A IMPLEMENTATION DETAILS FOR SD2PC

A.1 PSEUDO CODE OF ALGORITHM SD2PC

To eliminate the over-estimation problem of the critic network to improve the algorithm's performance, we introduced the double critic network presented in both TD3 and SAC to our SD2PC algorithm. In our practical SD2PC algorithm we parameterized θ_{Q1}, θ_{Q2} for a couple of critic networks, and $\theta'_{Q1}, \theta'_{Q2}$ for their target networks. These two networks have their independent training process but share a target value produced by minimum selection:

$$J^{Q}(\theta_{Qi}) = \Big[Q(s(t), a(t); \theta_{Qi}) - \big(r_t + \gamma \min_{j=\{1,2\}} Q(s(t), \widetilde{a}(t+1); \theta'_{Qj}) + \alpha\mathcal{H}\big)\Big]^2 \tag{23}$$

Which $i \in \{1, 2\}$, $\widetilde{a}(t+1)$ generated by equation 8. While evaluating critic values for policy optimization, we similarly use the minimum value between these two critic networks. The policy loss function based on the double critic network will come to:

$$J^{\pi}_{m,n}(\theta_\pi) = \pi_m(a^d_{m,n}|s(t);\theta_\pi)\Big(\alpha \log \pi_m(a^d_{m,n}|s(t);\theta_\pi) - \min_{j=\{1,2\}} \frac{1}{M} Q(s(t), \{a^d_{m,n}, \widetilde{\widetilde{a}}_m(t)\}; \theta_{Qj})\Big) \tag{24}$$

With the value and policy loss function in equation 23 and equation 24 replacing equation 10 and equation 13, we can give out the pseudo-code of our SD2PC algorithm with double critic networks, for $\lambda_\pi,\ \lambda_Q,\ \lambda_\alpha$ is the learning rate of the policy network, critic networks, and temperature α. $\{s(\mathcal{T}), a(\mathcal{T}), r_\mathcal{T}, s(\mathcal{T}+1)\}$ representing for the experiences sampled from the replay buffer:

Algorithm 1 Soft Stochastic Decomposed Discrete Policy-Critic (SD2PC)

Initialize network parameters θ_π, θ_{Q1}, θ_{Q2}, target network parameters θ'_π, θ'_{Q1}, θ'_{Q2}
Initialize replay buffer $\mathcal{B} \leftarrow \{\}$, batch size B, temperature α, hyperparameters $\lambda_\pi,\ \lambda_Q,\ \lambda_\alpha,\ \tau,\ \gamma$
Initialize target entropy $\overline{\mathcal{H}}$
  for $t = 0$ **to** T **do**
    Sample an action from policy distribution: $a(t) \sim \pi(a(t)|s(t);\theta_\pi)$
    Takes action $a(t)$, observes reward r_t and next state $s(t+1)$
    Store transition experience $\{s(t), a(t), r_t, s(t+1)\}$ in $\mathcal{B}$
    if Training **then**
      Sample a minibatch of B transitions $\{s(\mathcal{T}), a(\mathcal{T}), r_\mathcal{T}, s(\mathcal{T}+1)\}$ from $\mathcal{B}$
      Sample $a(\mathcal{T}+1)$ from $\pi(a(\mathcal{T}+1)|s(\mathcal{T}+1);\theta'_\pi)$
      $\theta_{Qi} \leftarrow \theta_{Qi} - \lambda_Q \nabla_{\theta_{Qi}} J^Q(\theta_{Qi})$ **For** $i \in \{1, 2\}$
      Sample $\widetilde{a}(\mathcal{T})$ from $\pi(\widetilde{a}(\mathcal{T})|s(\mathcal{T});\theta_\pi)$
      $\theta_\pi \leftarrow \theta_\pi - \lambda_\pi \nabla_{\theta_\pi} J^\pi(\theta_\pi)$
      $\alpha \leftarrow \alpha - \lambda_\alpha \nabla_\alpha J(\alpha)$
      $\theta'_\pi \leftarrow \tau\theta_\pi + (1-\tau)\theta'_\pi$
      $\theta'_{Q1} \leftarrow \tau\theta_{Q1} + (1-\tau)\theta'_{Q1}$
      $\theta'_{Q2} \leftarrow \tau\theta_{Q2} + (1-\tau)\theta'_{Q2}$
    end if
  end for

A.2 HYPERPARAMETERS FOR SD2PC

The critical parameters for SD2PC, which can significantly influence the algorithm's performance, are listed in Table 1 above. These hyperparameters include target entropy $\overline{\mathcal{H}}$, learning rates, and so on.

Table 1: Hyperparameters of SD2PC

Hyperparameter	Definition	Value
N	discrete actions per dimension	20
$\overline{\mathcal{H}}$	target entropy	-1.5
τ	stepsize of soft target update	5×10^{-3}
B	batch size	256
$\max(\log\ \alpha)$	temperature upper bound	2
$\min(\log\ \alpha)$	temperature lower bound	-10
λ_α	temperature learning rate	3×10^{-4}
λ_π	policy learning rate	3×10^{-4} (Humanoid-v2, Ant-v2) 10^{-3} (other environments)
λ_Q	critic learning rate	3×10^{-4} (Humanoid-v2, Ant-v2) 10^{-3} (other environments)

A.3 Network structure of algorithm SD2PC

As we pointed in Section 3, our decomposed discrete policy network outputs an $M \times N$ matrix, each row represents an action dimension's policy. However, it is difficult for a policy network to output a matrix straightly, so we let the policy network output a 1-dimensional array first, then reshape it to a 2-dimensional matrix. A detailed structure of the policy network can be shown in Figure 7.

