

## Appendix A. The shape of Swimmer-v2

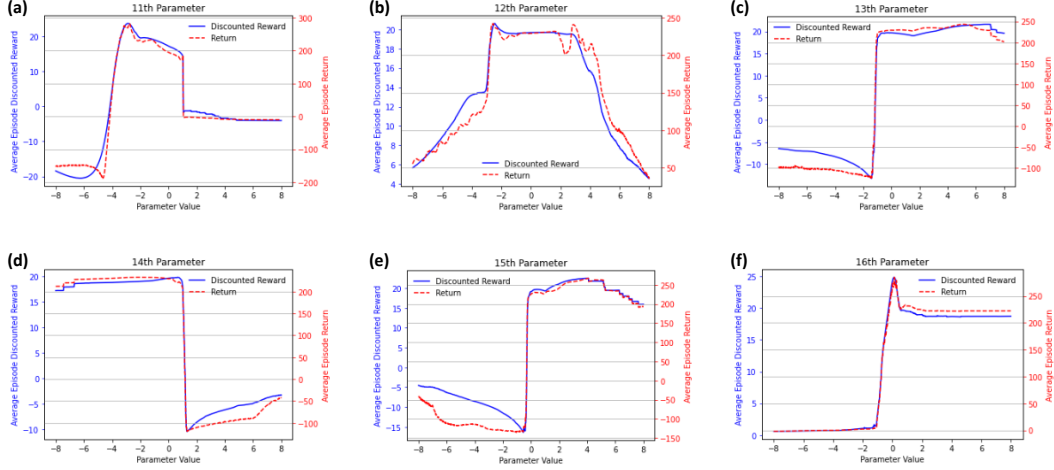


Figure 1: The graphs represent the change of average episodic discounted reward and average episodic return of a linear policy actor by the value change of a single parameter in the Swimmer, where the other parameters were fixed and we evaluated thirty times at different seeds for each policy.

$$\text{Average Episode Discounted Reward} = E_{\pi(\cdot|s)}[\sum_{t=0}^{\infty} \gamma^t r_t^\pi] \approx \frac{1}{30} \sum_{n=1}^{30} \sum_{t=0}^{999} \gamma^t r_t^\pi$$

$$\text{Average Episode Return} = E_{\pi(\cdot|s)}[\sum_{t=0}^{\infty} r_t^\pi] \approx \frac{1}{30} \sum_{n=1}^{30} \sum_{t=0}^{999} r_t^\pi$$

where  $\gamma = 0.99$ , and the architecture of the linear policy : [state dim, action dim]  $\rightarrow$  tanh

To find the reason why the previous state-of-the-art policy gradient methods, such as TRPO, PPO, SAC, and TD3, were hard to solve the Swimmer in the Mujoco Environment, we performed an interesting two-steps experiment. In the first step, we sat an actor, following a simple linear policy, and executed the CEM algorithm to find an optimal policy parameters  $\theta$  in the Swimmer. When the Return value of the actor reached over 200, we saved the policy parameters  $\theta^{200}$ . In the next step, while changing one parameter value, we evaluated the changed policy thirty times with different seeds and recorded the all Discounted Rewards ( $J(S_0)$ ) and Returns. We represents the some cases of the experiment in Figure 1.

As you can find in the Figure 1, we can discover interesting facts: 1) on the graphs (a), (d), and (f) in Figure 1, we can find wide regions, where the gradients  $\nabla_{\theta} J(S_0)$  are near zero; 2) except for (b) in Figure 1, at particular parameter value, the gradient of  $J(S_0)$  and Return is steep enough to be seen as a piece-wise; and 3) on the graphs (c), (d) and (e), the graphs are shaped like valleys neat at those steep points. We suspected the above facts as the cause of the deceptive gradient problem of the Swimmer, and raised the question in the Introduction. Finally, by considering those issues, we propose the combined algorithm with TD3 and CEM.

