Reusable Reinforcement Learning via Shallow Trails

Yang Yu, *Member, IEEE,* Shi-Yong Chen, Qing Da, Zhi-Hua Zhou *Fellow, IEEE*

***Abstract*—Reinforcement learning has shown great success in helping learning agents accomplish tasks autonomously from environment interactions. Meanwhile in many real-world applications, an agent needs not only to accomplish a fixed task, but instead a range of tasks. For this goal, an agent can learn a *meta-policy* over a set of training tasks that are drawn from an underlying distribution. By maximizing the total reward summed over all the training tasks, the meta-policy can then be reused in accomplishing test tasks from the same distribution. However, in practice we face two major obstacles to train and reuse meta-policies well. First, how to identify tasks that are unrelated or even opposite with each other, in order to avoid their mutual interference in the training. Second, how to characterize task features, according to which a meta-policy can be reused. In this work, we propose the MAPLE approach that overcomes the two difficulties by introducing the *shallow trail*. It probes a task by running a roughly trained policy. Using the rewards of the shallow trail, MAPLE automatically groups similar tasks. Moreover, when the task parameters are unknown, the rewards of the shallow trail also serves as task features. Empirical studies on several controlling tasks verify that MAPLE can train meta-policies well and receives high reward on test tasks.**

***Index Terms*—reinforcement learning, meta-policy, shallow trail**

I. Introduction

In reinforcement learning [1], an agent learns from trial-and-error feedback rewards from its environment, and results in a policy that maps states to actions to maximize the long-term total reward as a delayed supervision signal [2]. Reinforcement learning combining with the neural networks has made great progress recently, including playing Atari games [3] and beating world champions at the game of Go [4]. However, in traditional reinforcement learning setting, a fixed task is considered. The agent learns in a task and will be applied only in the same task. However in many real-world applications, it is quite rare that the task is static [5]. Once the task changes, in the traditional way, the agent has to be re-trained, which drastically degrades the practical applicability of reinforcement learning.

Many attempts have been made to address this problem. Studies mainly focused on transfer reinforcement learning [6] aiming at generalization across multiple tasks. In the transfer reinforcement learning setting, an agent has previously experienced a set of tasks. For learning in an unseen task, instead of starting from scratch, the agent utilizes its experience to help the learning [7]–[10]. A special family is multi-task reinforcement learning, which concerns a sequence of reinforcement learning tasks [11], [12]. The problem of generalization across multiple tasks, however, has only been partially addressed. There are still many issues need to be further investigated, including how to gain qualified experiences and how can the accumulation of experiences help learning future tasks. Moreover, the *negative transfer* problem that decreases the performance is often observed when transferring between irrelevant tasks [13], which needs to be avoided carefully.

We notice that, when tasks are well parameterized and the task parameters are drawn from an underlying distribution, it is possible to learn a policy over the task distribution [14], i.e., *meta-policy* in this paper, and reuse it on test tasks. However, learning a good meta-policy faces some difficulties. One major difficulty is that training tasks may be irrelevant or even have conflicting goals, for which the meta-policy with a single model may be hard to learn. A better way is to divide opposite tasks for different models, given that we are able to know if two tasks are opposite. Another one is that, previous approaches commonly assume that the task parameters are known in a priori [14], while it could be hard to obtain such parameters of real-world tasks.

This paper studies the meta-policy learning problem, attempting to tackle the two difficulties. In order to reveal the relationship among tasks, the idea is that similar tasks tend to get similar rewards when executing the same policy, while opposite tasks may have very different rewards. Thus we propose the *shallow trail* trick: we firstly obtain a group of prototype policies by a rough task-specific learning for just a few iterations on the training tasks. We then run all the prototype policies on each task to obtain a reward vector. The reward vector is used to measure the similarity among tasks. Besides, the reward vector could also be used as a task feature, since they can be highly correlated with the tasks. In this way we can learn a meta-policy even when the task parameters are unknown.

