MEPG: A Minimalist Ensemble Policy Gradient Framework for Deep Reinforcement Learning

Qiang He[1,2], Chen Gong[1,2], Yuxun Qu[1,2], Xiaoyu Chen[1], Xinwen Hou[1], and Yu Liu[1]

[1] Institute of Automation, Chinese Academy of Sciences
[2] School of Artificial Intelligence, University of Chinese Academy of Sciences
{heqiang2019,gongchen2020,quyuxun2019,xiaoyu.chen,xinwen.hou,yu.liu}@ia.ac.cn

Abstract. Ensemble reinforcement learning (RL) aims to mitigate instability in Q-learning and to learn a robust policy, which introduces multiple value and policy functions. In this paper, we consider finding a novel but simple ensemble Deep RL algorithm to solve the resource consumption issue. Specifically, we consider integrating multiple models into a single model. To this end, we propose the Minimalist Ensemble Policy Gradient framework (MEPG), which introduces minimalist ensemble consistent Bellman update. And we find one value network is sufficient in our framework. Moreover, we theoretically show that the policy evaluation phase in the MEPG is mathematically equivalent to a deep Gaussian Process. To verify the effectiveness of the MEPG framework, we conduct experiments on the gym simulator, which show that the MEPG framework matches or outperforms the state-of-the-art ensemble methods and model-free methods without additional computational resource costs.

1 Introduction

One of the main driving factors for the success of the current deep learning research paradigm is that intelligent systems can obtain open-world perceptions from large amounts of data [1,2] through high-capacity function approximators (neural networks, for instance). Deep reinforcement learning (DRL) follows this research paradigm, using neural networks for function approximation to solve sequential decision-making problems, and has achieved great success in a wide range of domains such as board games [3], video games [4,5], robot manipulation [6], etc.

However, it remains a challenging problem to create a DRL agent which maintains high sample efficiency, robustness, and low computing resource consumption [7,8]. There are considerable factors to influence the training and optimization process of DRL algorithms, such as high variance, sensitivity to hyper-parameters, high computational resource consumption but sample inefficiency caused by sample demand, and instability of training induced by the randomness of the learning process and environment. Some DRL algorithms attempt to tackle these issues from the perspective of ensemble methods, i.e., introducing more value functions, or more policy functions [9,10,11,12,13,14]. The ensemble methods in DRL work well when the models are sufficiently rich in diversity, which is tricky because the policy and value function would grow increasingly similar as training proceeds. Furthermore, the training of ensemble methods in DRL requires

multiples computing resources, introduces more hyper-parameters, and requires more non-algorithmic level tricks and subtle fine-tuning for hyper-parameters. Due to these requirements, there are still some tricky issues to be addressed when applying ensemble methods to DRL algorithms in practice compared to current model-free DRL methods.

We ask: can we find a novel but simple approach to ensemble DRL algorithms to solve the resource consumption problem? We find one network might be enough. In this paper, we propose the Minimalist Ensemble Policy Gradient framework (MEPG) to deal with the aforementioned issues of ensemble methods in DRL. The source of our insight is that ensemble methods can be achieved by integrating multiple models into a single model. In this way, the issue of heavy resource consumption of ensemble learning methods can be well addressed, as one single model does not consume more computational resources than model-free deep reinforcement learning algorithms. In addition, this framework introduces only one additional hyper-parameter on modern model-free DRL algorithms. We implement our insight by minimalist ensemble consistent Bellman update that utilizes dropout [15], which is widely used in deep learning but not in DRL. Each dropout operator acting on the neural networks is equivalent to a new sub-model (or sub-network). The complete model without the dropout operator is equivalent to all the sub-networks acting together, i.e., integrating all the models. However, applying the dropout operator directly to the value functions creates the problem of source-target mismatch, i.e., the left and right sides of the Bellman equation do not correspond to the same value function. Therefore, we introduce minimalist ensemble consistent Bellman update where the same dropout operator acts on both sides of the Bellman equation at the same time. Furthermore, we show theoretically that the policy evaluation of MEPG framework is equivalent to a deep Gaussian Process [16]. We apply our MEPG framework to simple DRL algorithms, DDPG [17], and SAC [6]. And our experimental results show that our algorithm matches or outperforms state-of-the-art ensemble DRL and model-free DRL algorithms. We highlight that our framework does not introduce any auxiliary loss function or additional computational consumption compared to the basic algorithm. Moreover, the parameters of MEPG are much less than those of the modern model-free DRL algorithms. We highlight the application of deep neural network techniques that may contribute to reinforcement learning, which is a new view to improve DRL algorithms.

Our contributions are summarized as follows. Firstly, we propose a general ensemble reinforcement learning framework, called MEPG, which is simple and easy to implement. This framework, without introducing any additional loss functions, solves the problem that ensemble reinforcement learning methods consume a large amount of computation resources. Secondly, we provide theoretical analysis for this framework, where the policy evaluation process in the MEPG framework is mathematically equivalent to a deep Gaussian Process. Thirdly, we experimentally demonstrate the effectiveness of the MEPG framework by combining our algorithm with the simple DDPG and SAC algorithms. The results show that the MEPG framework achieves or exceeds state-of-the-art ensemble method DRL and model-free DRL methods with less computational resources. Moreover, This framework can be combined with any DRL algorithm.

2 Related Work

We now briefly discuss off-policy DRL algorithms and ensemble DRL methods, then the dropout operator in DRL algorithms. And we compare these works with our framework.

2.1 Off-policy DRL algorithms

There is a clear trail for the development of off-policy Deep RL. Deep Q-networks (DQN) [18] is the first successful off-policy deep RL algorithm to be applied to video games. It has long been recognized that the overestimation in Q-learning could severely impair the performance [19]. Double Q-learning [20,21] is proposed to solve overestimation issue for discrete action space. The overestimation issue still exists in continuous control algorithms such as Deterministic Policy Gradient (DPG) [22] and its deep variant Deep Deterministic Policy Gradient (DDPG) [17]. [23] introduced clipped double Q-learning (CDQ), which decreases the overestimation issue in DDPG. [6] proposed Soft Actor Critic (SAC) algorithm which is based on maximum entropy reinforcement learning framework [24] and combined with CDQ, resulting in a stronger algorithm. However, the CDQ approach introduces a slight underestimation issue [25,26]. We applied the MEPG framework to DDPG and SAC algorithm, which achieves or surpasses the performance of compared methods even without introducing a technique that getting a precise value function.

2.2 Ensemble DRL algorithms

Ensemble methods, a popular idea of machine learning, use multiple learning algorithms to obtain better performance than that could be obtained from arbitrarily composed algorithms. And ensemble methods are also used in DRL [9,10,11] for different purposes. [12] showed that modeling error can be reduced by ensemble methods in model-based DRL. [25,27] tackled the estimation issue and gave unbiased estimation methods of the value function from the perspective of ensemble functions. [28] proposed Random Ensemble Mixture, which introduces a convex combination of multiple Q-networks to approximate the optimal Q-function. [13] proposed a method for model training and selection in a single run. [14] proposed SUNRISE framework that uses a ensemble-based weighted Bellman backups and UCB exploration [29]. We are inspired by ensemble learning. But unlike these methods, we only use a single model instead of multiple models to achieve our ensemble purpose.

2.3 Dropout in DRL algorithms

It was recognized early on that dropout [15] can improve the performance of DRL algorithms in video games [30,31]. Because the dropout operator prevents complex co-adaptations of units in neural networks where a feature detector is only helpful in the context of several other feature detectors. Therefore, the effectiveness of the dropout in pixel-level input DRL is somehow similar to the supervised learning tasks for image input. [32] proposed a privileged information dropout (PI-Dropout) method

to improve sample efficiency and performance, but requires prior knowledge, i.e., an optimal representation for inputs. Our work does not require any auxiliary information. [33,34] obtain the neural networks model uncertainty with dropout through multiple forwards given the same inputs. The closest work to ours is DQN+dropout [35] in a discrete environment, which also utilizes the uncertainty of Q-function induced by dropout. Another approach to measuring uncertainty is to determine it using the variance generated by multiple Q-functions [14]. Our approach differs from previous works in that we neither use multiple Q-functions to estimate model uncertainty nor prevent pixel-level feature co-adaptation. We use the model ensemble property introduced by the dropout operator.