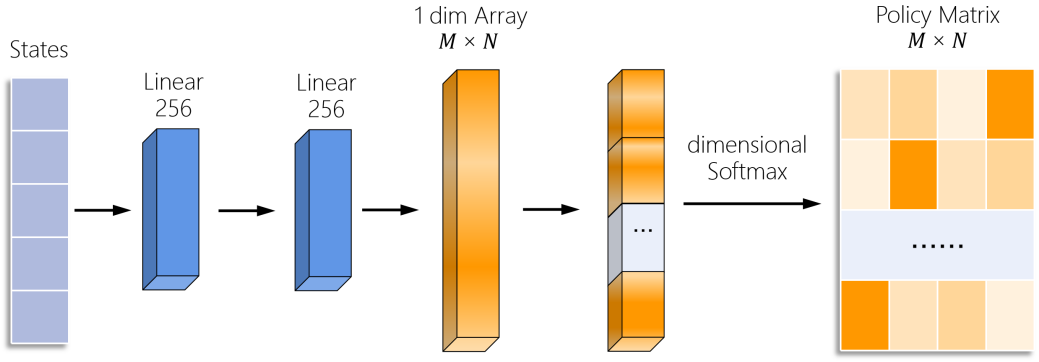

Figure 7: Details of SD2PC's decomposed discrete policy network

The structure of SD2PC's policy network is shown in Figure 8. Though the critic network represents a continuous state-action value function, we can only get the value of a single action point each time, represented by the dots on the critic value function presented in Figure 8. Though we need to forward the critic network for N times for each action dimension's policy update, the generated values are isolated from back-propagation. Moreover, the MN values needed by the global policy loss can be calculated simultaneously in a minibatch, so the computational complexity of the policy update of SD2PC is still reasonable.

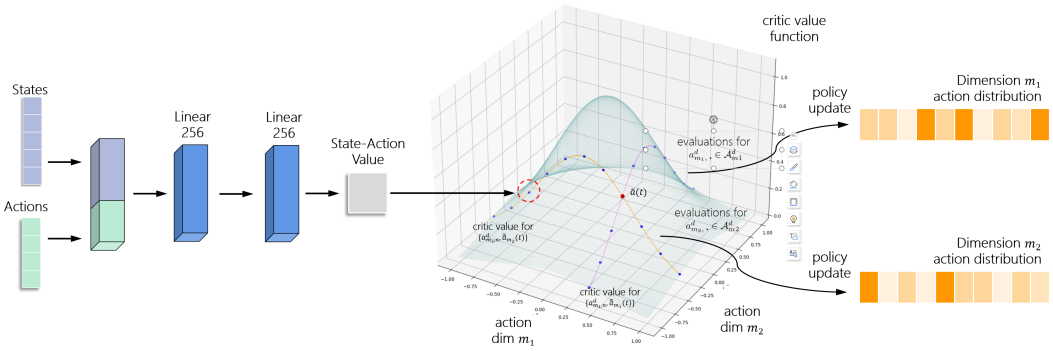

Figure 8: Details of SD2PC's critic network and SD2PC's policy update. N times of critic's forward propagation is needed for a single action dimension's policy update, and MN times is needed for the global policy's update but only consumes limited computational resources.

A.4 Action Probability Transfer of SD2PC

Consider the entropy presented in section 4. It's a global entropy composed of each action dimension's entropy $\mathcal{H}_m$:

$$\mathcal{H}_m = -\log \pi_m(\widetilde{a}_m(t+1)|s(t+1);\theta_\pi) \tag{25a}$$

$$\mathcal{H} = \sum_{m=1}^{M} \mathcal{H}_m. \tag{25b}$$

And the purpose of temperature adjustment in equation 14 is to fix the gap between the expectation of $\mathcal{H}_m$ and target entropy $\overline{\mathcal{H}}$ in each action dimension. In original settings without action probability transfer, the expectation of $\mathbb{E}(\mathcal{H}_i)$ is

$$\mathbb{E}(\mathcal{H}_m) = \sum_{n=1}^{N} -\pi_m(a^d_{m,n}|s(t);\theta_\pi)\log \pi_m(a^d_{m,n}|s(t);\theta_\pi) \tag{26}$$

The value of $\mathbb{E}(\mathcal{H}_m)$ represents the exploration rate of the discretized stochastic policy. However, consider two discretized policies with similar distribution but different action fractions N, their $\mathbb{E}(\mathcal{H}_m)$ can show distinctive diversities, which may bring difficulties turning $\overline{\mathcal{H}}$. For example, we set a continuous stochastic policy parameterized by Gaussian distribution with $\mu = 0, \sigma = 0.3$ and defined on $[-1, 1]$. If we transfer the continuous policy to discrete settings, we can get $\mathbb{E}(\mathcal{H}_m) = -3.78$ for $N = 100$, $\mathbb{E}(\mathcal{H}_m) = -3.09$ for $N = 50$ and $\mathbb{E}(\mathcal{H}_m) = -2.18$ for $N = 20$, which is showed in Figure 9.

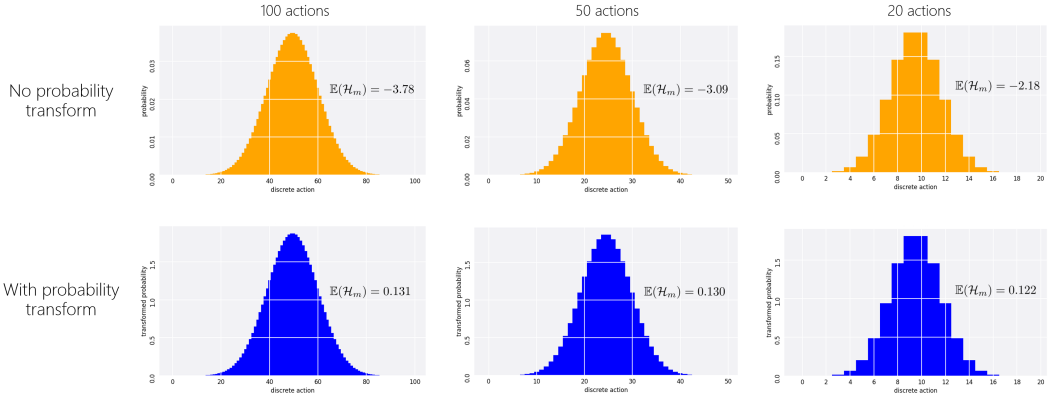

Figure 9: With probability transformation, $\mathbb{E}(\mathcal{H}_m)$ is less sensitive to discrete actions N