Combined the shallow trail idea with the policy search framework, we proposed the MAPLE (MetA-Policy LEarning) method. It joins the state features with the task parameters (MAPLE-P) when the task parameters are available, or the reward vectors (MAPLE-R) when the task parameters are unavailable, and learns a meta-policy by a policy search approach. The meta-policy can then be directly reused in new tasks drawn from the same task distribution. Experiments

Yang Yu, Shi-Yong Chen and Zhi-Hua Zhou are with National Key Laboratory for Novel Software Technology, and Collaborative Innovation Center of Novel Software Technology and Industrailization, Nanjing University, Nanjing, 210023, China. Qing Da is with Alibaba Inc. This work was partially done when Q. Da was a graduate student in Nanjing University. Z.-H. Zhou is the corresponding author. E-mail: {yuy, chensy, daq, zhouzh}@lamda.nju.edu.cn

Manuscript received January 10, 2018; revised xxx xx, 2018.

on controlling tasks in the OpenAI Gym environments, with varying configurations are conducted. In our experiments, employing TRPO [15] as the base policy search method, the results show that MAPLE-P and MAPLE-R both perform significantly better than the baselines. While MAPLE-R using the shallow trails is inferior to MAPLE-P using the ground-truth parameters, MAPLE-R can still achieve a good performance. As a summary, this paper makes the following contributions:

1) Propose the shallow trail trick to address the issue of task similarity measurement and the issue of missing task parameters at the same time;

2) Propose the MAPLE approach based on the shallow trail trick, and verify its effectiveness in learning from training tasks and reusing in test tasks;

3) Verify the effectiveness of the reward vectors from the shallow trails for task representation, and that task grouping is helpful in learning meta-policies.

We hope that the above contributions can provide useful reference for further development of reusable reinforcement learning.

The rest sections sequentially present the background, the proposed approach, the experiments, and the conclusion.

II. Background

A. Reinforcement Learning

Reinforcement learning is commonly studied through the Markov Decision Process (MDP) framework [1], [16]. An MDP is a tuple (S, A, T, R, γ) where S is the set of states, A is the set of action, $T(s_j|s_i, a) : S \times A \times S \to \mathbb{R}$ is the transition probability of reaching state s_j after executing action a on state s_i, $R(s, a) : S \times A \to \mathbb{R}$ is the immediate reward after executing action a from state s_i, and γ is the discounting factor. We use π to denote a stochastic policy, i.e., $\pi(s, a) : S \times A \to [0, 1]$ is the probability of executing action a at state s and $\sum_{a \in A} \pi(s, a) = 1$ for any s. The goal of reinforcement learning is to find a policy π that maximizes the expected long term reward, or return. we denote $J(\pi)$ as the expected value of discounted sum of rewards starting from an initial state s_0

$$J(\pi) = \mathbb{E}_{s_0, a_0, \ldots}[\sum\nolimits_{t=0}^{\infty} \gamma^t R(s_t, a_t)],$$

where $a_t \sim \pi(s_t, \cdot), s_{t+1} \sim P(\cdot|s_t, a_t)$, The expected total reward J is related to the value function

$$V^\pi(s) = \mathbb{E}[\sum\nolimits_{t=0}^{\infty} \gamma^t R(s_t, a_t)|s_0 = s, \pi],$$

which is the total reward of the policy π starting from state s, and the state-action value function only depends on the state s. Besides, the state-action value function is

$$Q^\pi(s, a) = \mathbb{E}[\sum\nolimits_{t=1}^{\infty} \gamma^t R(s_t, a_t)|s_0 = s, a_0 = a, \pi],$$

which is the total reward starting from state s and action a.

