

# Appendix: GRAC

Lin Shao<sup>1</sup>, Yifan You<sup>2</sup>, Mengyuan Yan<sup>1</sup>, Shenli Yuan<sup>1</sup>, Qingyun Sun<sup>3</sup>, Jeannette Bohg<sup>1</sup>

<sup>1</sup>Stanford AI Lab, <sup>3</sup>Department of Mathematics, Stanford University  
{lins2, mengyuan, shenliy, qysun, bohg}@stanford.edu

<sup>2</sup>Department of Computer Science, University of California, Los Angeles  
harry473417@ucla.edu

## 1 Implementation Details

### 1.1 Neural Network Architecture Details

We use a two layer feedforward neural network of 256 and 256 hidden nodes respectively. Rectified linear units (ReLU) are put after each layer for both the actor and critic except the last layer. For the last layer of the actor network, a tanh function is used as the activation function to squash the action range within  $[-1, 1]$ . *GRAC* then multiplies the output of the tanh function by *max action* to transform  $[-1, 1]$  into  $[-\text{max action}, \text{max action}]$ . The actor network outputs the mean and sigma of a Gaussian distribution.

### 1.2 CEM Implementation

Our CEM implementation is based on the CEM algorithm described in Pourchot [1].

---

#### Algorithm 1 CEM

---

Input: Q-function  $Q(s, a)$ ; size of population  $N_{pop}$ ; size of elite  $N_{elite}$  where  $N_{elite} \leq N_{pop}$ ; max iteration of CEM  $N_{cem}$ .

Initialize the mean  $\mu$  and covariance matrix  $\Sigma$  from actor network predictions.

- 1: **for**  $i = 1 \dots, N_{cem}$  **do**
- 2:   Draw the current population set  $\{a_{pop}\}$  of size  $N_{pop}$  from  $\mathcal{N}(\mu, \Sigma)$ .
- 3:   Receive the  $Q$  values  $\{q_{pop}\} = \{Q(s, a) | a \in \{a_{pop}\}\}$ .
- 4:   Sort  $\{q_{pop}\}$  in descending order.
- 5:   Select top  $N_{elite}$   $Q$  values and choose their corresponding  $a_{pop}$  as elite  $\{a_{elite}\}$ .
- 6:   Calculate the mean  $\mu$  and covariance matrix  $\Sigma$  of the set  $\{a_{elite}\}$ .
- 7: **end for**

Output: The top one elite in the final iteration.

---

### 1.3 Additional Detail on Algorithm 1: GRAC

The actor network outputs the mean and sigma of a Gaussian distribution. In Line 2 of Alg.1, the actor has to select action  $a$  based on the state  $s$ . In the test stage, the actor directly uses the predicted mean as output action  $a$ . In the training stage, the actor first samples an action  $\hat{a}$  from the predicted Gaussian distribution  $\pi_\phi(s)$ , then *GRAC* runs *CEM* to find a second action  $\tilde{a} = \text{CEM}(Q(s, \cdot; \theta_2), \pi_\phi(\cdot | s))$ . *GRAC* uses  $a = \arg \max_{\{\tilde{a}, \hat{a}\}} \{\min_{j=1,2} Q(s, \tilde{a}; \theta_j), \min_{j=1,2} Q(s, \hat{a}; \theta_j)\}$  as the final action.

## 2 Appendix on Experiments

### 2.1 Hyperparameters used

Table 1 and Table 2 list the hyperparameters used in the experiments.  $[a, b]$  denotes a linear schedule from  $a$  to  $b$  during the training process.

Parameters	Values
discount $\gamma$	0.99
replay buffer size	1e6
batch size	256
optimizer	Adam [2]
learning rate in critic	3e-4
learning rate in actor	2e-4
$N_{cem}$	2
$N_{pop}$	256
$N_{elite}$	5

Table 1: Hyperparameter Table

Environments	ActionDim	$K$ in Alg.1	$\alpha$ in Alg.1	CemLossWeight	Reward Scale
Ant-v2	8	20	[0.7, 0.9]	1.0/ActionDim	1.0
Hopper-v2	3	20	[0.85, 0.95]	0.3/ActionDim	1.0
HalfCheetah-v2	6	50	[0.7, 0.85]	1.0/ActionDim	0.5
Humanoid-v2	17	20	[0.7, 0.85]	1.0/ActionDim	1.0
Swimmer-v2	2	20	[0.5, 0.75]	1.0/ActionDim	1.0
Walker2d-v2	6	20	[0.8, 0.9]	0.3//ActionDim	1.0

Table 2: Environment Specific Parameters

### 2.1.1 Additional Learning Curves for Policy Improvement with Evolution Strategy

The learning curves are shown in Fig 1.

