter following policy π:

$$Q_\pi(s_t, a_t) = \mathbb{E}_\pi[R_t|s_t, a_t]. \tag{1}$$

In this work we use *CEM* to find optimal actions with maximum Q values. *CEM* is a randomized zero-order optimization algorithm. To find the action a that maximizes $Q(s, a)$, *CEM* is initialized with a paramaterized distribution over a, $P(a; \psi)$. Then it iterates between the following two steps [2]: First generate $a_1, \ldots, a_N \sim P(s; \psi)$. Retrieve their Q values $Q(s, a_i)$ and sort the actions to have decreasing Q values. Then keep the first K actions, and solve for an updated parameters ψ':

$$\psi' = \text{argmax}_\psi \frac{1}{K} \sum_{i=1}^{K} \log(P(a_i; \psi))$$

In the following sections, we denote *CEM*$(Q(s, \cdot), \pi(\cdot|s))$ as the action found by *CEM* to maximize $Q(s, \cdot)$, when *CEM* is intiailized by the distribution predicted by the policy.

4 Technical Approach

4.1 Self-Regularized TD Learning

Reinforcement learning is prone to instability and divergence when a nonlinear function approximator such as a neural network is used to represent the Q function [20]. Mnih *et al.*[13] identified several reasons for this. One is the correlation between the current action-values and the target value. Updates to $Q(s_t, a_t)$ often also increase $Q(s_{t+1}, a^*_{t+1})$ where a^*_{t+1} is the optimal next action. Hence, these updates also increase the target value y_t which may lead to oscillations or the divergence of the policy.

More formally, given transitions (s_t, a_t, r_t, s_{t+1}) sampled from the replay buffer distribution $\mathcal{B}$, the Q network can be trained by minimising the loss functions $\mathcal{L}(\theta_i)$ at iteration i:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{B}} \|(Q(s_t, a_t; \theta_i) - y_i)\|^2 \tag{2}$$

Algorithm 1 GRAC

Initialize critic network $Q_{\theta 1}$, $Q_{\theta 2}$ and actor network π_ϕ with random parameters $\theta 1$, $\theta 2$ and ϕ
Initialize replay buffer $\mathcal{B}$, Set $\alpha < 1$

for $i = 1, \ldots$ **do**
  Select action $a \sim \pi_{\phi_i}(s)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$
  Sample mini-batch of N transitions (s_t, a_t, r_t, s_{t+1}) from $\mathcal{B}$
  $\hat{a}_{t+1} \sim \pi_{\phi_i}(s_{t+1})$
  $\tilde{a}_{t+1} \leftarrow \textit{CEM}(Q(s_{t+1}, \cdot; \theta_2), \pi_{\phi_i}(\cdot|s_{t+1}))$
  $y \leftarrow r_t + \gamma \max\{\min_{j=1,2} Q(s_{t+1}, \tilde{a}_{t+1}; \theta_j), \min_{j=1,2} Q(s_{t+1}, \hat{a}_{t+1}; \theta_j)\}$
  $a^\dagger \leftarrow \arg\max_{\{\tilde{a}, \hat{a}\}}\{\min_{j=1,2} Q(s_{t+1}, \tilde{a}_{t+1}; \theta_j), \min_{j=1,2} Q(s_{t+1}, \hat{a}_{t+1}; \theta_j)\}$
  $y_1', y_2' \leftarrow Q(s_{t+1}, a^\dagger; \theta_1), Q(s_{t+1}, a^\dagger; \theta_2)$
  for $k = 1$ to K **do**
    $\mathcal{L}_k = \|y - Q(s_t, a_t; \theta_1)\|_2 + \|y - Q(s_t, a_t; \theta_2)\|_2 + \|y_1' - Q(s_{t+1}, a^\dagger; \theta_1)\|_2 + \|y_2' - Q(s_{t+1}, a^\dagger; \theta_2)\|_2$
    $\theta 1 \leftarrow \theta 1 - \lambda \nabla_{\theta 1} \mathcal{L}_k$, $\theta 2 \leftarrow \theta 2 - \lambda \nabla_{\theta 2} \mathcal{L}_k$
    if k > 1 and $\mathcal{L}_k < \alpha \mathcal{L}_2$ **then**
      Break
    end if
  end for
  $\hat{a}_t \sim \pi_{\phi_i}(s_t)$
  $J_\pi(\phi) = \mathbb{E}_{(s_t, \hat{a}_t)}[Q(s_t, \hat{a}_t; \theta_1)]$
  $\bar{a}_t \leftarrow$ CEM$(Q(s_t, \cdot; \theta_1), \pi_{\phi_i}(\cdot|s_t))$
  $\phi \leftarrow \phi - \lambda \nabla_\phi J_\pi(\phi) - \lambda\, \mathbb{E}_{(s_t, \hat{a}_t)}[Q(s_t, \bar{a}_t; \theta_1) - Q(s_t, \hat{a}_t; \theta_1)]_+ \nabla_\phi \log \pi(\bar{a}_t | s_t; \phi)$
end for