For reinforcement learning, the transition function and the reward function are not explicitly known, but must be inferred from interaction experiences. Reinforcement learning approaches can be roughly categorized as model-based and model-free. Model-based approaches reconstruct the environment model (the transition function and the reward function) and then solve the policy from the model, with representatives as R-MAX [17], and Fitted R-MAX [18]. Model-free approaches do not explicitly model the environment. A major branch of model-free approaches estimates value functions, such as Q-Learning. Value function estimation approaches, while often converge fast, may suffer from the policy degradation problem [19]. Moreover, value-based methods are usually hard to apply to continuous states and actions in high dimensional spaces. Another branch of the model-free reinforcement learning, policy search, learns the policy directly by maximizing the total reward, thus can avoid the policy degradation problem.

B. Policy Search

Methods in this category search directly in a policy space to maximize the total reward. They can be implemented by gradient ascent methods, e.g., [15], [20], [21], and derivative-free optimization methods, e.g., [22].

In policy search, a policy is often represented as a parameterized model π_θ using a parameterized potential function $f(\cdot|\theta)$. In deep reinforcement learning, f is a neural network and θ denotes the weights of the neural network. Gibbs distribution is commonly used for discrete action space,

$$\pi_\theta(i|s) = \frac{exp(f_i(s|\theta]))}{\sum_j exp(f_i(s|\theta]))},$$

and Gaussian distribution is used for continuous action space,

$$\pi_\theta(a|s) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{1}{\sigma^2}(f(s|\theta) - a)^2).$$

The goal of policy search is to find the best parameters θ of a given policy model $\pi_\theta(s, a)$. To measure the quality of a policy π, the direct objective function for a policy search reinforcement learning is the total reward J, which can be equivalently rewritten as:

$$J(\theta) = \int_S d^{\pi_\theta}(s) \int_A \pi_\theta(a|s) Q^{\pi_\theta}(s, a) ds da$$

where d^{π_θ} is the stationary distribution of the process.

Methods optimizing the parameters θ include gradient based methods and derivative-free methods. Policy gradient searches for a local maximum by ascending the parameters following the gradient of the policy with respect to the expected reward. By the *policy gradient theorem* [23], the basic policy gradient method employs the direct gradient of the objective

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta log \pi_\theta(a|s) Q^{\pi_\theta}(s, a)]$$

and then the parameters θ are updated by $\theta = \theta + \eta \Delta_\theta J(\theta)$. Meanwhile, derivative-free methods employ derivative-free optimization approaches such as CMA-ES [24] and RACOS [22], [25] to approximate the best parameters. These methods work by sampling parameters θ's, and learn from the total reward of each θ, in order to find better samples of the parameters.

Policy search methods have been shown advantages in handling sophisticated problems such as tasks with continuous states and actions. Moreover, policy search can better avoid the policy degradation issue, and can effectively learn in high dimensional spaces.

C. Related Work

Traditional reinforcement learning problem focuses on a fixed MDP, or a fixed task as called in this paper, while we often face the problem of a range of tasks in practice. Transfer reinforcement learning, a combination of transfer learning and reinforcement learning, addresses the cross-task generalization problem. Transfer learning is a subfield of machine learning that reuses the knowledge from related source domains to help the learning in the target domain [26].

Transfer reinforcement learning reuses the experiences gained from previous tasks to help the learning in the target task [6]. The type of knowledge transferred can be low level knowledge, such as sample instances [7], [27], value functions [28], and policys [29]. Recently, the researches focus more on the question that what could be transferred in deep neural networks [9], [10], [30].

Among various transfer settings, the multi-task reinforcement learning is particularly related, which solves multiple tasks simultaneously by making use of assumptions that the tasks share similarity in some components of the problem such as dynamics, reward structure, or value function. The canonical multi-task reinforcement learning aims at solving a fixed set of MDPs simultaneously, and generalizes only within these tasks [11], [31]. Some of the recent methods try to solve this problem by introducing the LSTM architecture to adapt to new tasks automatically [32]. Differently, meta-policy aims at generalizing over a distribution of tasks, so that unseen tasks can also be solved without re-training.