3 Background

We consider standard RL paradigm as a Markov Decision-making Process (MDP), which is characterized by a 6-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \rho_0, \gamma)$, i.e., a state space $\mathcal{S}$, an action space $\mathcal{A}$, a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, a transition probability $P(s_{t+1} \mid s_t, a_t)$ - specifying the probability of transitioning from state s_t to s_{t+1} given action a_t, an initial state distribution ρ_0, and a discount factor $\gamma \in [0, 1)$ [36]. The agent learns a policy, stochastic or deterministic from interacting with environment. At each time step, the agent generates action a w.r.t. policy π based on current state s and send a to environment. Then the agent receives a reward signal r and a new state s' from environment. Through multiple interactions, a trajectory $\tau = \{s_0, a_0, s_1, a_1, \cdots\}$ is generated. The optimization goal of RL is to maximize the expected cumulative discounted reward $J = \mathbb{E}_\tau[R_0]$, where $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$. Two functions play important roles in RL, the state value function $V^\pi(s) = \mathbb{E}_\tau[R_0|s_0 = s]$ and action value function (Q-function), $Q^\pi(s, a) = \mathbb{E}_\tau[R_0|s_0 = s, a_0 = a]$, a.k.a. critic, which represent how good a state is and how good a specific action is respectively. According to Bellman Equation [37], action value function satisfies

$$Q^\pi(s, a) = r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a), a' \sim \pi(\cdot|s')}[Q^\pi(s', a')]. \tag{1}$$

In Q-learning, the value function can be learned with Equation (1) style [36].

3.1 Deterministic Policy Gradient

For a large state space $\mathcal{S}$, we can utilize neural networks to represent the corresponding policy and Q-function. It is necessary to introduce policy gradient theorem [38] to learn a policy w.r.t. its value function. In actor critic style DRL algorithms, the policy can be updated by Deterministic Policy Gradient [22,17]

$$\nabla_\phi J^\pi_{\text{DDPG}}(\phi) = \mathbb{E}_{s \sim P_\pi}[\nabla_a Q^\pi(s, a; \theta)|_{a=\pi(s;\phi)} \nabla_\phi \pi(s; \phi)], \tag{2}$$

where the actor π and critic Q are parameterized by ϕ and θ respectively. The Q network is updated by temporal difference learning with a frozen auxiliary target network $Q^\pi(s, a; \theta')$

$$\begin{aligned} J^Q_{\text{DDPG}}(\theta) =& \mathbb{E}_{(s,a)\sim\mathcal{B}}\Big[\frac{1}{2}\Big(Q(s, a; \theta) \\ & - \big(r(s, a) + \gamma \mathbb{E}_{a' \sim \pi(s';\phi')}[Q(s', a'; \theta')]\big)\Big)^2\Big], \end{aligned} \tag{3}$$

where $\mathcal{B}$ is a replay buffer, storing some transition tuples (s, a, r, s'). The target network is updated by $\theta' \leftarrow \eta\theta + (1-\eta)\theta'$ at each time step, where η is a small constant. In actor critic style algorithm, the critic is usually used to evaluate how good the current policy π is. Thus how to learn a precise critic is called policy evaluation. Accordingly, how to make a policy better is called policy improvement. [23] introduced the TD3 algorithm, which takes the minimum value of a couple of critics as the target of the Bellman update.

3.2 Soft Actor Critic

[6] introduced Soft Actor Critic (SAC) algorithm based on maximum entropy RL framework [24], which encourages exploration by adding policy entropy to optimization objective. The policy optimization objective is

$$J^{\pi}_{\mathrm{SAC}}(\phi) = \mathbb{E}_{s\sim\mathcal{B}}[\mathbb{E}_{a\sim\pi(\cdot|s,\phi)}[\alpha\log(\pi(a \mid s;\phi)) - Q(s,a;\theta)]]. \tag{4}$$

The soft Q-function can be obtained by minimizing the soft Bellman residual

$$\begin{aligned} J^{Q}_{\mathrm{SAC}}(\theta) = \mathbb{E}_{(s,a)\sim\mathcal{B}}\Big[\frac{1}{2}\Big(&Q(s,a;\theta) \\ &- \big(r(s,a) + \gamma\mathbb{E}_{s'\sim P}[V(s';\theta')]\big)\Big)^2\Big], \end{aligned} \tag{5}$$

where $V(s;\theta') = \mathbb{E}_{a\sim\pi(\cdot|s;\phi)}[Q(s,a;\theta') - \alpha\log\pi(a|s;\phi)]$. The temperature parameter α can be given as a hyper-parameter or learned by minimizing

$$J^{\alpha}_{\mathrm{SAC}}(\alpha) = \mathbb{E}_{a\sim\pi^*}[-\alpha\log\pi^*(a|s;\alpha,\phi) - \alpha\mathcal{H}], \tag{6}$$

where $\mathcal{H}$ is pre-given target entropy.

4 Minimalist Ensemble Policy Gradient

In this section, we firstly propose minimalist ensemble consistent Bellman update. Then we propose the minimalist ensemble policy gradient (MEPG) framework. Secondly, we apply MEPG framework to model-free off-policy DRL algorithm, which integrates multiple value functions into a single model without introducing any loss function and auxiliary tasks. In principle, our framework can be applied to any modern DRL algorithms. In this paper, We deploy the MEPG framework in the popular DRL method DDPG and vanilla SAC. Thirdly, we show that the policy evaluation process in the MEPG framework is mathematically equivalent to a deep Gaussian Process.

4.1 Minimalist ensemble consistent Bellman update

It is intractable to integrate too many models for ensemble DRL due to limited computation resources. Thus, we consider integrating as many value functions as possible into a single model. To this end, we consider utilizing the shared parameters to achieve our purpose. We deploy the dropout operator [15] in our framework. Each dropout operator acting on the neural networks is equivalent to a sub-model (or sub-network).

The complete model without the dropout operator is equivalent to all the sub-networks acting together, i.e., integrating all the models. Thus, the ensemble property holds. A feed-forward operation of a standard neural network for layer l and hidden unit i can be described as

$$z_i^{l+1} = \mathbf{w}_i^{l+1}\mathbf{x}^l + b_i^{l+1}, \quad x_i^{l+1} = f(z_i^{l+1}), \tag{7}$$

where f is a activation function, and $\mathbf{w}$ and b represent the weights and bias respectively. We adopt the following dropout feed-forward style operation

$$\begin{aligned} m_j^l &\sim \text{Bernoulli}(1-p), \quad \tilde{\mathbf{x}}^l = \mathbf{m}^l \odot \mathbf{x}^l, \\ z_i^{l+1} &= f\big(\frac{1}{1-p}(\mathbf{w}_i^{l+1}\tilde{\mathbf{x}}^l + b_i^{l+1})\big), \end{aligned} \tag{8}$$

where p is the probability of an element to be set to zero and $\odot$ represents Hadamard product. The scale factor $\frac{1}{1-p}$ is added to ensure that the expected output from each unit would keep the same as the one without the dropout operator $\mathcal{D}_{\mathbf{m}}^l$. However, if the dropout operator directly acts on Bellman equation in this form, the algorithm cannot learn a precise Q-function due to the mismatch between the left side and the right side of Bellman equation (1). As a result, the algorithm fails to learn the value function, i.e., fail to estimate the policy π, resulting in the fail of whole process. To tackle this issue, we introduce minimalist ensemble consistent Bellman update. Let $\mathcal{D}_{\mathbf{m}}^l$ be a dropout operator acting on layer l with parameter $\mathbf{m} \sim \text{Bernoulli}(1-p)$. We define the form of minimalist ensemble consistent Bellman update as

$$\begin{aligned} \mathcal{D}_{\mathbf{m}}^l J^Q(\theta) =& \mathbb{E}_{(s,a)\sim\mathcal{B}}\Big[\frac{1}{2}\Big(\mathcal{D}_{\mathbf{m}}^l Q(s,a;\theta) - \\ & \big(r(s,a) + \gamma \mathbb{E}_{a'\sim\pi(s';\phi')}[\mathcal{D}_{\mathbf{m}}^l Q(s',a';\theta')]\big)\Big)^2\Big], \end{aligned} \tag{9}$$

which means we take the same mask matrix $\mathbf{m}$ acting on both sides of Bellman equation. Thus minimalist ensemble consistent Bellman update can eliminate the mismatch without destroying the diversity of value functions so that good value functions learned and then a good policy can be derived. The diversity and ensemble properties of value functions hold due to the dropout operator.