## Appendix B. Pseudocode of PGPS Algorithm

---

### Algorithm 1 Coupling Policy Gradient with Population based Search Algorithm

---

**Set hyper-parameters:** TD3:  $lr_{actor}, lr_{critic}, \tau, \gamma$ , and  $\kappa$ ; CEM: pop-size  $N$ , top  $K$ ,  $\Sigma_{init}, \Sigma_{end}$ , and  $\tau_{\Sigma}$ ; Mutual guidance:  $F_{TD3 \rightarrow CEM}, \beta_{init}$ , and  $D_{target}$ ; Q-critic filtering:  $T_{start-Q}$  and  $SR$ ; Increasing interaction steps:  $T_{max}, T_{init}$ , and  $T_{interval}$

- 1: Initialize the mean  $\mu$  of the multivariate Gaussian of the CEM
- 2: Initialize the TD3 actor  $\pi_{TD3}$  and TD3 critic  $Q_{TD3}$
- 3: initialize replay buffer  $R$
- 4: total\_steps = 0
- 5: **for** generation=1: $\infty$  **do**
- 6:   **if** total\_steps  $\geq T_{start-Q}$  **then**
- 7:      $pop \leftarrow$  Q-critic Filtering( $N, \pi_{elite}, \mu, \Sigma, Q_{TD3}, R$ )
- 8:   **else**
- 9:      $pop[1] \leftarrow \pi_{elite}, pop[2 : N]$  are sampled from  $N(\mu, \Sigma)$
- 10:   **end if**
- 11:   **if** generation mod  $F_{TD3 \rightarrow CEM} = 0$  **then**
- 12:      $pop[N] \leftarrow \pi_{TD3}$
- 13:   **end if**
- 14:    $T = \min(T_{init} + 100 \times \text{mod}(\text{total\_steps}, T_{interval}), T_{max})$
- 15:   interaction\_steps = 0
- 16:   **for** i=1:pop\_size  $N$  **do**
- 17:     Set the current actor  $\pi$  as  $pop[i]$ .
- 18:      $\text{Return}_i, (s_t, a_t, r_t, s_{t+1})_{t=1:t_{end}(t_{end} \leq T)} \leftarrow \text{Evaluate}(\pi, T)$
- 19:     Fill replay buffer  $R$  with  $(s_t, a_t, r_t, s_{t+1})_{t=1:t_{end}}$
- 20:     interaction\_steps = interaction\_steps +  $t_{end}$
- 21:   **end for**
- 22:   total\_steps = total\_steps + interaction\_steps
- 23:   Update  $(\pi_{elite}, \mu, \Sigma)$  with the top- $K$  Return actors
- 24:   num\_update = interaction\_steps / 5
- 25:   **if** generation mod  $F_{TD3 \rightarrow CEM} = 0$  and  $\text{Return}_N < \text{MEAN}(\text{Returns}) - \text{STD}(\text{Returns})$  **then**
- 26:     **for** i=1:5 **do**
- 27:       Sampled states 2 ( $SS_2$ ) are drawn from  $R$
- 28:       Update  $\beta = \begin{cases} \beta \times 2 & \text{if } E_{s \sim SS_2}[\|\pi_{TD3}(s) - \pi_{elite}(s)\|_2^2] > D_{target} \times 1.5 \\ \beta / 2 & \text{if } E_{s \sim SS_2}[\|\pi_{TD3}(s) - \pi_{elite}(s)\|_2^2] < D_{target} / 1.5 \end{cases}$
- 29:       Train  $Q_{TD3}$  for num\_update mini-batches from  $R$  using a standard TD3 algorithm
- 30:       Train  $\pi_{TD3}$  for num\_update mini-batches from  $R$  to minimize  $L^A(\pi_{TD3}, \pi_{elite}, \beta)$
- 31:     **end for**
- 32:   **else**
- 33:     **for** i=1:5 **do**
- 34:       Train  $Q_{TD3}$  for num\_update mini-batches from  $R$  using a standard TD3 algorithm
- 35:       Train  $\pi_{TD3}$  for num\_update mini-batches from  $R$  to minimize  $L^O(\pi_{TD3})$
- 36:     **end for**
- 37:   **end if**
- 38: **end for**

---

In contrast to a vanilla TD3[2], performing backpropagation per one step interaction with environment, our TD3 periodically performs backpropagation alternately with the population based search of CEM. In one update period, our TD3 carries out backpropagation as many times as the total interaction steps contained in the former CEM evaluation. For example, if there occurred 10000 steps in CEM evaluation, after that TD3 update its critic and actor five times, 2000 times each. This update process is widely used in previous studies in order to stabilize the volatility of the critic [1, 3, 4].