where for now let us assume $y_i = r_t + \gamma \max_a Q(s_{t+1}, a; \theta_i)$ to be the target for iteration i computed based on the current Q network parameters θ_i. $a^*_{t+1} = \arg\max_a Q(s_{t+1}, a)$. If we update the parameter θ_{i+1} to reduce the loss $\mathcal{L}(\theta_i)$, it changes both $Q(s_t, a_t; \theta_{i+1})$ and $Q(s_{t+1}, a^*_{t+1}; \theta_{i+1})$. Assuming an increase in both values, then the new target value $y_{i+1} = r_t + \gamma Q(s_{t+1}, a^*_{t+1}; \theta_{i+1})$ for the next iteration will also increase leading to an explosion of the Q function. We demonstrated this behavior in an ablation experiment with results in Fig. 2. We also show how maintaining a separate target network [13] with frozen parameters θ^- to compute $y_{i+1} = r_t + \gamma Q(s_{t+1}, a^*_{t+1}; \theta^-)$ delays the update of the target and therefore leads to more stable learning of the Q function. However, delaying the function updates also comes with the price of slowing down the learning process.

We propose a self-Regularized TD-learning approach to minimize the TD-error while also keeping the change of $Q(s_{t+1}, a^*_{t+1})$ small. This regularization mitigates the divergence issue [20], and no longer requires a target network that would otherwise slow down the learning process. Let $y_i' = Q(s_{t+1}, a^*_{t+1}; \theta_i)$, and $y_i = r_t + \gamma y_i'$. We define the learning objective as

$$\min_\theta \|Q(s_t, a_t; \theta) - y_i)\|^2 + \|Q(s_{t+1}, a^*_{t+1}; \theta)) - y_i'\|^2 \tag{3}$$

where the first term is the original TD-Learning objective and the second term is the regularization term penalizing large updates to $Q(s_{t+1}, a^*_{t+1})$. Note that when the current Q network updates its parameters θ, both $Q(s_t, a_t)$ and $Q(s_{t+1}, a^*_{t+1})$ change. Hence, the target value y_i will also change which is different from the approach of keeping a frozen target network for a few iterations. We will demonstrate in our experiments that this self-regularized TD-Learning approach removes the delays in the update of the target value thereby achieves faster and stable learning.

4.2 Self-Guided Policy Improvement with Evolution Strategies

The policy, known as the actor, can be updated through a combination of two parts. The first part, which we call Q-loss policy update, improves the policy through local gradients of the current Q function, while the second part, which we call *CEM* policy update, finds a high-value action via *CEM* in a broader neighborhood of the Q function landscape, and update the action distribution to concentrate towards this high-value action. We describe the two parts formally below.