4.2 MEPG framework

The MEPG framework is formulated by using the aforementioned minimalist ensemble consistent Bellman update in the policy evaluation phase and the conventional policy gradient methods in the policy improvement phase. We apply the MEPG framework to the DDPG algorithm and SAC algorithm, called ME-DDPG and ME-SAC respectively. For the policy improvement phase in ME-DDPG, we utilize the original Deterministic Policy Gradient method (Equation (2)) without $\mathcal{D}_{\mathbf{m}}^l$. For ME-SAC, we keep the original policy update style in SAC. For the policy evaluation phase, we adopt the aforementioned minimalist ensemble consistent Bellman update in Equation (9). In ME-DDPG, we take two tricks from TD3 algorithm [23], which include target policy smoothing regularization and delayed policy updates. And for ME-SAC, we only use one critic with delayed policy updates. ME-DDPG is summarized in Algorithm 1. ME-SAC is summarized in the technical appendix.

Algorithm 1 ME-DDPG

Initialize: actor network π, critic network Q with parameters ϕ, θ, target networks $\phi' \leftarrow \phi$, $\theta' \leftarrow \theta$, and replay buffer $\mathcal{B}$

Parameters: T, p, p, η, d, and $t = 0$

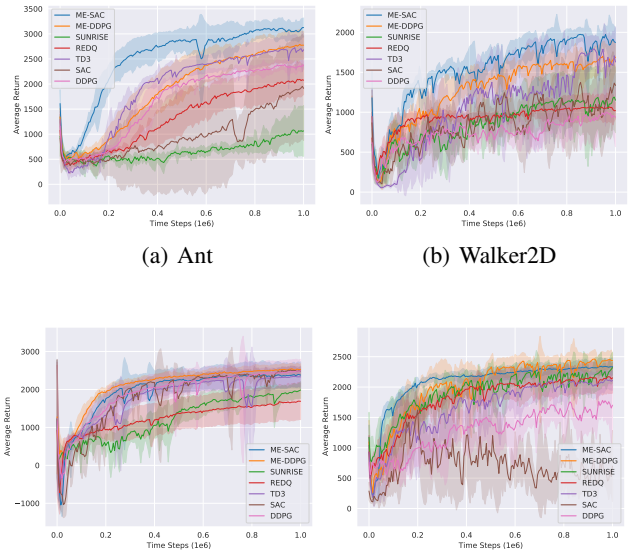
Reset the environment and receive initial state s
while $t < T$ **do**
  Select action with noise $a = \pi(s; \phi) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma^2)$, and receive reward r, new state s'
  Store transition tuple (s, a, r, s') to $\mathcal{B}$
  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi(s'; \phi') + \epsilon, \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}^2), -c, c)$
  Compute target for the Q-function:
  $y \leftarrow r + \gamma \mathcal{D}^l_{\mathbf{m}} Q(s', \tilde{a}; \theta')$
  Update θ by one step gradient descent using:
  $\nabla_\theta J(\theta) = N^{-1} \sum \nabla_\theta \frac{1}{2} \Big(y - \mathcal{D}^l_{\mathbf{m}} Q_\theta(s, a) \Big)^2$
  if t mod d **then**
    Update ϕ by one step of gradient ascent using the Deterministic Policy Gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q(s, a; \theta)|_{a=\pi(s;\phi)} \nabla_\phi \pi(s; \phi)$
    Update target networks:
    $\theta' \leftarrow \eta\theta + (1 - \eta)\theta', \phi' \leftarrow \eta\phi + (1 - \eta)\phi'$
  end if
  $t \leftarrow t + 1$
  $s \leftarrow s'$
end while

4.3 Theoretical Analysis

In this subsection, we provide theoretical insights into the MEPG framework. Let $\hat{Q}^\pi$ be the output of action value function (a neural network), and a loss function $\mathcal{F}(\cdots)$. Let $\mathbf{W}_i$ be the weight matrix of $M_i \times M_{i-1}$ dimensions and the bias $\mathbf{b}_i$ of layer i of dimension M_i for $i \in \{1, 2, \cdots, L\}$. We define Bellman backup operator as

$$\mathcal{T}Q^\pi(s, a) \stackrel{\text{def}}{=} r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a), a' \sim \pi(s')}[Q^\pi(s', a')]. \tag{10}$$

Let Q^π_{True} be the fixed point of Bellman equation (1) w.r.t. policy π. In the DRL setting, the critic network, which characters action value, is used to find the fixed point of Bellman equation w.r.t. current policy π through multiple Bellman update. This optimization method is a different paradigm from supervised learning. For convenient, we denote the input and output sets for critic as $\mathcal{X}$ and $\mathcal{Q}$ where $\mathcal{X} \subseteq \mathcal{S} \times \mathcal{A}$. For input x_i, the output of the action value function is $\hat{Q}_i$. We only discuss policy evaluation problems, i.e., how to approximate the true Q-function given policy π, thus we omit the π in the following statement. For a more detailed analysis on the dropout operator behind deep learning, we would recommend the readers check [35] for detailed analysis on this topic. And our analysis on approximating Bellman backup is based on [35]. We often utilize modern optimization techniques like Adam [39], which introduces the L_2 regularization term in the learning process. Thus the objective for policy evaluation in conventional DRL can



(a) Ant (b) Walker2D

(c) HalfCheetah (d) Hopper

Fig. 1. Performance curves on gym PyBullet suite. The shaded area represents half a standard derivation of the average evaluation over 10 trials. For visual clarity, we use slight exponential smoothing. Results show that the performance of the MEPG framework (ME-DDPG and ME-SAC) can match or outperform that of the tested algorithms.

be formulated as

$$\begin{aligned}\mathcal{L}_{\text{critic}} &= \mathbb{E}_Q\left[\mathcal{F}(\mathcal{T}\hat{Q},\hat{Q})\right] + \lambda\sum_{i=1}^{L}\left(\|\mathbf{W}_i\|_2^2 + \|\mathbf{b}_i\|_2^2\right)\\ &\approx \frac{1}{N}\sum_{i=1}^{N}\mathcal{F}(\mathcal{T}\hat{Q}_i,\hat{Q}_i) + \lambda\sum_{i=1}^{L}\left(\|\mathbf{W}_i\|_2^2 + \|\mathbf{b}_i\|_2^2\right).\end{aligned} \tag{11}$$

By minimizing Equation (11), we find the fixed point of Bellman equation, and then solve the policy evaluation problem. With the application of the dropout operator, the units of the neural network, are set to zero with probability p. Next we consider a deep Gaussian Process (GP) [16]. Now we are given a covariance function

$$\mathbf{C}(\mathbf{x},\mathbf{y}) = \int_{\mathbf{w},b} d\mathbf{w}db\ p(\mathbf{w})p(b)f(\mathbf{w}^\top\mathbf{x}+b)f(\mathbf{w}^\top\mathbf{y}+b), \tag{12}$$

where f is element-wise non-linear function. Equation (12) is a valid covariance function. Assume layer i have a parameter matrix W_i with dimension $\mathsf{M}_i \times \mathsf{M}_{i-1}$ and we include all parameters in a set $\omega = \{\mathsf{W}\}_{i=1}^{L}$. Let $p(\mathbf{w})$ in Equation (12) is the distribution of each row of W_i. We assume that the dimension of the vector $\mathbf{m}_i$ for each GP layer is M_i.