In order to address this problem, we mainly considered transforming the discrete policy distributions to probability densities on continuous action space. In most continuous control tasks, their actions are clipped to $[-1, 1]$. If we discretize an action dimension uniformly to N discrete actions, each discrete action represents a fraction with $2/N$ length on the continuous action space. Thus, we can transform the probability $\pi_m(a^d_{m,n}|s(t);\theta_\pi)$ to probability density $2\pi_m(a^d_{m,n}|s(t);\theta_\pi)/N$. With probability transfer, the expectation of $\mathbb{E}(\mathcal{H}_m)$ is

$$\mathbb{E}(\mathcal{H}_m) = \sum_{n=1}^{N} -\pi_m(a^d_{m,n}|s(t);\theta_\pi)\log\left[\frac{2}{N}\pi_m(a^d_{m,n}|s(t);\theta_\pi)\right] \tag{27}$$

Reconsidering the example above, with probability transform, we can get $\mathbb{E}(\mathcal{H}_m) = 0.131$ for $N = 100$, $\mathbb{E}(\mathcal{H}_m) = 0.130$ for $N = 50$ and $\mathbb{E}(\mathcal{H}_m) = 0.122$ for $N = 20$, which is stable to N. So, the probability transform we presented in SD2PC can effectively reduce the complexity of adjusting $\overline{\mathcal{H}}$, which made our algorithm less sensitive to hyperparameters.

A.5 KL DIVERGENCE POLICY LOSS FOR SD2PC

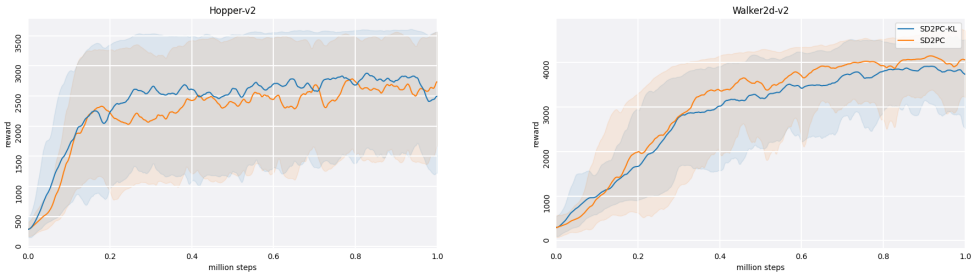

Figure 10: Experimental results of SD2PC-PG with policy loss equation 13 and SD2PC-KL with policy loss equation 30.

Expect the loss function in equation 12, KL divergence loss function, which shows off in soft RL methods like SQL Haarnoja et al. (2017) and both the first and the second edition of SAC Haarnoja et al. (2018a). Especially in SAC, they proposed to update the policy through replacing the old policy with a new policy π_{new}, which is based on KL divergence, and guaranteed on policy improvement compared with the old policy π:

$$\pi_{new} = \arg\min_{\pi\in\Pi} \mathrm{D}_{\mathrm{KL}}\left(\pi(\bullet\,|s(t))\middle\|\frac{\exp\big(Q(s(t),\bullet)\big)}{Z(s(t))}\right) \tag{28}$$

where $Z(s(t))$ is set to normalize the distribution. However, if the the policy delivers by neural networks, one can not replace π by π_new directly. So a policy loss function based on KL divergence is proposed in SAC's work replacing equation 28

$$J^\pi_K L(\theta_\pi) = \mathrm{D}_{\mathrm{KL}}\left(\pi(\bullet\,|s(t);\theta_\pi)\middle\|\frac{\exp\big(Q(s(t),\bullet;\theta_Q)\big)}{Z(s(t);\theta_Q)}\right). \tag{29}$$

In algorithms with continuous policies like SAC, the KL divergence in equation 29 can not be evaluated because we need to cover the entire continuous action space. So in soft RL algorithms with continuous actions, one can only exploit the practical policy loss function equation 3 to update the soft policy. In SD2PC with discrete policies, however, it is feasible to cover the entire action space $\mathcal{A}^d$ to evaluate the KL divergence policy loss. By the same consideration with equation 13 to reduce the computational complexity, we decouple the global KL divergence in equation 29 across action dimensions to a sum of the single-dimensional KL divergence policy loss functions, which can be evaluated by covering each dimension's discrete action sapce$\mathcal{A}^d_m$

$$J^\pi_{\mathrm{KL}}(\theta_\pi) = \frac{1}{M}\sum_{m=1}^{M} \mathrm{D}_{\mathrm{KL}}\left(\pi(\bullet\,|s(t);\theta_\pi)\middle\|\frac{\exp\left(\frac{1}{\alpha M}Q(s(t),\{\bullet,\widetilde{a}_m(t)\};\theta_Q)\right)}{Z(s(t),\widetilde{a}_m(t);\theta_Q)}\right) \tag{30}$$

To validate the effectiveness of the KL divergence policy loss function in equation 30 for SD2PC, we tested our SD2PC method incorporates equation 30 on the MuJoCo continuous control tasks. Experimental results in Figure 10 shows that our SD2PC with equation 30, which is marked as SD2PC-KL, performed equivalently to the SD2PC algorithm incorporates with policy loss function in equation 13. However, consider that equation 13 is easier to be realized, we insist on setting equation 13 to be the the policy loss function of our algorithm SD2PC.

B IMPLEMENTATION DETAILS FOR D3PC

B.1 EXPLORATION STRATEGY OF D3PC

In RL algorithms with discrete action space and deterministic policies like DQN, we can use ϵ-greedy to explore the action space. However, for continuous control tasks, a considerable ϵ can cause the high-frequency presence of extreme actions, which leads to unstable performances.

Mentioned that if we use RL algorithms with discrete action space to solve continuous control problems, we need to transform the output of the discrete action by the policy into continuous actions to interact with the environment. And for our D3PC algorithm, the transition experience $\{s(t), a(t), r_t, s(t+1)\}$ is used to train the critic network, which implies that the action $a(t)$ in the transition experience is continuous. So, in D3PC, we can use the Gaussian exploration strategy to first convert the discrete action into continuous action, and then add a Gaussian exploration noise to it.

However, during our evaluations, we found that with Gaussian Exploration Strategy only, D3PC may sometimes fall into local optimal in some baseline environments, such as Walker2d-v2, which may significantly influence the training efficiency of the algorithm. Considering that we can hardly explore the whole discrete action space through Gaussian exploration noise, in which some of the discrete actions far from the selected action is unreachable, we have designed a "ϵ-Gaussian exploration strategy hybridizing ϵ-greedy and Gaussian exploration strategy to explore the discrete action space effectively and stably.