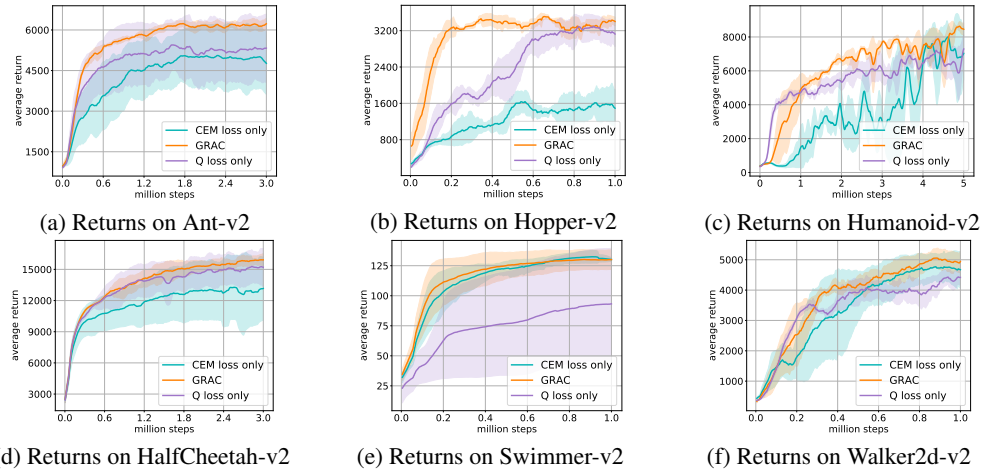


Figure 1: Learning curves for the OpenAI gym continuous control tasks. The *GRAC* actor network uses a combination of two actor loss functions, denoted as *QLoss* and *CEMLoss*. *QLoss Only* represents the actor network only trained with *QLoss*. *CEM Loss Only* represents the actor network only trained with *CEMLoss*. In general *GRAC* achieves a better performance compared to either using *CEMLoss* or *QLoss*.

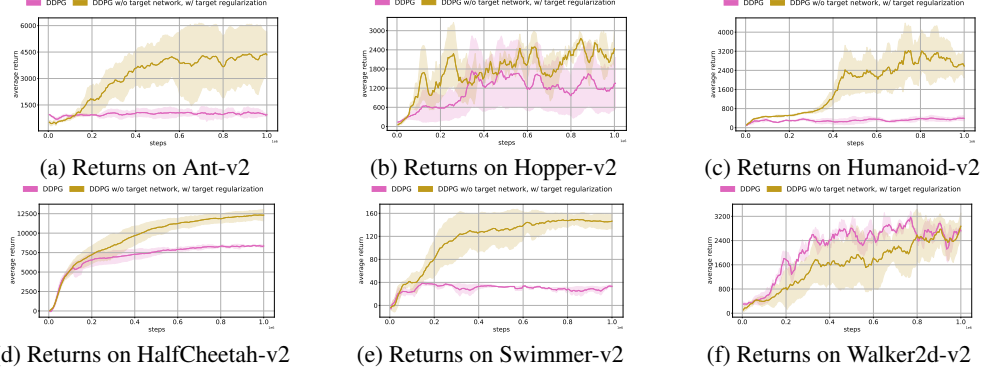


Figure 2: Learning curves of *DDPG w/o target network, w/ target regularization* and *DDPG* on the OpenAI gym continuous control tasks within one million steps over four random seeds. *DDPG w/o target network, w/ target regularization* outperforms *DDPG* by large margins in five out of six Mujoco tasks.

## 2.2 Additional Learning Curves for Ablation Study of Self-Regularized TD Learning

We report results in Figs 2 and 3.

In addition to learning curves, we also plot in Figure 4 how the regularization term changes as training progress, with and without self-regularized TD learning.

## 2.3 Hyperparameter Sensitivity for the Termination Condition of Critic Network Training

We also run experiments to examine how sensitive *GRAC* is to some hyperparameters such as  $K$  and  $\alpha$  listed in Alg.1. The critic networks will be updated until the critic loss has decreased to  $\alpha$  times the original loss, or at most  $K$  iterations, before proceeding to update the actor network. In practice, we decrease  $\alpha$  in the training process. Fig.5 shows five learning curves on Ant-v2 running with five different hyperparameter values. We find that a moderate value of  $K = 10$  is enough to stabilize the training process, and increasing  $K$  further does not have significant influence on training, shown on the right of Fig.5.  $\alpha$  is usually within the range of  $[0.7, 0.9]$  and most tasks are not sensitive to minor changes. However on the task of Swimmer-v2, we find that  $\alpha$  needs to be small enough ( $< 0.7$ ) to prevent divergence. In practice, without appropriate  $K$  and  $\alpha$  values, divergence usually happens within the first 50k training steps, thus it is quick to select appropriate values for  $K$  and  $\alpha$ .