In the pseudocode,  $L^O(\pi_{TD3})$  is the loss function of the vanilla TD3, represented in equation (1).  $L^A(\pi_{TD3}, \pi_{elite}, \beta)$  is the augmented loss function for guided policy learning, represented in equation (2).

For exploration, the vanilla TD3 applies action space noise by adding Gaussian noise or Ornstein-Uhlenbeck[5] noise to the action  $a_t$ . However, Pourchot and Sigaud [4] empirically showed that the action space noise does not bring performance improvement. We also could not find any improvement potential when applying it at our TD3. So, we determined not to utilize the action space noise in the proposed algorithms.

---

**Algorithm 2** Function Q-critic Filtering

---

```

1: procedure Q-critic Filtering( $N, \pi_{elite}, \mu, \Sigma, Q_{TD3}, R$ )
2:    $pop[1] \leftarrow \pi_{elite}$ ,  $pop[2 : N/2]$  are sampled from  $N(\mu, \Sigma)$ 
3:    $\pi_{j=1:M}(=SR*N)$  are sampled from  $N(\mu, \Sigma)$ 
4:   Sampled states 1 ( $SS_1$ ) are drawn from replay buffer  $R$ 
5:   for  $j=1:M$  do
6:      $P_j = E_{s \sim SS_1}[Q_{TD3}(SS_1, \pi_j)]$ 
7:   end for
8:    $pop[N/2 + 1, N] \leftarrow$  Select policies ( $\pi$ s) with higher  $P_j$  among  $\pi_{j=1:M}$ 
9:   Return  $pop$ 
10: end procedure

```

---



---

**Algorithm 3** Function Evaluate

---

```

1: procedure Evaluate( $\pi_{elite}, T$ )
2:    $returns, t, \text{buffer } (BF) = 0, 0, []$ 
3:   Reset environment and get initial state  $s_0$ 
4:   while env is not done and  $t \leq T$  do
5:     Select action  $a_t = \pi(s_t)$ 
6:     Execute action  $a_t$  and receive reward  $r_t$  and next state  $s_{t+1}$ 
7:     Fill  $BF$  with stat transition  $(s_t, a_t, r_t, s_{t+1})$ 
8:      $returns = returns + r_t$  and  $t = t + 1$ 
9:   end while
10:  Return  $returns, BF$ 
11: end procedure

```

---

## Appendix C. Proof of Proposition 1

In this section, we prove Proposition 1.

**Proposition1** If  $E_{a \sim \pi_i(\cdot|s)}[Q_{TD3}(s, a)] \geq E_{a \sim \pi_{TD3}(\cdot|s)}[Q_{TD3}(s, a)]$  for all  $s$ ,  
 $E_{a \sim \pi_i(\cdot|s)}[Q_{\pi_i}(s, a)] \geq E_{a \sim \pi_{TD3}(\cdot|s)}[Q_{TD3}(s, a)]$

Proof. For arbitrary  $s_t$

$$\begin{aligned}
& V_{\pi_{TD3}}(s_t) \\
&= E_{a_t \sim \pi_{TD3}(\cdot|s_t)}[Q_{\pi_{TD3}}(s_t, a_t)] \\
&\leq E_{a_t \sim \pi_i(\cdot|s_t)}[Q_{\pi_{TD3}}(s_t, a_t)] \\
&= E_{a_t \sim \pi_i(\cdot|s_t)}[r_t^{\pi_i} + \gamma E_{a_{t+1} \sim \pi_{TD3}(\cdot|s_{t+1})}[Q_{\pi_{TD3}}(s_{t+1}, a_{t+1})]] \\
&\leq E_{a_t \sim \pi_{TD3}(\cdot|s_t)}[r_t^{\pi_i} + \gamma E_{a_{t+1} \sim \pi_i(\cdot|s_{t+1})}[Q_{\pi_{TD3}}(s_{t+1}, a_{t+1})]] \\
&= E_{a_t \sim \pi_i(\cdot|s_t)}[r_t^{\pi_i} + \gamma r_{t+1}^{\pi_i} + \gamma^2 E_{a_{t+2} \sim \pi_{TD3}(\cdot|s_{t+2})}[Q_{\pi_{TD3}}(s_{t+2}, a_{t+2})]] \\
&\leq E_{a_t \sim \pi_i(\cdot|s_t)}[r_t^{\pi_i} + \gamma r_{t+1}^{\pi_i} + \gamma^2 E_{a_{t+2} \sim \pi_{TD3}(\cdot|s_{t+2})}[Q_{\pi_{TD3}}(s_{t+2}, a_{t+2})]] \\
&\dots
\end{aligned}$$