There are also some studies under the term of multi-task reinforcement learning but do consider a task distribution. The parameterized skills learning [14] proposes to estimate the connection between the task parameters and the policy parameters, i.e., the generalization part of the meta-policy learning. It assumes that optimal policies are in hand for the training tasks, and thus ignores the nontrivial training stage that obtains these policies. While our proposed approach solves both the training and generalization problems in an integrated procedure. The generalization over a task distribution is also considered in [33], by using a predefined task relationship. There are also some methods towards meta-policy, but they do not consider the possible conflict between different tasks, and always assume that task parameters are known [34]. These methods could be limited in applications, as the task parameters are usually unknown. As a comparison, the approach proposed in this paper explores the task relationship and groups similar tasks to avoid performance degradation and can fit to situations when the task parameters are unknown.

III. Meta-Policy Learning via Shallow Trails

A. Problem Statement

Assume there is a space of MDPs, $\mathcal{M}$, and a fixed underlying distribution $\mathcal{D}$ over $\mathcal{M}$. An agent is given a set of training tasks $D = \{M_1, M_2, ..., M_m\}$ where each $M_i \in \mathcal{M}$ is drawn independently from $\mathcal{D}$. We assume in this paper that tasks are different only in the reward and the transition function, but share the same state and action spaces. The agent aims at deriving a policy π from the training tasks to maximize the total reward with respect to the distribution $\mathcal{D}$, i.e., to maximize

$$J = \int_{\mathcal{D}} p(M) J_M(\pi) dM,$$

where $J_M(\pi)$ is the total reward of policy π on MDP M. The criterion $J_M(\pi)$ can be defined in various ways, such as the long-term expected discounted reward,

$$J_M(\pi) = \mathbb{E}\{\sum\nolimits_{t=1}^{\infty} \gamma^{t-1} r^t | M, \pi\},$$

where r_t is the reward at step t, and γ is the discounted factor.

Maximization of J is hard to achieve directly, since it is infeasible to enumerate all tasks for approximating the integral. According to the learning theory [35], if a policy achieves a performance on a set of m training tasks, its performance of J is upper bounded by the training performance with an extra term depends on m and the policy model complexity. While the complexity can be maintained by techniques such as dropout [36], we focus on maximizing the total reward on the training tasks,

$$J_{tr}(\pi) = \sum\nolimits_{M \in D} J_M(\pi).$$

A direct way of maximizing the objective is to employ the policy search method. Considering the policy gradient method, the gradient of the objective with respect to the policy parameters θ can be written as:

$$\frac{\partial J_{tr}(\pi)}{\partial \theta} = \sum\nolimits_{M \in D} \frac{\partial J_M(\pi)}{\partial \theta}.$$

Then the overall gradient of our objective can be written as:

$$\frac{\partial J_{tr}(\pi_\theta)}{\partial \theta} = \sum_{M \in D} \sum_{s} d_M^{\pi_\theta}(s) \sum_{a} Q_M^{\pi_\theta}(s,a) \frac{\partial \pi_\theta(s,a)}{\partial \theta}, \quad (1)$$

where $d_M^{\pi_\theta}(s)$ is the stationary distribution of states following π_θ on the task M, $Q_M^{\pi}(s,a)$ is the usual state-action value function on task M. Note that the value of $Q_M^{\pi_\theta}(s,a)$ is unknown, but can be approximated by its unbiased estimation, or using some function approximation method as in [23].

Once the above gradient can be estimated, we can maximize the objective by the gradient ascent update rule, $\theta_t = \theta_{t-1} + \alpha_t \cdot \partial J_{tr}/\theta_t$ where α_t is the step size. In addition to policy gradient, other policy search methods can share the similar idea to maximize the total reward over the task distribution.

However, although Eq.(1) can be used to update the policy parameters, the performance of the policy is limited if it cannot distinguish which task a state belongs to. Therefore, it is necessary to obtain task-related information for learning a meta-policy.