Given some precision parameter $\varepsilon > 0$, the predictive probability of the Deep GP model is

$$\begin{aligned} p(Q \mid \mathbf{x}, \mathcal{X}, \mathcal{Q}) &= \int_{\omega} d\omega \; p(Q \mid x, \omega) p(\omega \mid \mathcal{X}, \mathcal{Q}) \\ p(Q \mid \mathbf{x}, \omega) &= \mathcal{N}(Q; \hat{Q}(x; \omega), \varepsilon \mathbf{I}_D) \\ \hat{Q}(\mathbf{x}; \omega) &= \sqrt{\frac{1}{\mathsf{M}_L}} \mathsf{W}_L f\Big(\cdots \sqrt{\frac{1}{\mathsf{M}_1}} \mathsf{W}_2 f(\mathsf{W}_1 \mathbf{x} + \mathbf{m}_1) \cdots \Big) \end{aligned} \tag{13}$$

We utilize $q(\omega)$ to approximate the intractable posterior $p(\omega; \mathcal{X}, \mathcal{Q})$. Note that $q(\omega)$ is a distribution over matrices whose columns are randomly set to zero. We define $q(\omega)$ as

$$\begin{aligned} &\hat{\mathsf{G}}_i = \mathsf{G}_i \odot \mathrm{diag}([z_{i,j}]_{j=1}^{\mathsf{M}_i}) \\ &z_{i,j} \sim \mathrm{Bernoulli}(1 - p_i), \text{ for } i \in \{1, \cdots, L\}, \; j \in \{1, \cdots, \mathsf{M}_{i-1}\} \end{aligned} \tag{14}$$

where G_i is a matrix as variational parameters. The variable $z_{i,j} = 0$ means unit j in layer $i - 1$ being zero as an input to layer i, which recovers the dropout operator in neural networks. To learn the distribution $q(\omega)$, we minimize the KL divergence between $q(\omega)$ and $p(\omega)$ of the full Deep GP

$$J_{\mathrm{GP}} = -\int_{\omega} d\omega \, q(\omega) \log p(\mathcal{Q} \mid \mathcal{X}, \omega) + \mathrm{D}_{\mathrm{KL}}(q(\omega) \| p(\omega)). \tag{15}$$

The first term in Equation (15) can be approximated by Monte Carlo method. We can approximate the second term in Equation (15), and obtain $\sum_{i=1}^{L} (\frac{p_i l^2}{2} \|\mathsf{G}_i\|_2^2 + \frac{l^2}{2} \|\mathbf{m_i}\|_2^2)$ with prior length scale l (see section 4.2 in [40]). Given the precision parameter $\varepsilon > 0$, the objective of deep GP can be formulated as

$$\begin{aligned} \mathcal{L}_{\mathrm{GP}} \propto & \frac{1}{N\varepsilon} \sum_{i=1}^{N} - \log p(Q_i \mid \mathbf{x}_i; \hat{\omega}) \\ & + \frac{1}{N\varepsilon} \sum_{i=1}^{L} \left(\frac{p_i l^2}{2} \|\mathsf{G}_i\|_2^2 + \frac{l^2}{2} \|\mathbf{m}_i\|_2^2 \right). \end{aligned} \tag{16}$$

We can recover Equation (11) by setting $\mathcal{F}(\mathcal{T}\hat{Q}, \hat{Q}) = -\log p(Q_i \mid x_i; \hat{\omega})$. Note that the sampled $\hat{\omega}$ leads to the realization of the Bernoulli distribution $z_{i,j}$, which is equivalent to the binary variable $z_{i,j}$ in the dropout operator.

The above analysis shows that the policy evaluation process in the MEPG framework with dropout operator can be viewed as a deep Gaussian Process. The uncertainty introduced by dropout operator is still explicitly implied in the model. That means the uncertainty arises from the inherent properties of the model. Thus, the diversity and ensemble properties of value functions hold due to the dropout operator.

5 Experimental Results

In this subsection, we answer the follow questions:

Table 1. The average of top five maximum average returns over five trials of one million time steps for various algorithms. The maximum value for each task is bolded. "InvPen", "InvDou" and "InvPenSwingup" are shorthand for "InvertedPendlum", "InvertedDoublePendlum" and "InvertedPendulumSwingup" respectively.

Algorithm	Ant	HalfCheetah	Hopper	Walker2D	InvPen	InvDouPen	InvPenSwingup	Reacher
ME-SAC	**2906.98**	**3113.21**	2532.98	**1870.53**	1000.0	9359.96	**893.71**	24.51
ME-DDPG	2841.04	2582.32	**2546.56**	1770.11	1000.0	**9359.98**	893.02	24.34
SUNRISE	1234.89	2017.94	2386.46	1269.78	1000.0	9359.93	893.2	**27.44**
REDQ	2155.07	1734.5	2215.06	1085.09	1000.0	9359.16	891.37	26.02
TD3	2758.38	2360.2	2190.12	1868.65	1000.0	9359.66	890.97	24.99
SAC	2009.36	2567.7	2317.64	1776.34	1000.0	9358.78	892.36	24.13
DDPG	2446.75	2501.24	1918.42	1142.71	1000.0	9358.94	546.65	24.78

- How good MEPG framework is compared with SOTA model-free and ensemble DRL algorithms?
- What is the contribution of each component to the algorithm?
- How does the training time cost and number of parameters of our algorithm compared to other tested algorithms?
- How sensitive is MEPG to hyper-parameters?

To evaluate our framework MEPG, we conduct experiments on open source PyBullet suite [41], interfaced through Gym simulator [42]. We highlight the fact that the PyBullet suite is usually considered harder to train than MuJoCo suite [43]. Given the recent discussion in DRL reproducibility [7,8], we strictly control all random seeds, and our results are reported over 5 trials unless otherwise stated, with the same setting and fair evaluation metrics. More experimental results and more details can be found in the technical appendix.

5.1 Evaluation

To evaluate the MEPG framework, we measure ME-DDPG and ME-SAC performance on PyBullet tasks compared with the state-of-the-art model-free algorithm and recently proposed ensemble DRL algorithms such as SUNRISE [14] and REDQ [27]. For TD3[3], SUNRISE[4] and REDQ[5], we use the authors' implementation with default hyper-parameters. We keep the same hyper-parameters for DDPG and SAC. For a fair comparison, we replace one-time policy update every twenty times critic optimization with one-time policy update every two-time critic optimization in REDQ. We run each task for one million time steps and perform one gradient step after each interaction step. Every $5,000$ time steps, we execute an evaluation step over 10 episodes without

[3] Code: `https://github.com/sfujim/TD3`
[4] Code: `https://github.com/pokaxpoka/sunrise`
[5] Code: `https://github.com/watchernyu/REDQ`

any exploration operation in every algorithm. Our performance comparison results are presented in Table 1 and the learning curves are in Figure 1. The results show that our proposed algorithm ME-DDPG and ME-SAC are just as good as, if not better than, state-of-the-of methods without any auxiliary tasks. For the Pendulum family of environments, all algorithms are equally good. Our framework is best in terms of learning speed and final performance for the Ant, Walker2D and Hopper environments. We highlight that our proposed algorithms ME-DDPG and ME-SAC achieve or surpass the best performance while consuming fewer computational resources. More experimental results and details are in the technical appendix.

5.2 Ablation Studies

We conduct ablation studies to understand the contribution of each individual component. We show the ablation results for ME-DDPG and ME-SAC in Table 2, where we compare the performance of removing or adding specific component from ME-DDPG or ME-SAC. Firstly, we investigate three key components, i.e., dropout (DO), Target Policy Smoothing (TPS), and Delay Update (DU) in ME-DDPG. In practice, we do not utilize Clipped Double Q-learning (CDQ) from TD3. The results show the effectiveness of each component varies in different tasks. CDQ usually has some help that is not particularly obvious but is not consistent as there are some environments that overestimation may encourage exploration and thus lead to better performance. DO, DU and TPS help a lot. Secondly, we perform a similar ablation analysis for the ME-SAC algorithm, where "FIX-ENT" means we adopt a fixed entropy coefficient α. The ablation results of the ME-DDPG algorithm for CDQ, DU, and DO are also applicable to the ME-SAC method. Here we find that automatic adjustment of entropy is extremely helpful. In addtion, for both ME-DDPG and ME-SAC, the minimalist ensemble consistent Bellman update helps a lot and brings performance improvements. Additional experimental results and learning curves can be found in the technical appendix.

5.3 Run time and number of parameters

We evaluate the run time of one million time steps of training for each tested RL algorithm. In addition, we also quantify the number of parameters of neural networks corresponding to the tested algorithms in different environments. For a fair comparison, we keep the same update steps in REDQ. And we cancel the evaluation process in this test. We show the run time test results in Table 3 and parameters quantification in Table 4. The MEPG framework consumes the shortest time among the same skeleton algorithms. Unsurprisingly, We find our algorithm ME-DDPG and ME-SAC act favorably compared to other algorithms we tested in terms of wall-clock training time. Our MEPG framework not only runs in one-third to one-seventh the time of ensemble learning methods, but even less than the training time of model-free DRL algorithms. All run time experiments are conducted on a single GeForce GTX 1070 GPU and an Intel Core i7-8700K CPU at 2.4GHZ. The number of parameters of the MEPG framework is between 14% and 27% of the number of parameters of the tested ensemble algorithms, but our algorithms perform better than the tested ones.