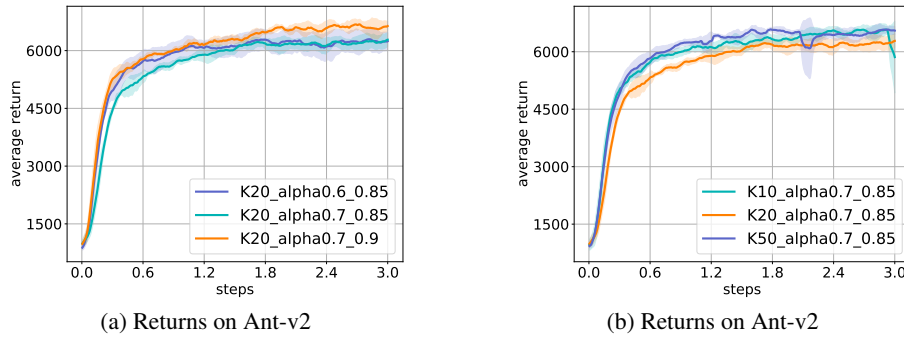


Figure 5: Learning curves for the OpenAI gym Ant-v2 environment.

## 2.4 Additional Learning Curves for OpenAI Gym Tasks

Due to time limit, we did not finished running *GRAC* on all 25 seeds. In Figure 6, we plot all the seeds we have for *GRAC*, which includes 20 seeds for Ant-v2, 15 seeds for Hopper-v2, 13 seeds for Humanoid-v2, 12 seeds for HalfCheetah-v2, 15 seeds for Swimmer-v2, and 10 seeds for Walker2d-v2.