Given states s_t sampled from the replay buffer, the Q-loss policy update maximizes the objective

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{B}, \hat{a}_t \sim \pi}[Q(s_t, \hat{a}_t)], \tag{4}$$

where $\hat{a}_t$ is sampled from the current policy $\pi(\cdot|s_t)$. The gradient is taken through the reparameterization trick. We reparameterize the policy using a neural network transformation as described in Haarnoja *et al.* [7],

$$\hat{a}_t = f_\phi(\epsilon_t|s_t) \tag{5}$$

where ϵ_t is an input noise vector, sampled from a fixed distribution, such as a standard multivariate Normal distribution. Then the gradient of $J_\pi(\phi)$ is:

$$\nabla J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{B}, \epsilon_t \sim \mathcal{N}}[\frac{\partial Q(s_t, f_\phi(\epsilon_t|s_t))}{\partial f}\frac{\partial f_\phi(\epsilon_t|s_t)}{\partial \phi}] \tag{6}$$

For the CEM policy update, given a minibatch of states s_t, we first find a high-value action $\bar{a}_t$ for each state by running *CEM* on the current Q function, $\bar{a}_t = \textit{CEM}(Q(s_t, \cdot), \pi(\cdot|s_t))$. Then the policy is updated to increase the probability of this high-value action. The guided update on the parameter ϕ of π at iteration i is

$$\mathbb{E}_{s_t \sim \mathcal{B}, \hat{a}_t \sim \pi}[Q(s_t, \bar{a}_t) - Q(s_t, \hat{a}_t)]_+ \nabla_\phi \log \pi_i(\bar{a}_t|s_t). \tag{7}$$

We used $Q(s_t, \hat{a}_t)$ as a baseline term, since its expectation over actions $\hat{a}_t$ will give us the normal baseline $V(s_t)$:

$$\mathbb{E}_{s_t \sim \mathcal{B}}[Q(s_t, \bar{a}_t) - V(s_t)] \nabla_\phi \log \pi_i(\bar{a}_t|s_t) \tag{8}$$

In our implementation, we only perform an update if the improvement on the Q function, $Q(s_t, \bar{a}_t) - Q(s_t, \hat{a}_t)$, is non-negative, to guard against the occasional cases where *CEM* fails to find a better action.

Combining both parts of updates, the final update rule on the parameter ϕ_i of policy π_i is

$$\phi_{i+1} = \phi_i - \lambda \nabla_\phi J_{\pi_i}(\phi_i) - \lambda\, \mathbb{E}_{s_t \sim \mathcal{B}, \hat{a}_t \sim \pi_i}[Q(s_t, \bar{a}_t) - Q(s_t, \hat{a}_t)]_+ \nabla_\phi \log \pi_i(\bar{a}_t|s_t)$$

where λ is the step size.

We can prove that if the Q function has converged to Q^π, the state-action value function induced by the current policy, then both the Q-loss policy update and the *CEM* policy update will be guaranteed to improve the current policy. We formalize this result in Theorem 1 and Theorem 2, and prove them in Appendix 3.1 and 3.2.

Theorem 1 ***Q-loss Policy Improvement*** *Starting from the current policy π, we maximize the objective $J_\pi = \mathbb{E}_{(s,a) \sim \rho_\pi(s,a)} Q^\pi(s,a)$. The maximization converges to a critical point denoted as π_{new}. Then the induced Q function, $Q^{\pi_{new}}$, satisfies $\forall(s,a), Q^{\pi_{new}}(s,a) \geq Q^\pi(s,a)$.*

Theorem 2 **CEM** ***Policy Improvement*** *Assuming the* CEM *process is able to find the optimal action of the state-action value function, $a^*(s) = \arg\max_a Q^\pi(s,a)$, where Q^π is the Q function induced by the current policy π. By iteratively applying the update $\mathbb{E}_{(s,a) \sim \rho_\pi(s,a)}[Q(s,a^*) - Q(s,a)]_+ \nabla \log \pi(a^*|s)$, the policy converges to π_{new}. Then $Q^{\pi_{new}}$ satisfies $\forall(s,a), Q^{\pi_{new}}(s,a) \geq Q^\pi(s,a)$.*

4.3 Max-min Double Q-Learning

Q-learning [22] is known to suffer from overestimation [18]. Hasselt *et al.* [8] proposed Double-Q learning which uses two Q functions with independent sets of weights to mitigate the overestimation problem. Fujimoto *et al.* [6] proposed Clipped Double Q-learning with two Q function denoted as $Q(s,a;\theta_1)$ and $Q(s,a;\theta_2)$, or Q_1 and Q_2 in short. Given a transition (s_t, a_t, r_t, s_{t+1}), Clipped Double Q-learning uses the minimum between the two estimates of the Q functions when calculating the target value in TD-error [17]:

$$y = r_t + \gamma \min_{j=1,2} Q(s_{t+1}, \hat{a}_{t+1}; \theta_j) \tag{9}$$

where $\hat{a}_{t+1}$ is the predicted next action.