The leading cause of unstable performances when ϵ-greedy apply to RL algorithms with discrete action space on continuous control tasks, is that they select a global policy scholastically instead of action-dimensional policies. In order to utilize ϵ-greedy stably, we apply ϵ-greedy independently to each action dimension, select a random action with probability ϵ. In that way, there will be only a few single-dimensional actions chosen randomly by ϵ-greedy in a single step, which can significantly eliminate the disturbances of extreme actions.

In our ϵ-Gaussian exploration strategy, after ϵ-greedy exploration, we applied a Gaussian exploration noise on the action of whether it is chosen by ϵ-greedy or not. We marked the exploration policy of D3PC with ϵ-Gaussian as $\pi_{\epsilon G}$, which is presented in D3PC's pseudo code. $\pi_{\epsilon G}$ can be expressed as follows:

$$a_m(t) = \pi_\epsilon\big(\mu(a(t)|s(t);\theta_d)\big) + \mathcal{N}(0,\sigma^2) \tag{31a}$$

$$a(t) = \{a_1(t), ..., a_m(t)\} \tag{31b}$$

in which π_ϵ represents the ϵ-greedy exploration strategy.

B.2 PSEUDO CODE OF ALGORITHM D3PC

Similar to SD2PC, we present double critic net work in D3PC to eliminate the over-estimation problem. With double critic networks and their parametersθ_{Q1}, θ_{Q2}, the value loss of D3PC becomes

$$J^Q(\theta_{Qi}) = \Big[Q(s(t), a(t); \theta_{Qi}) - \big(r_t + \gamma \min_{j=\{1,2\}} Q(s(t), \mu(s(t+1); \theta'_d); \theta'_{Qj})\big)\Big]^2 \tag{32}$$

And the deterministic policy loss under double critic networks

$$J^d_{m,n}(\theta_d) = \Big(Q_d(s(t), a^d_{m,n}; \theta_d) - \min_{j=\{1,2\}} Q(s(t), \{a^d_{m,n}, \overline{\mu}_m(s(t); \theta_d)\}; \theta_{Qj})\Big)^2 \tag{33}$$

With the value and policy loss function in equation 32 and equation 33 replacing equation 20 and equation 18, we can give out the pseudo code of double critic D3PC with $\lambda_d,\ \lambda_Q$ the learning rate of the decomposed Q network and the critic networks

Algorithm 2 Deterministic Decomposed Discrete Policy-Critic (D3PC)

Initialize network parameters θ_d, θ_{Q1}, θ_{Q2}, target network parameters θ'_d, θ'_{Q1}, θ'_{Q2}
Initialize replay buffer $\mathcal{B} \leftarrow \{\}$, batch size B, hyperparameters $\lambda_d,\ \lambda_Q,\ \tau,\ \gamma$
Initialize exploration rate σ, ϵ
  for $t = 0$ **to** T **do**
    Sample an action by ϵ-Gaussian: $a(t) \sim \pi_{\epsilon G}(a(t)|s(t);\theta_d)$
    Takes action $a(t)$, observes reward r_t and next state $s(t+1)$
    Store transition experience $\{s(t), a(t), r_t, s(t+1)\}$ in $\mathcal{B}$
    if Training **then**
      Sample a minibatch of B transitions $\{s(\mathcal{T}), a(\mathcal{T}), r_\mathcal{T}, s(\mathcal{T}+1)\}$ from $\mathcal{B}$
      $\theta_{Qi} \leftarrow \theta_{Qi} - \lambda_Q \nabla_{\theta_{Qi}} J^Q(\theta_{Qi})$ **For** $i \in \{1, 2\}$
      $\widetilde{a}_\mathcal{T} = \mu(\widetilde{a}_\mathcal{T}|s(\mathcal{T});\theta_d)$
      $\theta_d \leftarrow \theta_d - \lambda_d \nabla_{\theta_d} J^d(\theta_d)$
      $\theta'_d \leftarrow \tau\theta_d + (1-\tau)\theta'_d$
      $\theta'_{Q1} \leftarrow \tau\theta_{Q1} + (1-\tau)\theta'_{Q1}$
      $\theta'_{Q2} \leftarrow \tau\theta_{Q2} + (1-\tau)\theta'_{Q2}$
    end if
  end for

B.3 HYPERPARAMETERS OF D3PC

Table 2: Hyperparameters of D3PC

Hyperparameter	Definition	Value
N	discrete actions per dimension	20
ϵ	random exploration rate	0.05
σ	Gaussian exploration noise	0.05
τ	stepsize of soft target update	5×10^{-3}
B	batch size	256
λ_d	discrete Q learning rate	3×10^{-4} (Humanoid-v2, Ant-v2) 10^{-3} (other environments)
λ_Q	critic learning rate	3×10^{-4} (Humanoid-v2, Ant-v2) 10^{-3} (other environments)

B.4 NETWORK STRUCTURE OF D3PC

The structure of D3PC's critic network is the same as SD3PC's critic, which presents above in Figure 8. And for D3PC's decomposed Q network, its structure is similar to SD2PC's policy network in Figure 7 which have two hidden layers. The main distinction is that D3PC's policy matrix outputs straightly without softmax regularization because the discrete Q value is defined on $[-\infty, \infty]$.

C IMPLEMENTATION DETAILS FOR QPC

C.1 PSEUDO CODE OF QPC

With the introduction of double critic networks, Q_d and Q_p are presented in equation 21a are defined as follows

$$Q'_d = \min_{j=\{1,2\}} Q(s(t+1), \mu(s(t+1); \theta'_d); \theta'_{Qj}) \tag{34a}$$

$$Q'_p = \min_{j=\{1,2\}} Q(s(t+1), \mu^p(s(t+1), \theta'_p); \theta'_{Qj}). \tag{34b}$$

And the value loss function for $Q1$, $Q2$

$$J^Q(\theta_{Qi}) = \left[Q(s(t), a(t); \theta_{Qi}) - (r_t + \gamma Q'_{dp})\right]^2 \text{ for } i \in \{1, 2\} \tag{35}$$