Table 2. The average of the top five maximum average returns over five trials of one million time steps for ablation studies. ± means adding or removing corresponding component. The maximum value for each task is bolded. "MED", "MES" and "HCheetah", "Walker" are shorthand for "ME-DDPG", "ME-SAC" "HalfCheetah", and "Walker2D" respectively.

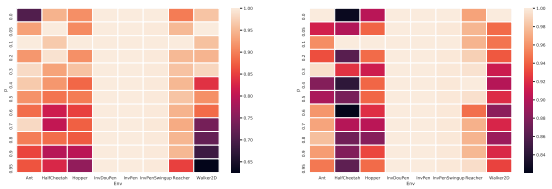
Algorithm	Ant	HCheetah	Hopper	Walker
MED	2841.04	**2582.32**	2546.56	1770.11
ME+CDQ	**2892.90**	2003.77	**2584.1**	**2094.6**
MED-DO	2001.56	2522.97	2326.2	1774.61
MED-DU	2610.23	2575.64	2330.16	1784.34
MED-TPS	2623.13	2605.99	2328.32	1675.31
DDPG	2446.75	2501.24	1918.42	1142.71
MES	2906.98	**3113.21**	**2532.98**	1870.53
MES+CQD	**3039.11**	2209.46	2408.86	**2060.8**
MES-DU	2605.52	2718.17	2211.27	1631.99
MES+FIXENT	818.03	733.59	1632.85	843.01
SAC	2009.36	2567.7	2317.64	1776.34

Table 3. Run time comparison of training each RL algorithm.

Algorithm	Ant	HalfCheetah	Hopper	Walker2D
ME-DDPG	47m	44m	42m	45m
TD3	57m	55m	52m	55m
DDPG	1h 1m	58m	56m	58m
ME-SAC	1h 14m	1h 13m	1h 10m	1h 12m
SAC	1h 17m	1h 15m	1h 12m	1h 15m
REDQ	3h 43m	3h 41m	3h 29m	3h 35m
SUNRISE	7h 24m	7h 15m	7h 10m	7h 16m

Table 4. The number of parameters is given in millions. For all tested environments, we utilize the same network architectures and the same hyper-parameters. The amount of parameters differs because different environments have various state and action dimensions, which results in different input and output dimensions of the neural network.

Algorithm	Ant	HalfCheetah	Hopper	Walker2D
ME-DDPG	0.302M	0.297M	0.283M	0.293M
ME-SAC	0.226M	0.223M	0.212M	0.220M
TD3	0.453M	0.446M	0.425M	0.440M
SAC	0.377M	0.372M	0.354M	0.367M
REDQ	1.586M	1.564M	1.489M	1.543M
SUNRISE	1.132M	1.117M	1.063M	1.101M



(a) ME-DDPG (b) ME-SAC

Fig. 2. Performance of ME-SAC and ME-DDPG algorithms given different p-values in different environments. Each cell represents the average of the top five maximum average returns in a run over five trials of one million time steps for various p-value. For most environments and p-values, both algorithms are able to learn successfully. Besides, the difference in performance between the different p-values is not very significant. Relatively speaking, smaller p-values give better performance. The performance of both algorithms perform best when $p = 0.1$.

5.4 Hyper-parameter Sensitivity

The MEPG framework only introduces one hyper-parameter p, which means the probability of setting a unit of neural network to zero. We take eleven p values in interval $[0, 1]$. For visual simplicity, we normalize the data by $\text{Ret}_p^{\text{env}} = \text{Ret}_p^{\text{env}} / \max_p\{\text{Ret}_p^{\text{env}}\}$, where $\text{Ret}_p^{\text{env}}$ means the average of top five maximum average returns in a run over five trials of one million time steps for various p-value on `env` environment. We show the results in Figure 2. Each cell represents the result of different p-value on various environments. The experimental results show that even large p-values, i.e., implicitly integrating enough models, do not cause the learning process to fail. Overall, the difference in performance between the different p-values is not very significant. For the Pendulum family of environments, the performance of the MEPG framework induced by various p values are equally good. Relatively speaking, smaller p-value give better performance. Our MEPG framework is not extremely hyper-parameter sensitive. Therefore we set $p = 0.1$ as the default hyper-parameter setting for the ME-DDPG and ME-SAC algorithms.

6 Conclusion

Ensemble RL methods introduce multiple value and policy functions, aiming to mitigate instability in Q-learning and to learn a robust policy. As a result, it raises with it the problem of large mount of resource consumption issue and introduce more hyper-parameters. Before applying the ensemble DRL algorithms to the real world, we need to solve the aforementioned tricky problems. Therefore, in this paper, we propose a novel ensemble reinforcement learning framework, called Minimalist Ensemble Policy Gradient (MEPG). The ensemble property of the MEPG framework is induced by minimalist ensemble consistent Bellman update that utilizes the dropout operator. Then we show that the policy evaluation in MEPG framework is mathematically equivalent to a deep Gaussian Process. Next we verify the effectiveness of MEPG in the PyBullet control suite, which is generally considered harder than the commonly used MuJoCo suite. Moreover, the

experimental results show that the performance of ensemble DRL algorithms can be easily outperformed by MEPG framework. Besides performance, the MEPG framework have tremendous advantages in terms of run time and number of parameters compared to the tested model-free and ensemble DRL algorithms. The core technique we adopt is the minimalist ensemble consistent Bellman update, wherein the dropout operator is a popular method in deep learning. For the reinforcement learning community, however, the neural network is generally considered as a function approximator under continuous control tasks. As we all know, the neural network is a complex and subtle tool, and reinforcement learning researcher may concentrate on the role of neural networks in DRL. Thus, we highlight that the potential of neural networks may not have been fully exploited, as exemplified by our MEPG framework.

7 Acknowledgments

We thank Jing Cui for her helpful comments.

References

1. Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners, 2020.
2. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
3. David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.
4. 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):529–533, 2015.
5. Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.
6. Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.
7. Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
8. Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, et al. What matters in on-policy reinforcement learning? a large-scale empirical study. In *ICLR 2021-Ninth International Conference on Learning Representations*, 2021.
9. Marco A Wiering and Hado Van Hasselt. Ensemble algorithms in reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(4):930–936, 2008.

10. Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29:4026–4034, 2016.
11. Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in Neural Information Processing Systems*, 31, 2018.
12. Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. In *International Conference on Learning Representations*, 2018.
13. Rohan Saphal, Balaraman Ravindran, Dheevatsa Mudigere, Sasikant Avancha, and Bharat Kaul. Seerl: Sample efficient ensemble reinforcement learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1100–1108, 2021.
14. Kimin Lee, Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning. In *International Conference on Machine Learning*, pages 6131–6141. PMLR, 2021.
15. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.
16. Andreas Damianou and Neil D Lawrence. Deep gaussian processes. In *Artificial intelligence and statistics*, pages 207–215. PMLR, 2013.
17. 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 *ICLR (Poster)*, 2016.
18. 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):529–533, 2015.
19. Sebastian Thrun. Issues in using function approximation for reinforcement learning. In *Proceedings of the Fourth Connectionist Models Summer School*, pages 255–263. Hillsdale, NJ, 1993.
20. Hado Hasselt. Double q-learning. *Advances in neural information processing systems*, 23:2613–2621, 2010.
21. Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.
22. David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014.
23. Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.
24. Brian D Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University, 2010.
25. Qingfeng Lan, Yangchen Pan, Alona Fyshe, and Martha White. Maxmin q-learning: Controlling the estimation bias of q-learning. In *International Conference on Learning Representations*, 2019.
26. Qiang He and Xinwen Hou. Wd3: Taming the estimation bias in deep reinforcement learning. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 391–398. IEEE, 2020.
27. Xinyue Chen, Che Wang, Zijian Zhou, and Keith W Ross. Randomized ensembled double q-learning: Learning fast without a model. In *International Conference on Learning Representations*, 2020.

28. Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pages 104–114. PMLR, 2020.
29. Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.
30. Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
31. Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.
32. Pierre-Alexandre Kamienny, Kai Arulkumaran, Feryal Behbahani, Wendelin Boehmer, and Shimon Whiteson. Privileged information dropout in reinforcement learning, 2020.
33. Björn Lütjens, Michael Everett, and Jonathan P. How. Safe reinforcement learning with model uncertainty estimates. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8662–8668, 2019.
34. Yue Wu, Shuangfei Zhai, Nitish Srivastava, Joshua Susskind, Jian Zhang, Ruslan Salakhutdinov, and Hanlin Goh. Uncertainty weighted actor-critic for offline reinforcement learning, 2021.
35. Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.
36. Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
37. Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954.
38. Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.
39. Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015.
40. Y Gal and Z Ghahramani. Dropout as a bayesian approximation: Appendix. *arXiv preprint arXiv:1506.02157*, 2015.
41. Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2021.
42. Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.
43. Antonin Raffin and Freek Stulp. Generalized state-dependent exploration for deep reinforcement learning in robotics. *CoRR*, abs/2005.05719, 2020.
44. Denis Yarats and Ilya Kostrikov. Soft actor-critic (sac) implementation in pytorch. `https://github.com/denisyarats/pytorch_sac`, 2020.