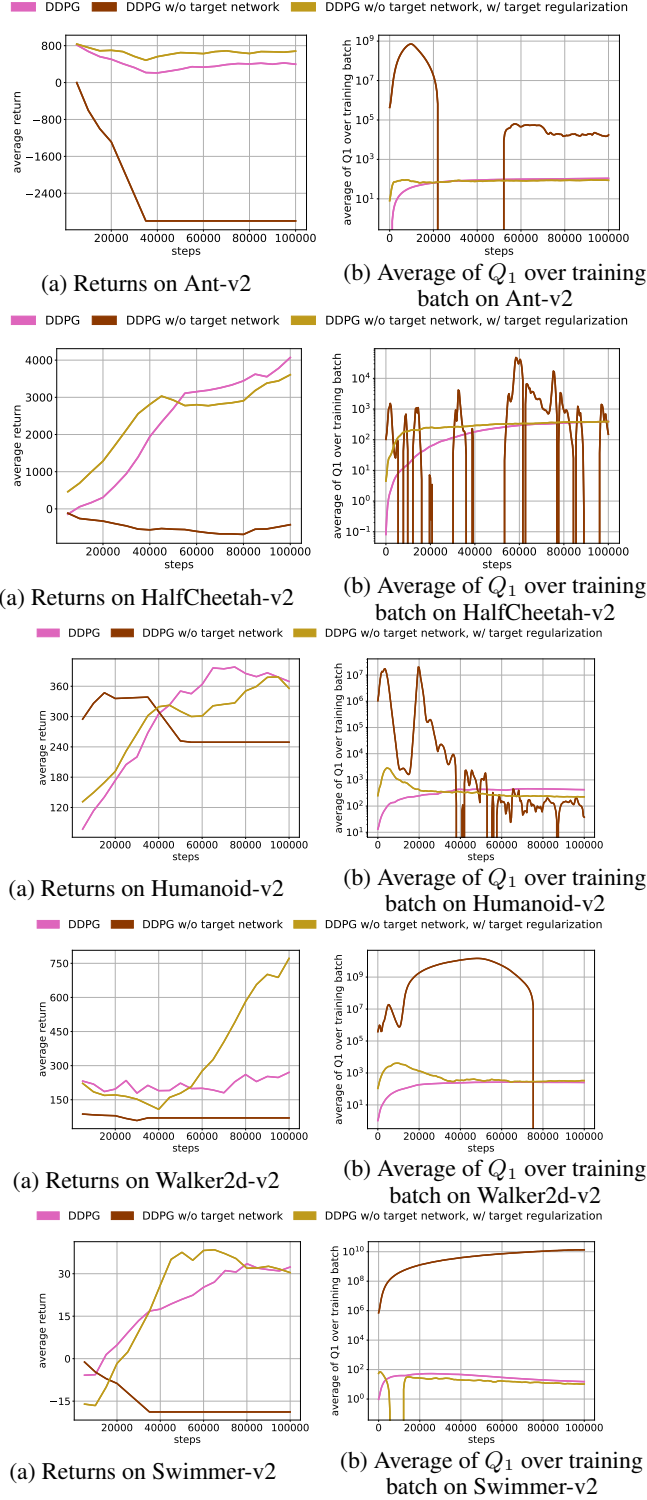


Figure 3: Learning curves and average  $Q_1$  values ( $y'_1$  in Alg. 1 of the main paper). *DDPG* w/o target network quickly diverges as seen by the unrealistically high  $Q$  values. *DDPG* is stable but often progresses slower. If we remove the target network and add the proposed target regularization, we both maintain stability and achieve a faster or comparable learning rate.

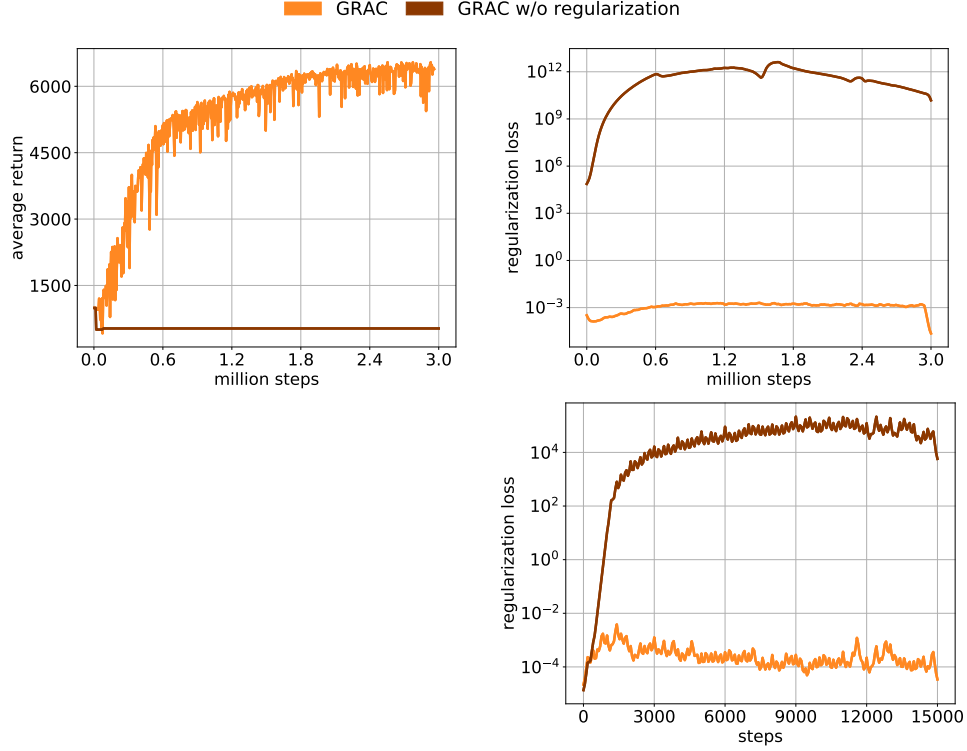


Figure 4: Learning curves (top left) and the self-regularized TD component (top right, bottom right) on Ant-v2. Top right and bottom right both plot the regularization term, except top right plots 3 million time-steps, while bottom right plots 15000 time-steps. GRAC w/o regularization achieves very low average returns, and the regularization term explodes.