In QPC, we apply the delayed policy updates in TD3 Fujimoto et al. (2018) to our continuous policy network, which updates θ_p once every h step. As we mentioned in Section 6 we set the initial steps of QPC to $T_\beta = 1 \times 10^5$ with β fixed to 1, and a decay rate $\beta^- = 1 - 5 \times 10^6$ to smoothly transform β to $\beta_{min} = 0.5$.

with equation 35 replacing equation 21b, pseudo-code of QPC is presented as follows, with λ_d, λ_p and λ_Q the learning rate of decomposed Q network, continuous policy network, and critic networks, and $J^p(\theta_p)$ the same format with equation 2

Algorithm 3 Q-policy-Critic (QPC)

Initialize network parameters θ_d, θ_p, θ_{Q1}, θ_{Q2}, target network parameters θ'_d, θ'_{Q1}, θ'_{Q2}, θ'_p
Initialize replay buffer $\mathcal{B} \leftarrow \{\}$, batch size B, hyperparameters $\lambda_d,\ \lambda_p,\ \lambda_Q,\ \tau,\ \gamma,\ h$
Initialize exploration rate σ, ϵ,
Initialize $\beta = 1, \beta^-, T_\beta$
for $t = 0$ **to** T **do**
  Sample action from Q network by ϵ-Gaussian: $a(t) \sim \pi_{\epsilon G}(a(t)|s(t); \theta_d)$
  Sample action from the continuous policy : $a^p(t) = \mu^p(s(t); \theta_p) + \mathcal{N}(0, \sigma^2)$
  $a(t) \leftarrow \beta a(t) + (1 - \beta)a(t)$
  Takes action $a(t)$, observes reward r_t and next state $s(t+1)$
  Store transition experience $\{s(t), a(t), r_t, s(t+1)\}$ in $\mathcal{B}$
  if Training **then**
    Sample a minibatch of B transitions $\{s(\mathcal{T}), a(\mathcal{T}), r_\mathcal{T}, s(\mathcal{T}+1)\}$ from $\mathcal{B}$
    Generate Q'_v, Q'_p
    $Q'_{vp} = \beta Q'_v + (1 - \beta)Q'_p$
    $\theta_{Qi} \leftarrow \theta_{Qi} - \lambda_Q \nabla_{\theta_{Qi}} J^Q(\theta_{Qi})$ **For** $i \in \{1, 2\}$
    $\widetilde{a}_\mathcal{T} = \mu(\widetilde{a}_\mathcal{T}|s(\mathcal{T}); \theta_d)$
    $\theta_d \leftarrow \theta_d - \lambda_d \nabla_{\theta_d} J^d(\theta_d)$
    if t **mod** h **then**
      $\theta_p \leftarrow \theta_p - \lambda_p \nabla_{\theta_p} J^p(\theta_p)$
      $\theta'_p \leftarrow \tau\theta_p + (1 - \tau)\theta'_p$
    end if
    $\theta'_{Q1} \leftarrow \tau\theta_{Q1} + (1 - \tau)\theta'_{Q1}$
    $\theta'_{Q2} \leftarrow \tau\theta_{Q2} + (1 - \tau)\theta'_{Q2}$
    if t > T_β **then**
      $\beta \leftarrow (\beta - \beta_{min}) \cdot \beta^- + \beta_{min}$
    end if
  end if
end for

C.2 HYPERPARAMETERS OF QPC

Table 3: Hyperparameters for QPC

Hyperparameter	Definition	Value
N	discrete actions per dimension	20
ϵ	random exploration rate	0.05
σ	Gaussian exploration noise	0.05
τ	stepsize of soft target update	5×10^{-3}
B	batch size	256
T_β	initial steps	2×10^5
β^-	decay rate of β	$1 - 5 \times 10^{-6}$
β_{min}	minimum value of β	0.5
h	policy delay	2
λ_d	decomposed Q learning rate	3×10^{-4} (Humanoid-v2, Ant-v2) 10^{-3} (other environments)
λ_p	continuous policy learning rate	3×10^{-4} (Humanoid-v2, Ant-v2) 10^{-3} (other environments)
λ_Q	critic learning rate	3×10^{-4} (Humanoid-v2, Ant-v2) 10^{-3} (other environments)

C.3 DETAILS OF THE CONCEPTUAL ALGORITHMS

Algorithm 4 conceptual Algorithm D3PC-CA

Initialize network parameters θ_d, θ_p, θ_{Q1}, θ_{Q2}, target network parameters θ'_d, θ'_{Q1}, θ'_{Q2}, θ'_p
Initialize replay buffer $\mathcal{B} \leftarrow \{\}$, batch size B, hyperparameters $\lambda_d,\ \lambda_p,\ \lambda_Q,\ \tau,\ \gamma,\ h$
Initialize exploration rate σ, ϵ,
  for $t = 0$ **to** T **do**
    if Evaluation **then**
      Sample action from the continuous policy : $a(t) = \mu^p(s(t); \theta_p)$
    else
      Sample action from Q network by ϵ-Gaussian: $a(t) = \pi_{\epsilon G}(a(t)|s(t); \theta_d) + \mathcal{N}(0, \sigma^2)$
    end if
    Takes action $a(t)$, observes reward r_t and next state $s(t+1)$
    Store transition experience $\{s(t), a(t), r_t, s(t+1)\}$ in $\mathcal{B}$
    if Training **then**
      Sample a minibatch of B transitions $\{s(\mathcal{T}), a(\mathcal{T}), r_\mathcal{T}, s(\mathcal{T}+1)\}$ from $\mathcal{B}$
      $\theta_{Qi} \leftarrow \theta_{Qi} - \lambda_Q \nabla_{\theta_{Qi}} J^Q(\theta_{Qi})$ **For** $i \in \{1, 2\}$
      $\widetilde{a}_\mathcal{T} = \mu(\widetilde{a}_\mathcal{T}|s(\mathcal{T}); \theta_d)$
      $\theta_d \leftarrow \theta_d - \lambda_d \nabla_{\theta_d} J^d(\theta_d)$
      if t **mod** h **then**
        $\theta_p \leftarrow \theta_p - \lambda_p \nabla_{\theta_p} J^p(\theta_p)$
        $\theta'_p \leftarrow \tau\theta_p + (1-\tau)\theta'_p$
      end if
      $\theta'_d \leftarrow \tau\theta_d + (1-\tau)\theta'_d$
      $\theta'_{Q1} \leftarrow \tau\theta_{Q1} + (1-\tau)\theta'_{Q1}$
      $\theta'_{Q2} \leftarrow \tau\theta_{Q2} + (1-\tau)\theta'_{Q2}$
    end if
  end for

In Section 6, we presents two conceptual algorithms: D3PC-continuous actor(D3PC-CA) and TD3-decomposed Q net (TD3-DQ) to explore the possibility of using D3PC's critic network to fit a continuous policy or utilizing TD3's critic network to train a decomposed discrete policy. Essentially, D3PC-CA can be regarded as QPC with $\beta = 0$ and presents the continuous policy during evaluation. We give out D3PC-CA'S pseudo-code in Algorithm 4.