Technical Appendix

8 Missing Algorithm

We give the detailed description about ME-SAC in Algorithm 2.

Algorithm 2 ME-SAC

Initialize: actor network π and critic network Q with parameters ϕ, θ, target networks $\theta' \leftarrow \theta$
Initialize: Replay buffer $\mathcal{B}$ **Parameters**: total steps T, p, $t = 0$, target entropy $\mathcal{H}$ and dropout probability p.

Reset the environment and receive initial state s
while $t < T$ **do**
  Select action $a \sim \pi(\cdot \mid s; \phi)$, and receive reward r, new state s'
  Store transition tuple (s, a, r, s') to $\mathcal{B}$
  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  Compute target for the Q-function:
  $y \leftarrow r + \gamma \mathcal{D}^l_{\mathbf{m}} Q(s', \tilde{a}'; \theta') - \alpha \log \pi(\tilde{a}' \mid s'; \phi),\ \tilde{a}' \sim \pi(\cdot \mid s'; \phi)$
  Update θ by one step of gradient descent using:
  $\nabla_\theta J(\theta) = \nabla_\theta N^{-1} \sum \Big(y - \mathcal{D}^l_{\mathbf{m}} Q_\theta(s, a) \Big)^2$
  if t mod d **then**
    Update ϕ by one step of gradient ascent using:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a \Big(Q(s, \tilde{a}; \theta) - \alpha \log \pi(\tilde{a} \mid s; \phi) \Big),\ \tilde{a} \sim \pi(\cdot \mid s; \phi)$
    Update α by one step gradient descent using:
    $\nabla_\alpha J(\alpha) = N^{-1} \sum \nabla_\alpha \Big(- \alpha \log \pi(a \mid s; \alpha, \phi) - \alpha \mathcal{H} \Big)$
    Update target network:
    $\theta' \leftarrow \eta\theta + (1 - \eta)\theta'$
  end if
  $t \leftarrow t + 1$
  $s \leftarrow s'$
end while

9 More Experimental Results and Details

We mostly report experimental results on four environments in the body due to space limitations. In this section, we provide more experimental results and details on eight environments. Note that all the experimental results are conducted on the PyBullet suite. We highlight the fact that the PyBullet suite is usually considered harder to train than MuJoCo suite [43] as we discussed. Besides, we strictly control all random seeds, and our results are reported over 5 trials unless otherwise stated, with the same setting and fair evaluation metrics.

Evaluation. We provide the learning curves on eight environments in Figure 3.

Ablation. We provide the learning curves for ablation analysis for ME-DDPG and ME-SAC in Figure 4 and Figure 5 respectively. And We also compute the top five

maximum average returns of one million time steps for ablation analysis, which are show in Table 7 and Table 8.

Run time. We show the run time statistics in Table 6.

Number of parameters. We show the number of parameters statistics in Table 9.

Hyper-paramter sensitivity. We provide the learning curves for different p in different environments in Figure 6 and Figure 7. And we also provide the average of top five maximum average returns of one million time steps in Table 10 and Table 11.

9.1 Implementation details and Hyper-parameter setting

We utilize the authors implementation of TD3, SUNRISE and REDQ without any modification as we discussed. For the implementation of SAC, we refer to [44]. And we do not change the default hyper-parameters for TD3, SAC, SUNRISE and REDQ algorithm. For a fair comparison, we keep the same hyper-parameters to TD3 and SAC implementations respectively. But we only utilize one critic with its target. If the hyper-parameters of ME-SAC and ME-DDPG correspond to the SAC and DDPG, we use the same hyper-parameters. To implement minimalist ensemble consistent Bellman update, we take a $\mathbf{1}$ matrix, then let the $\mathbf{1}$ matrix pass through a dropout layer. The output of the dropout produce a consistent mask. We apply the same mask to the critic and its target. We give the detail description about our hyper-parameters in Table 5.

Table 5. Hyper-parameters settings for our implementation.

Hyper-parameter	Value
Shared hyper-parameters	
discount (γ)	0.99
Replay buffer size	10^6
Optimizer	Adam [39]
Learning rate for actor	3×10^{-4}
Learning rate for critic	3×10^{-4}
Number of hidden layer for all networks	2
Number of hidden units per layer	256
Activation function	ReLU
Mini-batch size	256
Random starting exploration time steps	2.5×10^4
Target smoothing coefficient (η)	0.005
Gradient Clipping	False
Target update interval (d)	2
TD3	
Variance of exploration noise	0.2
Variance of target policy smoothing	0.2
Noise clip range	$[-0.5, 0.5]$
Delayed policy update frequency	2
ME-DDPG	
Variance of exploration noise	0.2
Variance of target policy smoothing	0.2
Noise clip range	$[-0.5, 0.5]$
Delayed policy update frequency	2
Dropout probability (p)	0.1
SAC	
Target Entropy	- dim of $\mathcal{A}$
Learning rate for α	1×10^{-4}
ME-SAC	
Target Entropy	- dim of $\mathcal{A}$
Learning rate for α	1×10^{-4}
Dropout probability (p)	0.1

Table 6. Run time comparison of training each RL algorithm for eight PyBullet environments. "InvPen", "InvDou" and "InvPenSwingup" are shorthand for "InvertedPendlum", "InvertedDoublePendlum" and "InvertedPendulumSwingup" respectively.

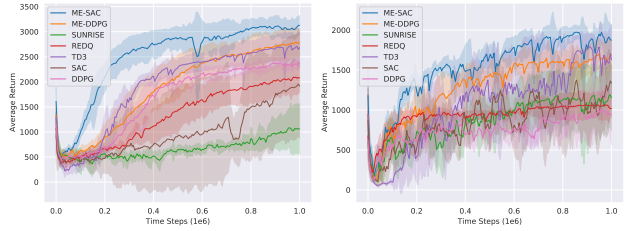
Algorithm	Ant	HalfCheetah	Hopper	Walker2D	Reacher	InvPen	InvDouPen	InvPenSwingup
ME-DDPG	47m	44m	42m	45m	37m	36m	36m	36m
TD3	57m	55m	52m	55m	48m	47m	47m	47m
DDPG	1h 1m	58m	56m	58m	51m	49m	50m	49m
ME-SAC	1h 14m	1h 13m	1h 10m	1h 12m	1h 5m	1h 3m	1h 4m	1h 4m
SAC	1h 17m	1h 15m	1h 12m	1h 15m	1h 8m	1h 6m	1h 6m	1h 6m
REDQ	3h 43m	3h 41m	3h 29m	3h 35m	2h 57m	3h 10m	3h 12m	3h 12m
SUNRISE	7h 24m	7h 15m	7h 10m	7h 16m	7h 1m	6h 56m	7h 6m	6h 57m

Table 7. The average of the top five maximum average returns over five trials of one million time steps for ablation studies. ± means adding or removing corresponding component. "InvPen", "InvDou" and "InvPenSwingup" are shorthand for "InvertedPendlum", "InvertedDoublePendlum" and "InvertedPendulumSwingup" respectively.

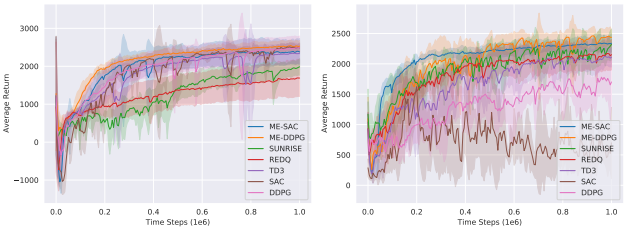
Algorithm	Ant	HalfCheetah	Hopper	Walker2D	InvPen	InvDouPen	InvPenSwingup	Reacher
ME-DDPG	2841.04	2582.32	2546.56	1770.11	1000.0	9359.98	893.02	24.34
ME-DDPG+CDQ	2892.9	2003.77	2584.1	2094.6	1000.0	9358.76	890.76	15.05
ME-DDPG-DO	2001.56	2522.97	2326.2	1774.61	1000.0	9358.73	892.04	21.82
ME-DDPG-DU	2610.23	2575.64	2330.16	1784.34	1000.0	9359.18	890.84	24.08
ME-DDPG-TPS	2623.13	2605.99	2328.32	1675.31	1000.0	7522.81	891.4	23.93
DDPG	2446.75	2501.24	1918.42	1142.71	1000.0	9358.94	546.65	24.78

Table 8. The average of the top five maximum average returns over five trials of one million time steps for ablation studies. ± means adding or removing corresponding component. "InvPen", "InvDou" and "InvPenSwingup" are shorthand for "InvertedPendlum", "InvertedDoublePendlum" and "InvertedPendulumSwingup" respectively.