Fujimoto *et al.* [6] mentioned that such an update rule may induce an underestimation bias. In addition, $\hat{a}_{t+1} = \pi_\phi(s_{t+1})$ is the prediction of the actor network. The actor network's parameter ϕ is optimized according to the gradients of Q_1. In other words, $\hat{a}_{t+1}$ tends be selected according to the Q_1 network which consistently increases the discrepancy between the two Q-functions. In practice, we observe that the discrepancy between the two estimates of the Q-function, $|Q_1 - Q_2|$, can increase dramatically leading to an unstable learning process. An example is shown in Fig. 4 where $Q(s_{t+1}, \hat{a}_{t+1}; \theta_1)$ is always bigger than $Q(s_{t+1}, \hat{a}_{t+1}; \theta_2)$.

We introduce *Max-min Double Q-Learning* to reduce the discrepancy between the Q-functions. We first select $\hat{a}_{t+1}$ according to the actor network $\pi_\phi(s_{t+1})$. Then we run *CEM* to search the landscape of Q_2 within a broad neighborhood of $\hat{a}_{t+1}$ to return a second action $\tilde{a}_{t+1}$. Note that *CEM* selects an action $\tilde{a}_{t+1}$ that maximises Q_2 while the actor network selects an action $\hat{a}_{t+1}$ that maximises Q_1. We gather four different Q-values: $Q(s_{t+1}, \hat{a}_{t+1}; \theta_1)$, $Q(s_{t+1}, \hat{a}_{t+1}; \theta_2)$, $Q(s_{t+1}, \tilde{a}_{t+1}; \theta_1)$, and $Q(s_{t+1}, \tilde{a}_{t+1}; \theta_2)$. We then run a max-min operation to compute the target value that cancels the biases induced by $\hat{a}_{t+1}$ and $\tilde{a}_{t+1}$.

$$y = r_t + \gamma \max\{\min_{j=1,2} Q(s_{t+1}, \hat{a}_{t+1}; \theta_j), \min_{j=1,2} Q(s_{t+1}, \tilde{a}_{t+1}; \theta_j)\} \tag{10}$$

The inner min-operation $\min_{j=1,2} Q(s_{t+1}, \hat{a}_{t+1}; \theta_j)$ is adopted from Eq. 9 and mitigates overestimation [18]. The outer max operation helps to reduce the difference between Q_1 and Q_2. In addition, the max operation provides a better approximation of the Bellman optimality operator [17]. We visualize Q_1 and Q_2 during the learning process in Fig. 4. The following theorem formalizes the convergence of the proposed Max-min Double Q-Learning approach in the finite MDP setting. We prove the theorem in Appendix 3.3.

5 Experiments

5.1 Comparative Evaluation

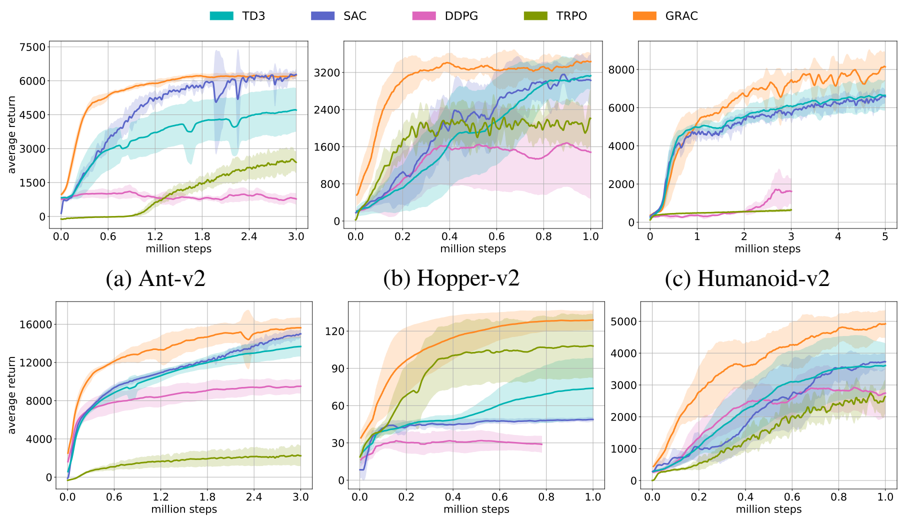
We present *GRAC*, a self-guided and self-regularized actor-critic algorithm as summarized in Algorithm 1. To evaluate *GRAC*, we measure its performance on the suite of MuJoCo continuous control tasks [19], interfaced through OpenAI Gym [3]. We compare our method with *DDPG* [11], *TD3* [6], *TRPO* [16], and *SAC* [7]. We use the source code released by the original authors and adopt the same hyperparameters reported in the original papers. Hyperparameters for all experiments are in Appendix 2.1. Results are shown in Figure 1. *GRAC* outperforms or performs comparably to all other algorithms in both final performance and learning speed across all tasks.