For TD3-DQ, it can be regarded as QPC with $\beta = 1$, or TD3 with a decomposed Q network training alone, optimizing itself with TD3's critic providing evaluations. To give credible results, we present TD3's target policy smoothness in TD3-DQ. TD3-DQ's pseudo-code is in Algorithm 5.

Algorithm 5 conceptual Algorithm TD3-DQ

Initialize network parameters θ_d, θ_p, θ_{Q1}, θ_{Q2}, target network parameters θ'_d, θ'_{Q1}, θ'_{Q2},θ'_p
Initialize replay buffer $\mathcal{B} \leftarrow \{\}$, batch size B, hyperparameters $\lambda_d,\ \lambda_p,\ \lambda_Q,\ \tau,\ \gamma,\ h$
Initialize Gaussian exploration noise σ, target policy smoothness σ'
  for $t = 0$ **to** T **do**
    if Evaluation **then**
      Gets an action from the Q network: $a(t) = \mu(a(t)|s(t);\theta_d)$
    else
      Gets an action from the continuous policy : $a(t) = \mu^p(s(t);\theta_p) + \mathcal{N}(0,\sigma^2)$
    end if
    Takes action $a(t)$, observes reward r_t and next state $s(t+1)$
    Store transition experiences $\{s(t), a(t), r_t, s(t+1)\}$ in $\mathcal{B}$
    if Training **then**
      Sample a minibatch of B transitions $\{s(\mathcal{T}), a(\mathcal{T}), r_\mathcal{T}, s(\mathcal{T}+1)\}$ from $\mathcal{B}$
      Get the target action $a(\mathcal{T}+\infty) = \mu^p(s(\mathcal{T});\theta_p) + \mathcal{N}(0,\sigma'^2)$
      $\theta_{Qi} \leftarrow \theta_{Qi} - \lambda_Q \nabla_{\theta_{Qi}} J^Q(\theta_{Qi})$ **For** $i \in \{1,2\}$
      $\widetilde{a}_\mathcal{T} = \mu(\widetilde{a}(\mathcal{T})|s(\mathcal{T});\theta_d)$
      $\theta_d \leftarrow \theta_d - \lambda_d \nabla_{\theta_d} J^d(\theta_d)$
      if t **mod** h **then**
        $\theta_p \leftarrow \theta_p - \lambda_p \nabla_{\theta_p} J^p(\theta_p)$
        $\theta'_p \leftarrow \tau\theta_p + (1-\tau)\theta'_p$
      end if
      $\theta'_d \leftarrow \tau\theta_d + (1-\tau)\theta'_d$
      $\theta'_{Q1} \leftarrow \tau\theta_{Q1} + (1-\tau)\theta'_{Q1}$
      $\theta'_{Q2} \leftarrow \tau\theta_{Q2} + (1-\tau)\theta'_{Q2}$
    end if
  end for

D CONTINUOUS CONTROL TASKS BASED ON MUJOCO ENVIRONMENTS

The continuous control tasks, which are utilized to evaluate our algorithms, are based on the MuJoCo physics engine and OpenAI Gym benchmark suite. These tasks are summarized in Fig.11, which both own high-dimensional continuous state space, like humanoid-v2 with 376 continuous states and Ant-v2 with 111 continuous states. As to the actions, some of these tasks own high-dimensional continuous action space, which is set to evaluate the effectiveness of our algorithms in handling high-dimensional continuous actions. For some other tasks like InvertedDoublePendulum-v2 with only one continuous action dimension, evaluations of our proposed algorithms in these environments proved that our algorithms perform well in simple environments. Attributes of all the six continuous control tasks are listed in Table.4.

Table 4: Attributes of the baseline environments

Environment	State dimension	Action dimension	with termination
Hopper-v2	11	3	True
Walker2d-v2	17	6	True
HalfCheetah-v2	17	6	False
Ant-v2	111	8	True
Humanoid-v2	376	17	True
InvertedDoublePendulum-v2	11	1	True

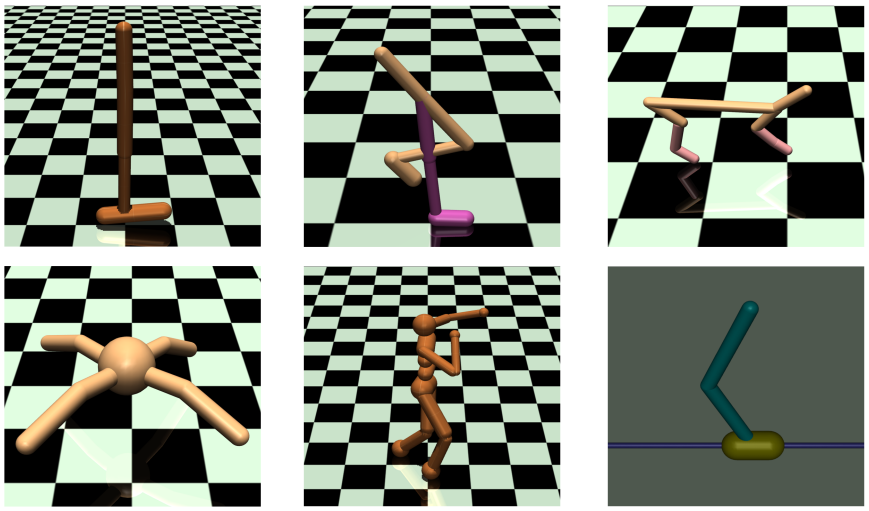

Figure 11: MuJoCo continuous control tasks. Top: Hopper-v2, Walker2d-v2, HalfCheetah-v2; Bottom: Ant-v2, Humanoid-v2, InvertedDoublePendulum-v2