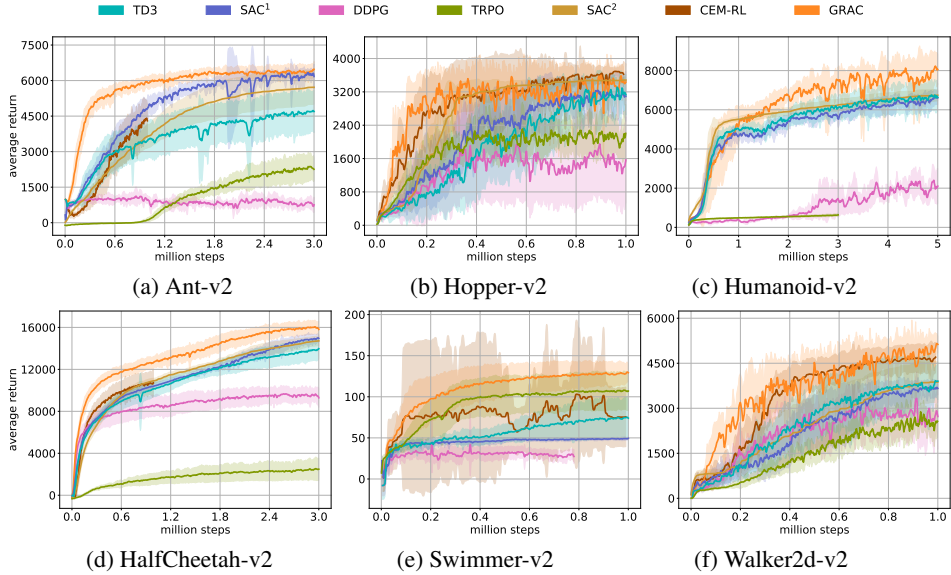


Figure 6: Learning curves for the OpenAI gym continuous control tasks. All curves are based on 10 seeds except for GRAC. Due to time limit, we have not finished running all 25 seeds for GRAC. For GRAC, we plot 20 seeds for Ant-v2, 15 seeds for Hopper-v2, 13 seeds for Humanoid-v2, 12 seeds for HalfCheetah-v2, 15 seeds for Swimmer-v2, and 10 seeds for Walker2d-v2.

## 2.5 Robustness of GRAC to Local Noise in the Q Function

We design a comparison experiment to demonstrate that GRAC is robust to local noise in the Q function. We add noise to the Q function by adding noise to the last fully-connected layer of the Q function at every time-step. For each weight parameter, the noise is sampled from a Gaussian distribution of mean zero and variance one percent of the parameter magnitude. We add the noise in a similar way to TD3 [3] and compare their relative performances with and without noise in the Q function. Due to limited time, we run the experiment only on Ant-v2. Learning curves are available in Figure 7.

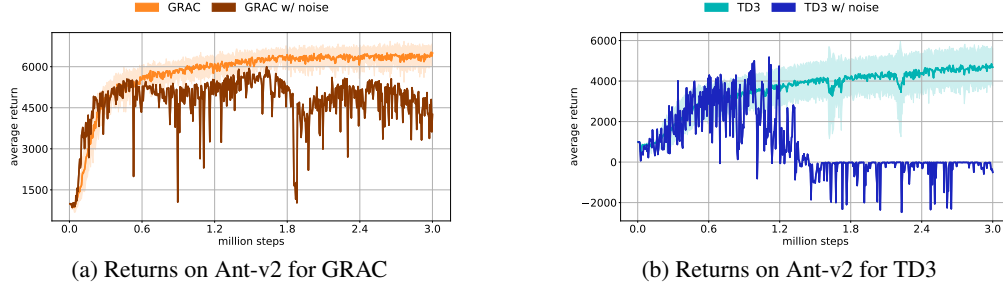


Figure 7: Learning curves for GRAC and TD3, with and without noise in the Q function. We add noise to the Q function by adding random Gaussian noise to the last fully-connected layer of the Q function at every time-step. We evaluate the result on Ant-v2 for three million time-steps. GRAC can still achieve a relatively high reward compared to without noise, while TD3 experiences high instability when noise is added in the Q function.

## 2.6 Extra Compute Cost of CEM in GRAC

The CEM component in GRAC introduces additional compute cost. Adding the CEM search in both the actor and critic slows down training clock time by 10% compared to GRAC without using CEM.

## 3 Theorems and Proofs

For the sake of clarity, we make the following technical assumption about the function approximation capacity of neural networks that we use to approximate the action distribution.

**State separation assumption:** The neural network chosen to approximate the policy family  $\Pi$  is expressive enough to approximate the action distribution for each state  $\pi(s, \cdot)$  separately.