B. Policy Search with Task Features

We consider parameterized task distributions [8], [14], where each task has several observable features, i.e., every MDP M corresponds to a vector $\tau \in T$ drawn from distribution $\mathcal{D}(\tau)$, where the features can essentially capture the MDP. It is obvious that if the task parameters, such as the target positions of the GridWorld [1], are known to us, they can be easily used to represent the task features. Then the policy function can be conveniently extended to take state, action and task features into consideration as $\pi_\theta(s,a,\tau): S\times A\times T\rightarrow[0,1]$. With task features as part of the policy input, the meta-policy can be learned over a range of tasks. While the policy is a probability distribution $\pi_\theta(a|s,\tau)$, it can be parameterized with a linear or nonlinear model. Considering a continuous action space, the extended policy model is

$$\pi_\theta(a|s,\tau)=\frac{1}{\sqrt{2\pi\sigma^2}}exp(-\frac{1}{\sigma^2}((f(s,\tau|\theta)-a)^2)$$

which allows the policy to perform task-aware actions. Similar to the fixed task setting, the direct objective for this situation would be:

$$J(\theta)=\int_T p(\tau)\int_S d^{\pi_\theta}(s,\tau)\int_A \pi_\theta(a|s,\tau)Q^{\pi_\theta}(s,a,\tau)dadsd\tau$$

It's obvious that current off-the-shelf policy search algorithms can be easily adopted to solve this objective function.

C. Shallow Trail

If we simply adopt the policy search framework with the task features being part of policy input without other supplement, there would still be some obstacles to achieve a good performance. In meta-policy learning, we could face a very large task space. In such a space, there can be tasks that have very different or even conflicting goals. It is unwise to build only one policy model for all conflicting tasks, which may lead to a bad performance. This situation requires us to automatically discover the similarities and difference among tasks, and build policies each for a group of similar tasks. Therefore, it becomes critical to measure the similarities between the tasks before grouping tasks.

Moreover, meta-policy learning requires task features as part of the policy input. If we already know the parameters of these tasks, they can easily be applied to represent the task features. However, in many real environments, we often fail to get the parameters. How to adapt meta-policy learning mechanism in such situations becomes a key issue. If some task-corresponding variables can distinguish different tasks, they may be a good choice as task features.

Considering these two problems together, we need to find useful features to distinguish and represent tasks. Ideally, the similarities of tasks can be measured by the reward of executing the optimal policy of one task on the other task [13]. Although it is infeasible to solve the optimal policy of a test task for obtaining the features, this inspires us a more practical idea to measure the similarities among tasks: similar tasks would give similar relative rewards to the same policy, i.e., if a task gives policy A a higher total reward than policy B, a similar task would also prefer A to B, which gives us more information about these tasks and can help to comprise task features. Note that A and B are not necessarily optimal policies.

Therefore, we propose the *shallow trail* trick. Suppose that there are m tasks. Firstly, in order to discover the characteristics of the tasks, we run a policy search method with a very small number of iterations for each task to obtain prototype policies $\{\tilde{\pi}_1,\tilde{\pi}_2,\tilde{\pi}_3,...\tilde{\pi}_m\}$. The number of iterations is called the shallow trail *depth*. Secondly, we assign each task a episode reward vector, where each element of the vector is the total reward received by a prototype policy on the task, i.e., for task i the episode reward vector is

$$v_i=(R(\tau_i|\tilde{\pi}_1),R(\tau_i|\tilde{\pi}_2),R(\tau_i|\tilde{\pi}_3),...R(\tau_i|\tilde{\pi}_m))$$

We denote this process as shallow trail, as for all tasks including test tasks, we only collect the rewards from running these prototype policies, instead of solving the optimal policies.

D. Task Grouping

Since tasks can be very different or even conflicting in a large task space, it is necessary to put dissimilar tasks in different groups. The reward vectors obtained by shallow trails can serve this purpose well, as they are associated with rewarding properties of the tasks. We employ the cosine distance of reward vectors as the similarity between tasks.