E ADDITIONAL EXPERIMENTS

E.1 STUDIES ON THE ACCURACY OF ACTION DISCRETIZATION

In our algorithms with decomposed discrete policies, we need to discretize each continuous action space into N discrete actions before the training process. However, the quantity of N may influence the algorithm's training efficiency. In this section, we evaluated our algorithm SD2PC on several baseline environments with $N = 10, 20$, or 50. Experimental results show that higher N may lead to more effective performances. In contrast, if a small N presents in our algorithms which indicates the action discretization accuracy is low, the algorithm may be failed to update an effective policy. Our algorithms, including SD2PC and D3PC, can inherit some relatively high N and not lead to training failures. However, if our algorithms incorporate an excessive N, the policy network that owns MN output neurons may become hard to train. What's more, higher N indicates higher computational complexity.

So, during our evaluations, we usually set N to 20. Evaluation of different action discretization levels N is showed in Figure 12.

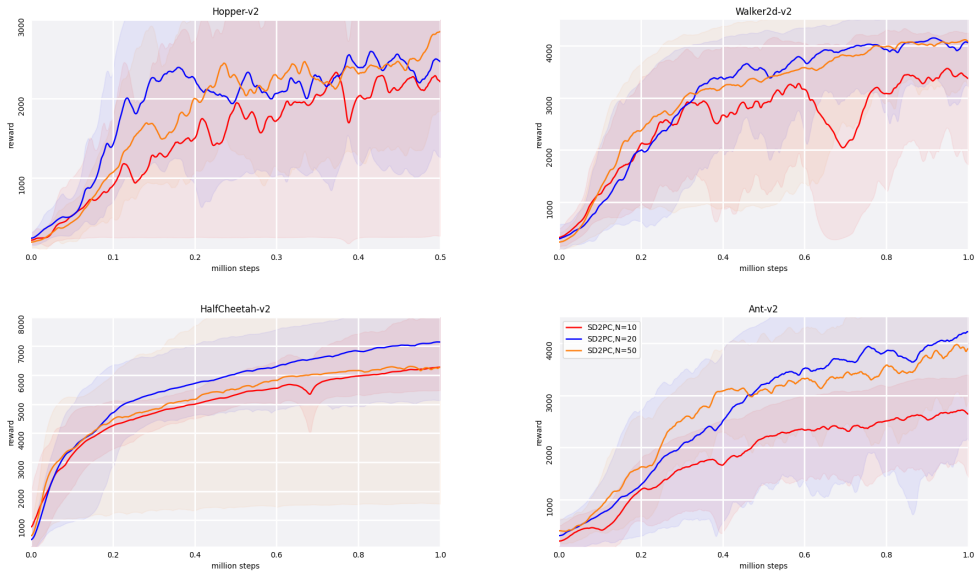

Figure 12: Results for different action discretization levels N.

E.2 STUDIES ON SD2PC'S TARGET ENTROPY

In our algorithms with decomposed discrete policies, we need to discretize each continuous action space into N discrete actions before the training process. However, the quantity of N may influence the algorithm's training efficiency. In this section, we evaluated our algorithm SD2PC on several baseline environments with $N = 10, 20$, or 50. Experimental results show that higher N may lead to more effective performances. In contrast, if a small N presents in our algorithms which indicates the action discretization accuracy is low, the algorithm may be failed to update an effective policy. Our algorithms, including SD2PC and D3PC, can inherit some relatively high N and not lead to training failures. However, if our algorithms incorporate an excessive N, the policy network that owns MN output neurons may become hard to train. What's more, higher N indicates higher computational complexity.

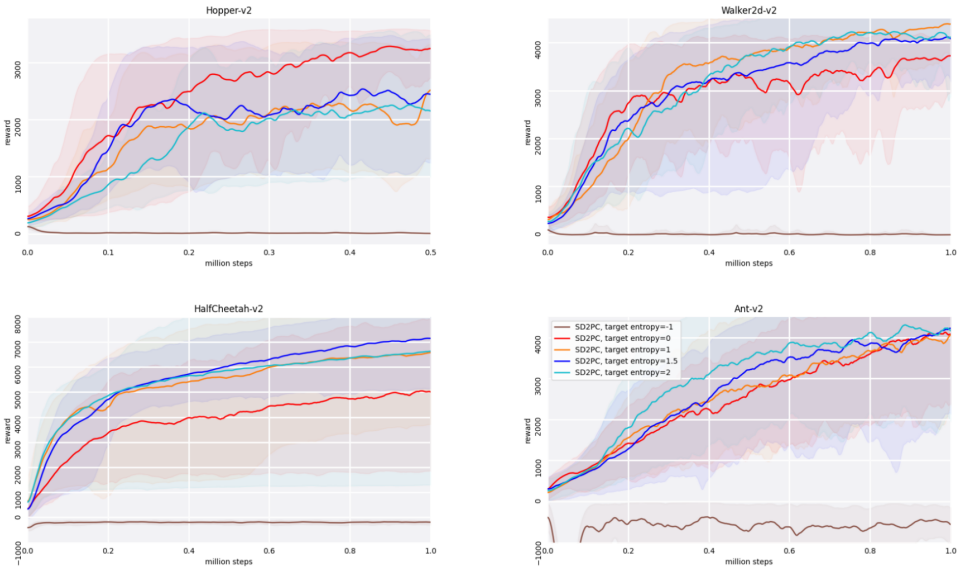

Figure 13: Results for different target entropy $\overline{\mathcal{H}}$ in SD2PC.

E.3 ABLATION STUDIES ON THE EXPLORATION STRATEGY OF D3PC

In our algorithm D3PC, we present a ϵ-Gaussian hybrid strategy, which associates ϵ-greedy and Gaussian exploration noise to explore the environment. In this section, we set an ablation study comparing different exploration strategies of D3PC:

ϵ-greedy exploration, with only an $\epsilon = 0.1$ to explore the action space

Gaussian exploration strategy, setting a Gaussian exploration noise chich $\mu = 0$ and $\sigma = 0.1$

ϵ-Gaussian exploration strategy, which $\epsilon = 0.05$ and $\sigma = 0.05$

Training curves of different exploration strategies are shown in Figure 14. Although ϵ-greedy or Gaussian exploration strategy works well independently in some environments, in other environments, they may have drawbacks leading to low training efficiency or local optimal problems. In our algorithm D3PC, we set ϵ-Gaussian as the exploration strategy. It works well in all of the six continuous control tasks.