### 3.1 Theorem 1: Q-loss Policy Improvement

**Theorem 1.** *Starting from the current policy  $\pi$ , we update the policy to 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  $\pi_{new}$ . Then the induced Q function,  $Q^{\pi_{new}}$ , satisfies  $\forall (s, a), Q^{\pi_{new}}(s, a) \geq Q^\pi(s, a)$ .*

*Proof of Theorem 1.* Under the state separation assumption, the action distribution for each state,  $\pi(s, \cdot)$ , can be updated separately, for each state we are maximizing  $\mathbb{E}_{a \sim \pi(s, \cdot)} Q^\pi(s, a)$ . Therefore, we have  $\forall s, \mathbb{E}_{a \sim \pi_{new}(s, \cdot)} Q^\pi(s, a) \geq \mathbb{E}_{a \sim \pi(s, \cdot)} Q^\pi(s, a) = V^\pi(s)$ .

$$\begin{aligned}
 Q^\pi(s, a) &= r(s, a) + \gamma \mathbb{E}_{s'} V^\pi(s') \\
 &\leq r(s, a) + \gamma \mathbb{E}_{s'} \mathbb{E}_{a' \sim \pi_{new}} Q^\pi(s', a') \\
 &= r(s, a) + \gamma \mathbb{E}_{s'} \mathbb{E}_{a' \sim \pi_{new}} [r(s', a') + \gamma \mathbb{E}_{s''} V^\pi(s'')] \\
 &\leq r(s, a) + \gamma \mathbb{E}_{s'} \mathbb{E}_{a' \sim \pi_{new}} r(s', a') + \gamma^2 \mathbb{E}_{s'} \mathbb{E}_{a' \sim \pi_{new}} \mathbb{E}_{s''} \mathbb{E}_{a'' \sim \pi_{new}} Q^\pi(s'', a'') \\
 &= \dots \text{ (repeatedly unroll Q function) } \\
 &\leq Q^{\pi_{new}}(s, a)
 \end{aligned} \tag{1}$$

□

### 3.2 Theorem 2: CEM Policy Improvement

**Theorem 2.** We assume that 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  $Q^\pi$  is the  $Q$  function induced by the current policy  $\pi$ . 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  $\pi_{new}$ . Then  $Q^{\pi_{new}}$  satisfies  $\forall(s, a), Q^{\pi_{new}}(s, a) \geq Q^\pi(s, a)$ .

*Proof of Theorem 2.* Under the state separation assumption, the action distribution for each state,  $\pi(s, \cdot)$ , can be updated separately. Then, for each state  $s$ , the policy  $\pi_{new}$  will converge to a delta function at  $a^*(s)$ . Therefore we have  $\forall s, \max_a Q^\pi(s, a) = \mathbb{E}_{a \sim \pi_{new}(s, \cdot)} Q^\pi(s, a) \geq \mathbb{E}_{a \sim \pi(s, \cdot)} Q^\pi(s, a) = V^\pi(s)$ . Then, following Eq. (1) we have  $\forall(s, a), Q^{\pi_{new}}(s, a) \geq Q^\pi(s, a)$   $\square$

### 3.3 Theorem 3: Max-Min Double Q-learning Convergence

**Theorem 3.** We keep two tabular value estimates  $Q_1$  and  $Q_2$ , and update via

$$\begin{aligned} Q_{t+1,1}(s, a) &= Q_{t,1}(s, a) + \alpha_t(s, a)(y_t - Q_{t,1}(s, a)) \\ Q_{t+1,2}(s, a) &= Q_{t,2}(s, a) + \alpha_t(s, a)(y_t - Q_{t,2}(s, a)), \end{aligned} \quad (2)$$

where  $\alpha_t(s, a)$  is the learning rate and  $y_t$  is the target:

$$\begin{aligned} y_t &= r_t(s_t, a_t) + \gamma \max_{a' \in \{a^\pi, a^*\}} \min_{i \in \{1,2\}} Q_{t,i}(s_{t+1}, a') \\ a^\pi &\sim \pi(s_{t+1}) \\ a^* &= \operatorname{argmax}_{a'} Q_{t,2}(s_{t+1}, a') \end{aligned}$$

We assume that the MDP is finite and tabular and the variance of rewards are bounded, and  $\gamma \in [0, 1]$ . We assume each state action pair is sampled an infinite number of times and both  $Q_1$  and  $Q_2$  receive an infinite number of updates. We further assume the learning rates satisfy  $\alpha_t(s, a) \in [0, 1]$ ,  $\sum_t \alpha_t(s, a) = \infty$ ,  $\sum_t [\alpha_t(s, a)]^2 < \infty$  with probability 1 and  $\alpha_t(s, a) = 0, \forall(s, a) \neq (s_t, a_t)$ . Finally we assume CEM is able to find the optimal action  $a^*(s) = \arg \max_{a'} Q(s, a'; \theta_2)$ . Then Max-Min Double Q-learning will converge to the optimal value function  $Q^*$  with probability 1.

*Proof of Theorem 3.* This proof will closely follow Appendix A of [3].