With the similarity function, we can group the tasks via clustering algorithms, e.g., the K-means algorithm [37] and the K-medoids algorithm [38]. After the grouping, we need also assign test tasks to groups. Therefore, we build a mapping from task feature vector to group number by learning a classifier. Again, we use either task parameters or shallow trail outcome as task features.

Another question is that how many groups of these tasks should be clustered into, which could greatly affect the quality of the meta-policy. To determine the number of clusters. we employ the Distance-Dependent Chinese Restaurant Process (dd-CRP) [39], which is described as follows. A customer comes to a Chinese restaurant and sits down alone at a new table with a probability proportional to α and sits down with another customer at his table with a probability proportional to their similarities. After all customers have sat down, the number of used tables can be used to approximate the number of underlying clusters of customers. To apply dd-CRP, we set the probability that task i sits with task j as

$$p_{i,j}\propto\begin{cases} exp(-||v_i-v_j||_2/\rho), if & j\neq i\\ \alpha, \qquad if & j=i\end{cases}$$

where v_i is the normalized reward vector of task i, v_j is that of task j, ρ is a parameter that controls the importance of the distance, and α controls the chance of sitting on a new table. With the probability, we then sample the process by scanning the tasks and determining if two tasks are connected (sit in the same table), and find the number of tables. The process is sampled several times to obtain an average number, which is used as the number of clusters.

The objective is transformed as

$$\begin{aligned}J(\theta)=\int_T p(v_\tau)\int_S d^{\pi_\theta}(s,v_\tau)\\ \int_A \pi_\theta(a,|s,v_\tau)Q^{\pi_\theta}(s,a,v_\tau)dadsdv_\tau.\end{aligned}$$

where v_τ is the episode reward vector of the task τ obtained by shallow trail before.

Algorithm 1 MAPLE-P : Training

```
Input:
    $\mathcal{L}$: A policy search algorithm
    $\mathcal{D} = \{\tau_i\}_{i=1}^m$: parameters of the training tasks
    $L_c$: A classification algorithm
    $T$: Number of iterations
    $N$: Shallow trail depth
    $\rho, \alpha$: Parameters of dd-CRP
Output:
    $\pi_{meta}$ : The meta-policy
    $\Psi(\tau)$: A classifier
1:  $\forall i = 1, 2, ..., m : \tilde{\pi}_i \leftarrow$ run $\mathcal{L}$ on task $i$ with $N$ iterations
2:  $\forall i = 1, 2, ..., m : v_i \leftarrow (R(\tilde{\pi}_1, \tau_i), ..., R(\tilde{\pi}_m, \tau_i))$,and
    $v_i \leftarrow v_i/||v_i||_2$, i.e., normalized reward vector
3:  $K \leftarrow$ dd-CRP$(v, \rho, \alpha)$
4:  $\{G_1, G_2, ..., G_K\} \leftarrow$ groups by k-means$(v, K)$
5:  $\Psi \leftarrow$ run $L_c$ on dataset $\{(\tau_i, k)|\tau_i \in G_k\}$
6:  $\forall k = 1, 2, ...K$ : initialize $\theta_0^k$
7:  for $t \leftarrow 1$ to $T$ do
8:      for $k \leftarrow$1 to $K$ do
9:          $J(\theta_t^k) = \int_{S,T} p(\tau) d^{\pi_{\theta_t^k}}(s, \tau) \int_A \pi_{\theta_t^k}(a|s, \tau)$
            $\cdot Q^{\pi_{\theta_t^k}}(s, a, \tau) \mathrm{d}a \mathrm{d}s \mathrm{d}\tau$
10:         $\theta_{t+1}^k = \arg\max_\theta J(\theta_t^k)$ by $\mathcal{L}$
11:     end for
12: end for
13: return $\pi_{meta}(s, a, \tau) = \sum_{k=1}^K \pi_{\theta_T^k(s,a,\tau)} \cdot I_{\Psi(\tau)=k}$
```