5.2 Ablation Study

In this section, we present ablation studies to understand the contribution of each proposed component: Self-Regularized TD-Learning (Section 4.1), Self-Guided Policy Improvment (Section 4.2), and Max-min Double Q-Learning (Section 4.3). We present our results in Fig. 3 in which we compare the performance of *GRAC* with alternatives, each removing one component from GRAC. Additional learning curves can be found in Appendix 2.2. We also run experiments to examine how sensitive GRAC is to some hyperparameters such as α and K listed in Alg. 1, and the results can be found in Appendix 2.4.

Self-Regularized TD Learning To verify the effectiveness of the proposed self-regularized TD-learning method, we apply our method to DDPG (*DDPG w/o target network w/ target regularization*). We compare against two baselines: the original DDPG and DDPG without target networks for both actor and critic (*DDPG w/o target network*). We choose DDPG, because it does not have additional components such as Double Q-Learning, which may complicate the analysis of this comparison.

In Fig. 2, we visualize the average returns, and average Q_1 values over training batchs (y'_1 in Alg.1). The Q_1 values of *DDPG w/o target network* changes dramatically which leads to poor average returns. *DDPG* maintains stable Q values but makes slow progress. Our proposed *DDPG w/o target network w/ target regularization* maintains stable Q values and learns considerably faster. This demonstrates the effectiveness of our method and its potentials to be applied to a wide range of DRL methods. Due to page limit, we only include results on Hopper-v2. The results on other tasks are in Appendix 2.3. All tasks exhibit a similar phenomenon.



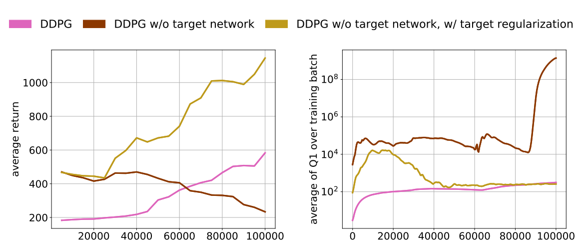
(a) Ant-v2 (b) Hopper-v2 (c) Humanoid-v2

(d) HalfCheetah-v2 (e) Swimmer-v2 (f) Walker2d-v2

Figure 1: Learning curves for the OpenAI gym continuous control tasks. For each task, we train 8 instances of each algorithm, using 8 different seeds. Evaluations are performed every 5000 interactions with the environment. Each evaluation reports the return (total reward), averaged over 10 episodes. For each training seed, we use a different seed for evaluation, which results in different start states. The solid curves and shaded regions represent the mean and standard deviation, respectively, of the average return over 8 seeds. All curves are smoothed with window size 10 for visual clarity. *GRAC* (orange) learns faster than other methods across all tasks. *GRAC* achieves comparable result to the state-of-the-art methods on the Ant-v2 task and outperforms prior methods on the other five tasks including the complex high-dimensional Humanoid-v2.