$$\begin{aligned}
&\leq E_{a_t \sim \pi_i(\cdot|s_t)}[r_t^{\pi_i} + \gamma r_{t+1}^{\pi_i} + \gamma^2 r_{t+2}^{\pi_i} + \dots + \gamma^\infty r_\infty^{\pi_i} + \dots] \\
&\cong E_{a_t \sim \pi_i(\cdot|s_t)}[\sum_{k=t}^{\infty} \gamma^{k-t} r_k^{\pi_i}] \\
&\cong E_{a \sim \pi_i(\cdot|s)}[Q_{\pi_i}(s, a)] \\
&= V_{\pi_i}(s_t)
\end{aligned}$$

From **Proposition1**, we assumed that having higher  $E_{a \sim \pi_i(\cdot|s)}[Q_{TD3}(s, a)]$  implies having higher  $E_{a \sim \pi_i(\cdot|s)}[Q_{\pi_i}(s, a)]$  among policy  $\pi_i$ 's, which means policy  $\pi_i$  with higher  $E_{a \sim \pi_i(\cdot|s)}[Q_{TD3}(s, a)]$  has higher probability to get better performance, compared to other policies with lower  $E_{a \sim \pi_i(\cdot|s)}[Q_{TD3}(s, a)]$ . Even we could not prove this assumption in mathematical way, but could add credibility to it by the numerical experiment, represented in Figure 6 of the main text. Finally, Referring the assumption, we developed a new sampling method, named Q-critic Filtering.

## Appendix D. Detailed Hyperparameters Setting

Table 1 represents the architecture of the neural networks for the actor and critic in our algorithm. Table 2 shows the fixed hyperparameters for all our tasks. Table 3 shows the other varying hyperparameters for each task.

Table 1: The architecture of the neural networks.

Actor	Critic
[state dim, 400]	[state dim + action dim, 400]
elu	elu
[400, 300]	[400, 300]
elu	elu
[300, action dim]	[300, 1]
tanh	-

Table 2: Hyperparameters constant across all tasks.

Hyperparameter	Value
Target weight ( $\tau$ )	0.005
TD3 Actor learning rate ( $lr_{actor}$ )	2e-3
TD3 Critic learning rate ( $lr_{critic}$ )	1e-3
Discount factor ( $\gamma$ )	0.99
Replay buffer size	1e-6
Batch size ( $\kappa$ )	256
CEM initial covariance ( $\Sigma_{init}$ )	7.5e-3
CEM limit covariance ( $\Sigma_{limit}$ )	1e-5
Initial distance weight ( $\beta_{init}$ )	1
Target distance ( $D_{target}$ )	0.05
Sampled states 2 ( $SS_2$ ) size	512
Q-filtering start step ( $T_{start-Q}$ )	150000
Multiple sample ratio ( $SR$ )	50
Sampled states 1 ( $SS_1$ ) size	64
Max interaction ( $T_{max}$ )	1000
Initial interaction steps ( $T_{init}$ )	400
Interval for increasing step ( $T_{inter}$ )	100000

Table 3: Hyperparameters varying across tasks.

Task	Swimmer	HalfCheetah	Hopper	Walker2d	Ant
Population size (N)	10	10	6	6	6
top K	5	5	3	3	3
Frequency of TD3 to CEM ( $F_{TD3 \rightarrow CEM}$ )	3	1	1	1	1
Decaying covariance constant ( $\tau_\Sigma$ )	0.01	0.03	0.03	0.03	0.03

## References

- [1] Achiam, J. (2018). Openai spinning up. *GitHub, GitHub repository*.
- [2] Fujimoto, S., Van Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*.
- [3] Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y. (2018). Stable baselines. <https://github.com/hill-a/stable-baselines>.
- [4] Pourchot, A. and Sigaud, O. (2018). Cem-rl: Combining evolutionary and gradient-based methods for policy search. *arXiv preprint arXiv:1810.01222*.
- [5] Uhlenbeck, G. E. and Ornstein, L. S. (1930). On the theory of the brownian motion. *Physical review*, 36(5):823.