We will first prove that  $Q_2$  converges to the optimal Q value  $Q^*$ . Following notations of [3], we have

$$\begin{aligned} F_t(s_t, a_t) &\triangleq y_t(s_t, a_t) - Q^*(s_t, a_t) \\ &= r_t + \gamma \max_{a' \in \{a^\pi, a^*\}} \min_{i \in \{1,2\}} Q_{t,i}(s_{t+1}, a') - Q^*(s_t, a_t) \\ &= F_t^Q(s_t, a_t) + c_t \end{aligned} \quad (3)$$

Where

$$\begin{aligned} F_t^Q(s_t, a_t) &= r_t + \gamma Q_{t,2}(s_{t+1}, a^*) - Q^*(s_t, a_t) \\ &= r_t + \gamma \max_{a'} Q_{t,2}(s_{t+1}, a') - Q^*(s_t, a_t) \\ c_t &= \gamma \max_{a' \in \{a^\pi, a^*\}} \min_{i \in \{1,2\}} Q_{t,i}(s_{t+1}, a') - \gamma Q_{t,2}(s_{t+1}, a^*) \end{aligned}$$

$F_t^Q$  is associated with the optimum Bellman operator. It is well known that the optimum Bellman operator is a contractor, We need to prove  $c_t$  converges to 0.

Based on the update rules (Eq. (A2)), it is easy to prove that for any tuple  $(s, a)$ ,  $\Delta_t(s, a) = Q_{t,1}(s, a) - Q_{t,2}(s, a)$  converges to 0. This implies that  $\Delta_t(s, a^\pi) = Q_{t,1}(s, a^\pi) - Q_{t,2}(s, a^\pi)$  converges to 0 and  $\Delta_t(s, a^*) = Q_{t,1}(s, a^*) - Q_{t,2}(s, a^*)$  converges to 0. Therefore,  $\min_{i \in \{1,2\}} Q_{t,i}(s, a) - Q_{t,2}(s, a) \leq 0$  and the left hand side converges to zero, for  $a \in a^\pi, a^*$ . Since we have  $Q_{t,2}(s, a^*) \geq Q_{t,2}(s, a^\pi)$ , then

$$\min_{i \in \{1,2\}} Q_{t,i}(s, a^*) \leq \max_{a' \in \{a^\pi, a^*\}} \min_{i \in \{1,2\}} Q_{t,i}(s, a') \leq Q_{t,2}(s, a^*)$$

Therefore  $c_t = \gamma \max_{a' \in \{a^\pi, a^*\}} \min_{i \in \{1,2\}} Q_{t,i}(s, a') - Q_{t,2}(s, a^*)$  converges to 0. And we proved  $Q_{t,2}$  converges to  $Q^*$ .

Since for any tuple  $(s, a)$ ,  $\Delta_t(s, a) = Q_{t,1}(s, a) - Q_{t,2}(s, a)$  converges to 0,  $Q_{t,1}$  also converges to  $Q^*$ .  $\square$

### 3.4 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-learning [3] denoted as *GRAC w/o CriticCEM*. In Fig. 8, we visualize the learning curves of the average return,  $Q_1$  ( $y'_1$  in Alg. ??), and  $Q_1 - Q_2$  ( $y'_1 - y'_2$  in Alg.1). *GRAC* achieves high 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.

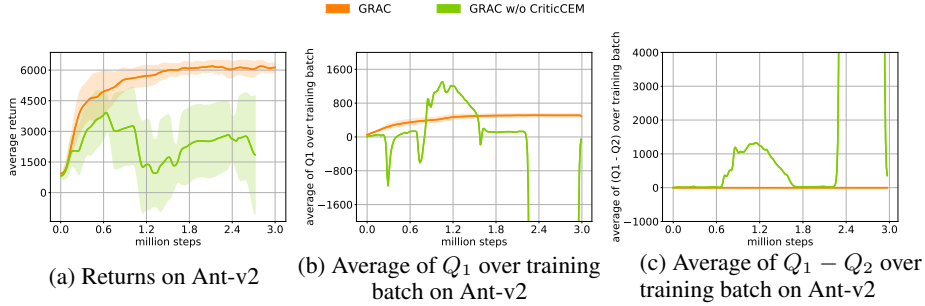


Figure 8: Learning curves (left), average  $Q_1$  values (middle), and average of the difference between  $Q_1$  and  $Q_2$  (right) on Ant-v2. Average Q values are computed as minibatch average of  $y'_1$  and  $y'_2$ , defined in Alg.1. *GRAC w/o CriticCEM* represents replacing Max-min Double Q-Learning 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.