Policy Improvement with Evolution Strategies The GRAC actor network uses a combination of two actor loss functions, denoted as *QLoss* and *CEMLoss*. *QLoss* refers to the unbiased gradient estimators which extends the DDPG-style policy gradients [11] to stochastic policies. *CEMLoss* represents the policy improvement guided by the action found with the zero-order optmization method CEM. We run another two ablation experiments on all six control tasks and compare it with our original policy training method denoted as *GRAC*. As seen in Fig.3, in general *GRAC* achieves a better performance compared to either using *CEMLoss* or *QLoss*. The significance of the improvements varies within the six control tasks. For example, *CEMLoss* plays a dominant role in Swimmer while *QLoss* has a major effect in HalfCheetah. It suggests that *CEMLoss* and *QLoss* are complementary.

Max-min Double Q-Learning We additionally verify the effectiveness of the proposed Max-min Double Q-Learning method. We run an ablation experiment by replacing Max-min by Clipped Double Q-learing [6] denoted as *GRAC w/o CriticCEM*. In Fig. 4, we visualize the learning curves of the average return, Q_1 (y'_1 in Alg. 1), and $Q_1 - Q_2$ ($y'_1 - y'_2$ in Alg. 1). *GRAC* achieves high



(a) Returns on Hopper-v2 (b) Average of Q_1 over training batch on Hopper-v2

Figure 2: Learning curves and average Q_1 values (y'_1 in Alg. 1) on Hopper-v2. DDPG w/o target network quickly diverges as seen by the unrealistically high Q values. DDPG is stable but progresses slowly. If we remove the target network and add the proposed target regularization, we both maintain stability and achieve faster learning than DDPG.

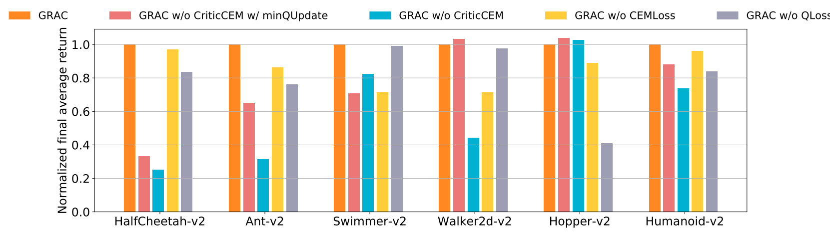
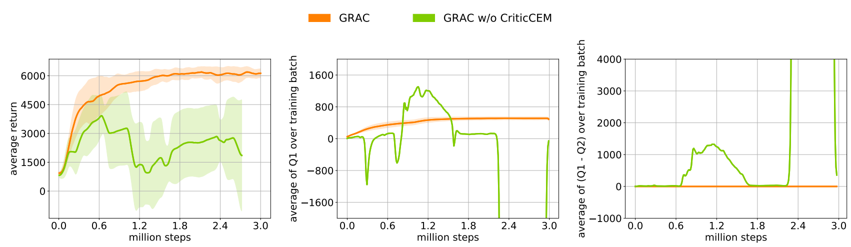


Figure 3: Final average returns, normalized w.r.t *GRAC* for all tasks. For each task, we train each ablation setting with 4 seeds, and average the last 10 evaluations of each seed (40 evaluations in total). Actor updates without CEMLoss (*GRAC w/o CEMLoss*) and actor updates w.r.t minimum of both Q networks (*GRAC w/o CriticCEM w/ minQUpdate*) achieves slightly better performance on Walker2d-v2 and Hopper-v2. GRAC achieves the best performance on 4 out of 6 tasks, especially on more complicated tasks with higher-dimensional state and action spaces (Humanoid-v2, Ant-v2, HalfCheetah-v2). This suggests that individual components of GRAC complement each other.

performance while maintaining a smoothly increasing Q function. Note that the difference between Q functions, $Q_1 - Q_2$, remains around zero for *GRAC*. *GRAC w/o CriticCEM* shows high variance and drastic changes in the learned Q_1 value. In addition, Q_1 and Q_2 do not always agree. Such unstable Q values result in a performance crash during the learning process. Instead of Max-min Double Q Learning, another way to address the gap between Q_1 and Q_2 is to perform actor updates on the minimum of Q_1 and Q_2 networks (as seen in SAC). Replacing Max-min Double Q Learning with this trick achieves lower performance than *GRAC* in more complicated tasks such as HalfCheetah-v2, Ant-v2, and Humanoid-v2 (See *GRAC w/o CriticCEM w/ minQUpdate* in Fig.3).