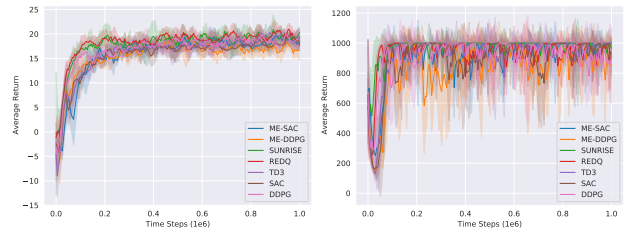
Algorithm	Ant	HalfCheetah	Hopper	Walker2D	InvPen	InvDouPen	InvPenSwingup	Reacher
ME-SAC	2906.98	3113.21	2532.98	1870.53	1000.0	9359.96	893.71	24.51
ME-SAC+CQD	3039.11	2209.46	2408.86	2060.8	1000.0	9356.73	890.18	21.68
ME-SAC-DU	2605.52	2718.17	2211.27	1631.99	1000.0	9357.95	892.2	24.7
ME-SAC+FIXENT	818.03	733.59	1632.85	843.01	1000.0	9358.94	835.59	26.32
SAC	2009.36	2567.7	2317.64	1776.34	1000.0	9358.78	892.36	24.13



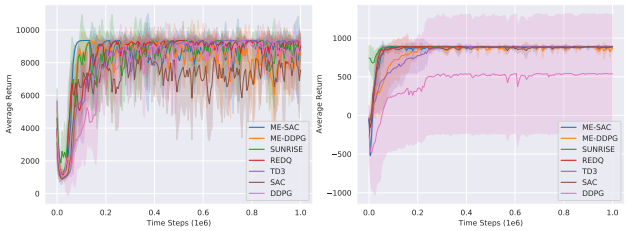
(a) Ant (b) Walker2D



(c) HalfCheetah (d) Hopper

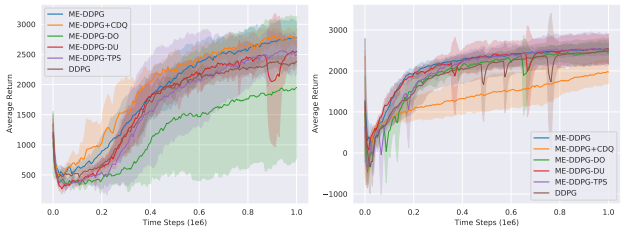


(e) Reacher (f) InvertedPendulum

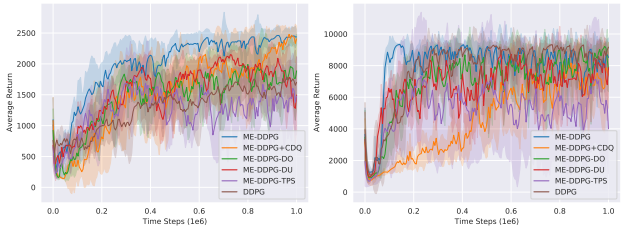


(g) InvertedDoublePendulum (h) InvertedPendulumSwingup

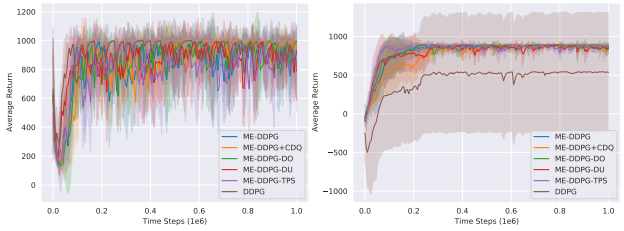
Fig. 3. Performance curves on gym PyBullet suite. The shaded area represents half a standard derivation of the average evaluation over 10 trials. For visual clarity, we use slight exponential smoothing. Results show that the performance of the MEPG framework (ME-DDPG and ME-SAC) can match or outperform that of the tested algorithms.



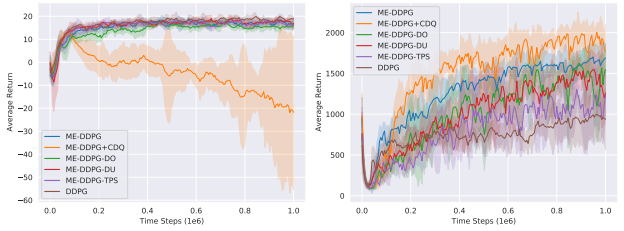
(a) Ant (b) HalfCheetah



(c) Hopper (d) InvertedDoublePendulum

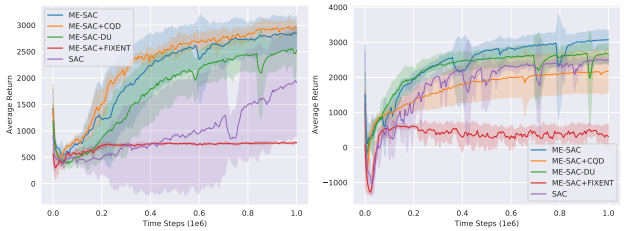


(e) InvertedPendulum (f) InvertedPendulumSwingup

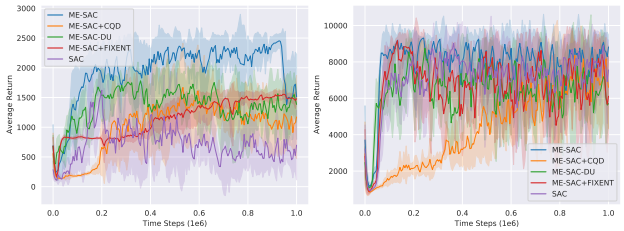


(g) Reacher (h) Walker2D

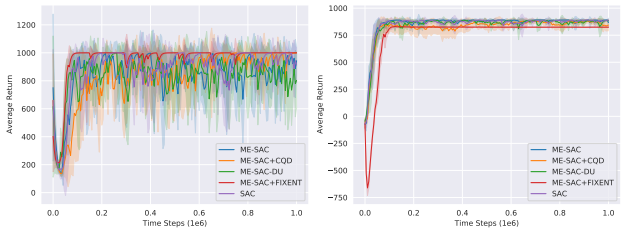
Fig. 4. Performance curves on gym PyBullet suite. The shaded area shows a half standard deviation.



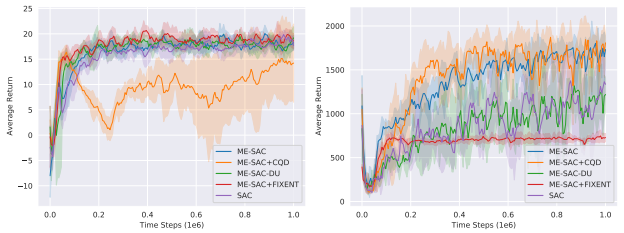
(a) Ant (b) HalfCheetah



(c) Hopper (d) InvertedDoublePendulum

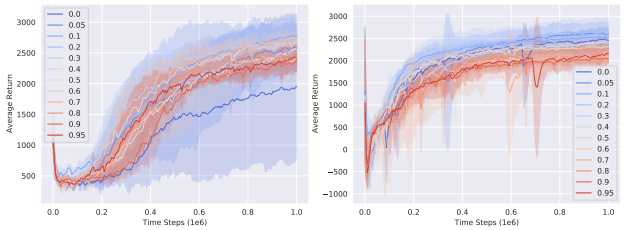


(e) InvertedPendulum (f) InvertedPendulumSwingup

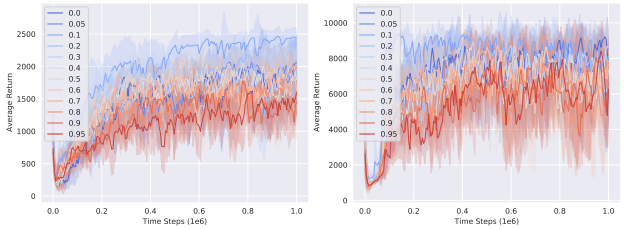


(g) Reacher (h) Walker2D

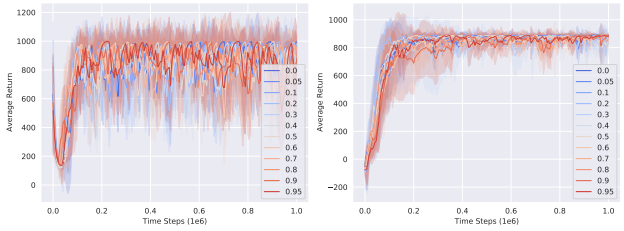
Fig. 5. Performance curves on gym PyBullet suite. The shaded area shows a half standard deviation.