Algorithm 2 MAPLE-P: Reusing

```
Input:
    $\tau$ : The Parameter of the test task
    $\Psi$: The classifier trained in Algorithm 1
Output:
    $\pi_{meta}$ : The meta-policy
1:  Run classifier $\Psi$ on task parameters, i.e., $k = \Psi(\tau)$
2:  Run $\pi_{\theta_T^k(s,a,\tau)}$
```

E. MAPLE

Combining the components above, we proposed the MAPLE (MetA-Policy LEarning) framework. By MAPLE, an agent can learn a qualified meta-policy that could be able to efficiently adapt to a range of tasks instead of a fixed task even when the task parameters are unknown.

1) MAPLE-P Algorithm: If we assume that task parameters are known to us, we can directly use these parameters as task features to learn meta-policy. In this situation, our algorithm is denoted as MAPLE-P as in Algorithm 1.

MAPLE-P is given a set of training tasks, where we denote τ as a mixed meaning of the task and task parameters, and output a meta-policy. The input of MAPLE-P include a policy search algorithm, a classifier algorithm which can be an arbitrary state-of-the-art algorithm, the number of iterations T which can be tens to hundreds depending on the difficulty of the tasks, the number of iteration N for shallow trail to obtain prototype policies, ρ and α are the parameters of dd-CRP.

At the beginning, MAPLE-P obtains prototype policies by a rough training with a small number of iterations of policy search algorithm on each of the training task (line 1), and run all of these policies on each training task to obtain the normalized reward vectors (line 2). Although we have the ground-truth task parameters as the task features, the distance function based on task parameters may not really reflect the similarity. Thus we use the reward vectors for similarity measurement. We employ dd-CRP for hundreds of times, and then we can determine the number of groups K (line 3). After getting K, the K-means algorithm could be employed (for its efficiency and commonness, but other clustering algorithms are not excluded) to cluster tasks (line 4). A grouping classifier is trained (line 5), which will be used in the meta-policy to tell which group a task belongs.

Algorithm 3 MAPLE-R: Training

```
Input:
    $\mathcal{L}$: A policy search algorithm
    $D = \{\tau_i\}_{i=1}^m$: Training tasks
    $L_c$: A classification algorithm
    $T$: Number of iterations
    $N$: Shallow trail depth
    $\rho, \alpha$: Parameters of dd-CRP
Output:
    $\pi_{meta}$ : The meta-policy
    $\Psi(\tau)$: The classifier
    $\tilde{\pi}_1, \tilde{\pi}_2, ...\tilde{\pi}_m$ : $m$ prototype polices
1:  $\forall i = 1, 2, ..., m : \tilde{\pi}_i \leftarrow$ run $\mathcal{L}$ on task $i$ with $N$ iterations
2:  $\forall i = 1, 2, ..., m : v_i \leftarrow (R(\tilde{\pi}_1, \tau_i), ..., R(\tilde{\pi}_m, \tau_i))$, and
    $v_i \leftarrow v_i/||v_i||_2$, i.e., normalized reward vector
3:  $K \leftarrow$ dd-CRP$(v, \rho, \alpha)$
4:  $\{G_1, G_2, ..., G_K\} \leftarrow$ groups by k-means$(v, K)$
5:  $\Psi \leftarrow$ run $L_c$ on dataset $\{(v_i, k)|v_i \in G_k\}$
6:  $\forall k = 1, 2, ...K$ : initialize $\theta_0^k$
7:  for $t \leftarrow 1$ to $T$ do
8:      for $k \leftarrow$1 to $K$ do
9:          $J(\theta_t^k) = \int_{S,T} p(v_\tau) d^{\pi_{\theta_t^k}}(s, v_\tau) \int_A \pi_{\theta_t^k}(a|s, v_\tau)$
            $\cdot Q^{\pi_{\theta_t^k}}(s, a, v_\tau) \mathrm{d}a \mathrm{d}s \mathrm{d}v_\tau$
10:         $\theta_{t+1}^k = \arg\max_\theta J(\theta_t^k)$ by running $\mathcal{L}$
11:     end for
12: end for
13: return $\pi_{meta}(s, a, v) = \sum_{k=1}^K \pi_{\theta_T^k(s,a,v)} \cdot I_{\Psi(v)=k}$
```