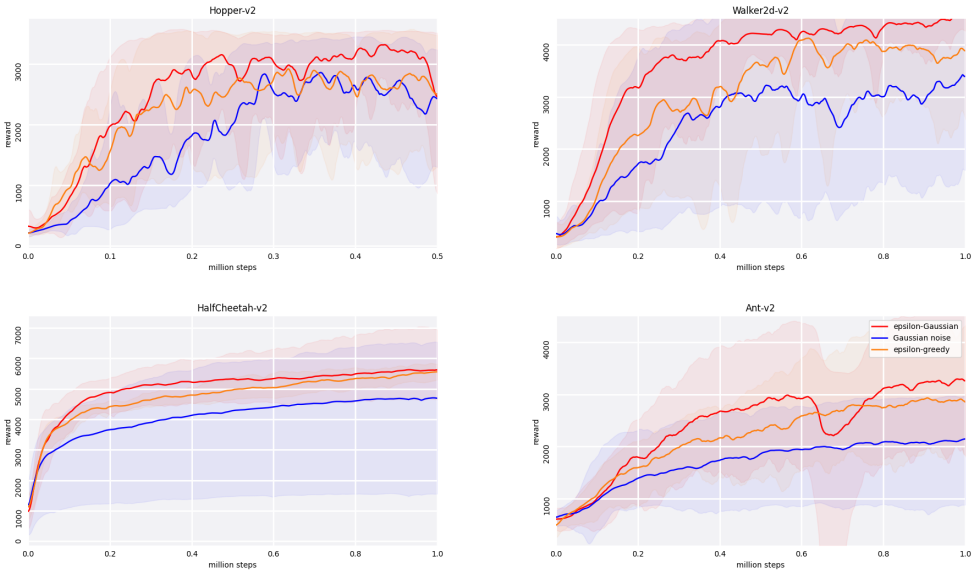

Figure 14: Results for different exploration strategies associated with D3PC: ϵ-greedy with $\epsilon = 0.1$, Gaussian exploration strategy with $\sigma = 0.1$, and ϵ-Gaussian exploration strategy with $\epsilon = 0.05$ and $\sigma = 0.05$.

E.4 SEPERATED EVALUATIONS OF ALGORITHMS WITH STOCHASTIC OR DETERMINISTIC POLICIES

In this section, we present the baseline evaluation with separation, which compares the deterministic algorithms, including DDPG, TD3, D3PC, QPC; and the stochastic algorithms, including SAC, and SD2PC, respectively. Results in Fig. 15 indicate our SD2PC performs similarly to SAC on Humanoid-v2 but better than SAC in the other tasks. As for the deterministic algorithms, Fig. 16 indicates our QPC performs better than TD3 in all of the six benchmark tasks.

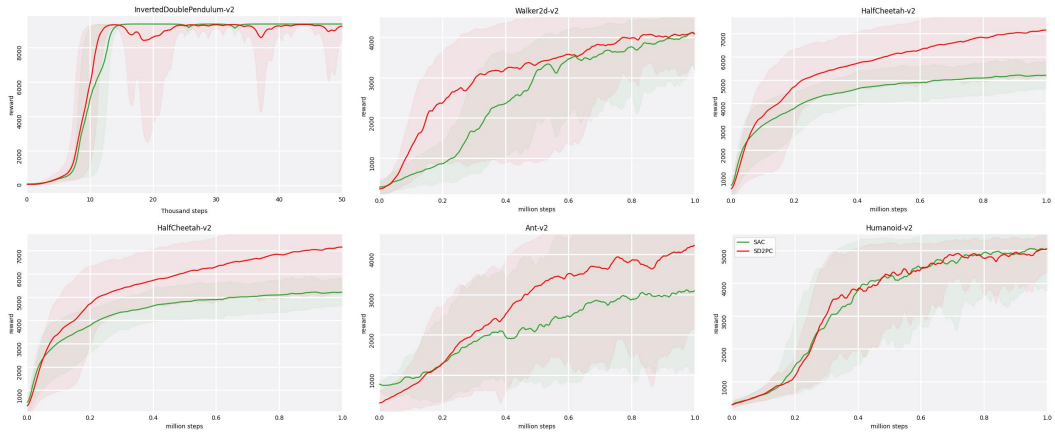

Figure 15: Training curves of the stochastic algorithms, which are averaged over five random seeds.

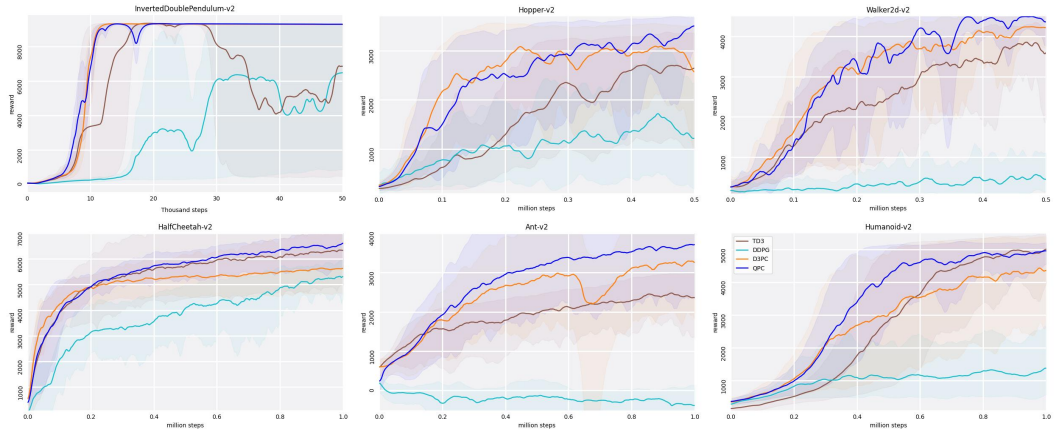

Figure 16: Training curves of the deterministic algorithms, which are averaged over five random seeds.