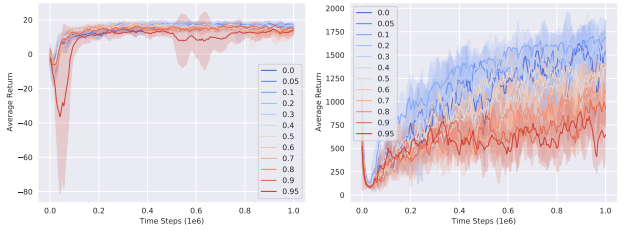
(a) Ant (b) HalfCheetah



(c) Hopper (d) InvertedDoublePendulum

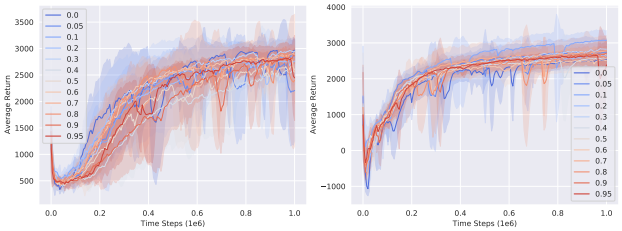


(e) InvertedPendulum (f) InvertedPendulumSwingup

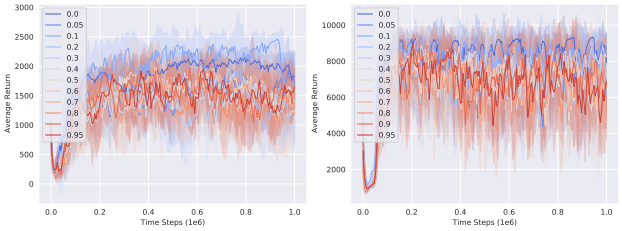


(g) Reacher (h) Walker2D

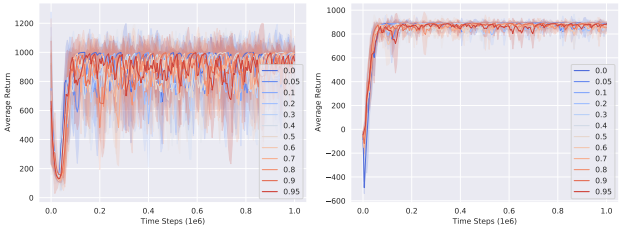
Fig. 6. Performance curves for sensitivity analysis on gym PyBullet suite. The shaded area shows a standard derivation.



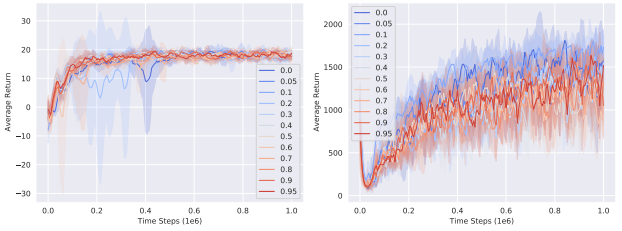
(a) Ant (b) HalfCheetah



(c) Hopper (d) InvertedDoublePendulum



(e) InvertedPendulum (f) InvertedPendulumSwingup



(g) Reacher (h) Walker2D

Fig. 7. Performance curves for sensitivity analysis on gym PyBullet suite. The shaded area shows a standard derivation.

Table 9. The number of parameters is given in millions. For all tested environments, we utilize the same network architecture and the same hyper-parameters. The amount of parameters differs because different environments have various state and action dimensions, which results in different input and output dimensions of the neural network.

Algorithm	Ant	HalfCheetah	Hopper	Walker2D	InvPen	InvDouPen	InvPenSwingup	Reacher
ME	0.302M	0.297M	0.283M	0.293M	0.271M	0.275M	0.271M	0.276M
ME-SAC	0.226M	0.223M	0.212M	0.220M	0.203M	0.206M	0.203M	0.207M
TD3	0.453M	0.446M	0.425M	0.440M	0.407M	0.413M	0.407M	0.414M
SAC	0.377M	0.372M	0.354M	0.367M	0.339M	0.344M	0.339M	0.345M
REDQ	1.586M	1.564M	1.489M	1.543M	1.424M	1.446M	1.424M	1.451M
SUNRISE	1.132M	1.117M	1.063M	1.101M	1.017M	1.032M	1.017M	1.036M

Table 10. The average of the top five maximum average returns of ME-DDPG algorithm in a run over five trials of one million time steps for various p-value. Overall, the difference in performance between the different p-values is not very significant. Relatively speaking, smaller p-values give better performance.

p	Ant	HalfCheetah	Hopper	Walker2D	InvPen	InvDouPen	InvPenSwingup	Reacher
0.0	2001.56	2522.97	2326.2	1774.61	1000.0	9358.73	892.04	21.82
0.05	2641.03	2668.75	2311.99	1846.5	1000.0	9359.11	891.21	23.16
0.1	2841.04	2582.32	6.56	1770.11	1000.0	9359.98	893.02	24.34
0.2	2616.36	2645.25	2326.77	1747.17	1000.0	9359.44	891.41	23.14
0.3	2791.95	2481.9	2443.04	1807.43	1000.0	9359.35	890.52	23.36
0.4	2746.82	2457.72	2244.52	1548.22	1000.0	9359.14	890.61	23.12
0.5	2603.27	2376.87	2283.3	1687.19	1000.0	9359.21	890.7	23.38
0.6	2512.87	2174.87	2169.95	1631.38	1000.0	9359.34	890.77	23.01
0.7	2790.01	2146.34	2226.71	1360.51	1000.0	9359.31	890.22	22.84
0.8	2552.4	2360.52	2216.69	1335.82	1000.0	9359.15	890.97	23.09
0.9	2427.33	2119.58	2029.8	1266.88	1000.0	9359.37	890.76	23.1
0.95	2461.24	2209.9	1939.54	1154.81	1000.0	9359.24	890.54	20.73

Table 11. The average of the top five maximum average returns of ME-SAC algorithm in a run over five trials of one million time steps for various p-value. Overall, the difference in performance between the different p-values is not very significant. Relatively speaking, smaller p-values give better performance.

p	Ant	HalfCheetah	Hopper	Walker2D	InvPen	InvDouPen	InvPenSwingup	Reacher
0.0	3036.48	2569.0	2284.7	1973.73	1000.0	9359.1	893.35	24.32
0.05	2772.88	2800.64	2385.61	1864.32	1000.0	9358.91	892.12	24.53
0.1	2906.98	3113.21	2532.98	1870.53	1000.0	9359.96	893.71	24.51
0.2	2835.34	2683.07	2392.05	1849.91	1000.0	9358.78	892.13	24.82
0.3	3018.22	2875.04	2308.31	1796.81	1000.0	9358.82	892.16	25.04
0.4	2673.15	2623.41	2386.58	1778.47	1000.0	9358.21	892.18	25.15
0.5	2721.36	2726.1	2386.39	1815.28	1000.0	9358.95	892.27	25.02
0.6	2919.25	2554.68	2332.94	1744.59	1000.0	9358.85	892.15	23.79
0.7	2937.18	2804.65	2287.79	1810.17	1000.0	9358.9	892.18	24.47
0.8	2974.49	2717.76	2233.29	1756.7	1000.0	9358.5	892.01	24.5
0.9	2927.92	2737.94	2358.26	1811.99	1000.0	9358.88	892.41	24.54
0.95	2883.05	2690.42	2391.78	1834.05	1000.0	9358.59	892.38	24.99