(a) Returns on Ant-v2

(b) Average of Q_1 over training batch on Ant-v2

(c) Average of $Q_1 - Q_2$ over training batch on Ant-v2

Figure 4: Learning curves (left), average Q_1 values (middle), and average of the difference between Q_1 and Q_2 (right) on Ant-v2. Average Q values are computes as minibatch average of y'_1 and y'_2, defined in Alg. 1. *GRAC w/o CriticCEM* represents replacing Max-min Double Q-Learning with Clipped Double Q-Learning. Without Max-min double Q-Learning to balance the magnitude of Q_1 and Q_2, Q_1 blows up significantly compared to Q_2, leading to divergence.

6 Conclusion

Leveraging neural networks as function approximators, DRL has been successfully demonstrated on a range of decision-making and control tasks. However, the nonlinear function approximators also introduce issues such as divergence and overestimation. We proposed a self-regularized TD-learning method to address divergence without requiring a target network that may slow down learning progress. The proposed method is agnostic to the specific Q-learning method and can be added to any of them. We introduced Max-min Double Q-learning to mitigate over-estimation while reducing the discrepancy between the two Q functions and to provide a better approximation of the Bellman optimality operator. In addition, we propose self-guided policy improvement by combining policy-gradient with zero-order optimization such as the Cross Entropy Method. This helps to search for actions associated with higher Q-values in a broad neighborhood and is robust to local noise in the Q function approximation. Taken together, these three components define *GRAC*, a novel self-guided and self-regularized actor critic algorithm. We evaluate our method on the suite of OpenAI gym tasks, achieving or outperforming state of the art in every environment tested.

7 Broader Impact

This work propose new methods to improve reinforcement learning in continuous control tasks. (1) DRL takes a lot of data, thus compute, to train. Our method speeds up training, thus reduces the necessary compute and the corresponding carbon footprint and energy consumption. (2) DRL's successful application in robotics would have societal impacts. On the positive side, automating repetitive manual labour increases productivity and thus increases the wealth of the society. On the other hand, automation may lead to unemployment of workers who have previously performed this manual labour. New policies are required to ensure decent income and access to professional education, so the impacted workers can transition into newly created jobs.

References

[1] Joshua Achiam, Ethan Knight, and Pieter Abbeel. Towards characterizing divergence in deep q-learning. *arXiv preprint arXiv:1903.08894*, 2019.

[2] Zdravko Botev, Dirk Kroese, Reuven Rubinstein, and Pierre L'Ecuyer. *The Cross-Entropy Method for Optimization*, volume 31, pages 35–59. 12 2013.

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

[4] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[5] Ishan Durugkar and Peter Stone. Td learning with constrained gradients. In *Proceedings of the Deep Reinforcement Learning Symposium, NIPS 2017*, Long Beach, CA, USA, December 2017.

[6] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[7] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[8] Hado V Hasselt. Double q-learning. In *Advances in neural information processing systems*, pages 2613–2621, 2010.

[9] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673, 2018.

[10] Seungchan Kim, Kavosh Asadi, Michael Littman, and George Konidaris. Deepmellow: removing the need for a target network in deep q-learning.

[11] 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 *International Conference on Learning Representations*, 2016.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[13] 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, 2015.

[14] Pourchot and Sigaud. CEM-RL: Combining evolutionary and gradient-based methods for policy search. In *International Conference on Learning Representations*, 2019.

[15] R. Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 1:127—-190, 1999.

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

[17] Richard S Sutton et al. *Introduction to reinforcement learning*, volume 135.

[18] Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning.

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

[20] John N Tsitsiklis and Benjamin Van Roy. Analysis of temporal-diffference learning with function approximation. In *Advances in neural information processing systems*, pages 1075–1081, 1997.

[21] Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*, 2018.

[22] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford, 1989.

[23] M. Yan, A. Li, M. Kalakrishnan, and P. Pastor. Learning probabilistic multi-modal actor models for vision-based robotic grasping. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 4804–4810, 2019.