## 4 In-Hand Manipulation Experiment

Besides the six mujoco locomotion tasks, we demonstrate that GRAC can learn a policy to solve in-hand manipulation tasks on a complex multi-fingered hand. We train the robotic hand [4] in simulation to rotate a cube by 50 degrees around the direction of gravity and demonstrate the zero-shot transfer capability to the real world. We use the simulation environment in [4] based on Mujoco 2.0. and a visualization of the simulation is in Fig 6. of [4]. The input states to the policy are the current, previous, initial, and target object position and orientation, and all nine gripper joint positions at the current time step. The actions are nine joint positions for the gripper at the next time step. We implement a function to linearly map the action range  $[-1, 1]$  of GRAC to the actuator range of the hand [lower joint limit, upper joint limit]. We use the same evaluation metric introduced in [4] which calculates the pose difference between the current pose and the target pose denoted as  $Er_t$ . The reward at time step  $t$  is  $Er_{t-1} - Er_t$ . The task is considered successful if the pose difference at the current step is less than 10% of the pose difference at the initial step. We train the policy for 500k iterations in simulation and transfer the learned policy to the real hand without finetuning. The real-world experimental setup includes the robotic hand and an overhead Intel Realsense D400series RGBD camera. To track object position and orientation in the real world, QR-tags are put on all cube's six faces. We test three trials in the real world. GRAC learns a policy that successfully completes all three trials demonstrating the proposed algorithm's effectiveness at solving in-hand manipulation tasks on a complex multi-fingered hand. The video is available in the supplementary material.

#### 4.1 Results on In-Hand Manipulation Task

We train GRAC, TD3 [3], and SAC [5] for five hundred thousand time-steps. Figure 9 shows the corresponding learning curves. All three algorithms discovered a policy that produces maximum reward, which corresponds to task success. There are no significant differences between the algorithms' performance on this specific task. The motivation of this experiment is to demonstrate that GRAC has the potential to be useful in robotic manipulation tasks.

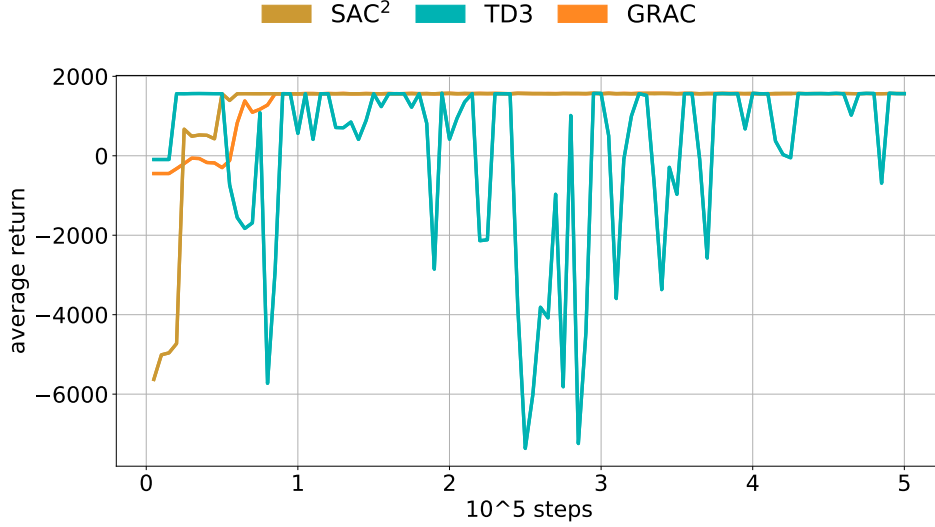


Figure 9: Learning curves for the in-hand manipulation task. We train each algorithm for five hundred thousand time-steps. There are no significant differences between different algorithms at this specific task. Both SAC and GRAC converge to the maximum reward, corresponding to task success. This result indicates that GRAC has the potential to be applied to robotic manipulation tasks.

## Acknowledgments

Toyota Research Institute ("TRI") provided funds to assist the authors with their research but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity. This article is also supported by Siemens and the HAI-AWS Cloud Credits. We thank reviewers for their useful comments.

## References

- [1] Pourchot and Sigaud. CEM-RL: Combining evolutionary and gradient-based methods for policy search. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=BkeU5jOctQ>.
- [2] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Int. Conf. on Learning Representations (ICLR)*, 2015.
- [3] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.
- [4] S. Yuan, L. Shao, C. L. Yako, A. Gruebele, and J. K. Salisbury. Design and control of roller grasper v2 for in-hand manipulation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.
- [5] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.