Algorithm 4 MAPLE-R: Reusing

```
Input:
    $\tau$: The test task
    $\Psi$: The classifier trained in Algorithm 3
    $\tilde{\pi}_1, \tilde{\pi}_2, ...\tilde{\pi}_m$ :The prototype policies trained in Algo-
    rithm3
Output:
    $\pi_{meta}$ : The meta-policy
1:  $v \leftarrow (R(\tilde{\pi}_1, \tau), ..., R(\tilde{\pi}_m, \tau))$and $v \leftarrow v/||v||_2$
2:  Run classifier $\Psi$ on task features of $\tau$, i.e., $k = \Psi(v)$
3:  Run $\pi_{\theta_T^k(s,a,v)}$ on task $\tau$
```

The policy optimization procedure starts form line 6. It initializes the policy parameters at the first iteration and iterates for T times. We train one sub-policy for one group by the policy search algorithm $\mathcal{L}$. Note that the Q function in line 9 can be replaced by the advantage function $A = Q - V$ for some policy search algorithms. Finally, the meta-policy is a consensus of the sub-policies (line 13), where $I_{expression}$ is 1 if the expression is true and 0 otherwise.

After learning the meta-policy, the process of reusing the meta-policy for an unseen task is shown in Algorithm 2. First, the trained L_c determines which group the current task belongs to. The agent then chooses the sub-policy corresponding to its group. Finally, the appropriate sub-policy runs for the task.

2) MAPLE-R Algorithm: Under the circumstances where we do not know the task parameters, we cannot use MAPLE-P. MAPLE-R is then proposed to uses the episode reward vectors to replace the task parameters. MAPLE-R is depicted in Algorithm 3. It is similar to Algorithm 1, but the task parameters are not needed. Note that the group classifier trained in Algorithm 3 also uses the reward vectors.

The same as MAPLE-P, MAPLE-R could also be applied to unseen tasks, which is shown in Algorithm 4. But different from MAPLE-P, we can not directly determine which group a new task belongs to, as the tasks parameters are not available. Therefore, in MAPLE-R, we need to record all the prototype policies after the learning stage. For an unseen task, we run each of the prototype policy to obtain the reward vector. Then the reward vector is used as the feature of the new task for by the grouping classifier and the meta-policy.

IV. Experiments

We empirically evaluate MAPLE, particularly, addressing the following questions:

$Q1$ Does MAPLE fit the goal of meta-policy learning that maximizes the total reward over the task distribution?

$Q2$ How is the generalization performance of MAPLE on unseen tasks?

$Q3$ How do the hyper-parameters effect the MAPLE, including the shallow trail depth, task grouping, and the number of training tasks?

$Q4$ How if MAPLE serves as the initial policy for further task-specific training on unseen tasks?

A. Experiment Settings

We employ six environments in the MuJoCo simulator [40] of OpenAI Gym to evaluate the algorithms, including Swimmer, HalfCheetah, Walker2d, Ant, Humanoid and Humanoid Standup. The state space, actions, and reward function are the original ones provided in the MuJoCo environments. All the tasks require to control an agent to move, where the agent are all composited by legs and joints. To create a set of tasks from each environment, we vary some parameters, including the length of legs and the range that legs can cross. The detail setting is described in the following, where the meaning of the variables can be found in the MuJoCo simulator.

Swimmer is to train a 3-link swimming robot in a viscous fluid to swim by actuating the two joints. The task distribution is created by varying "rot2" in $[-129, 80]$ and "rot3" in $[80, 129]$ uniformly at random;

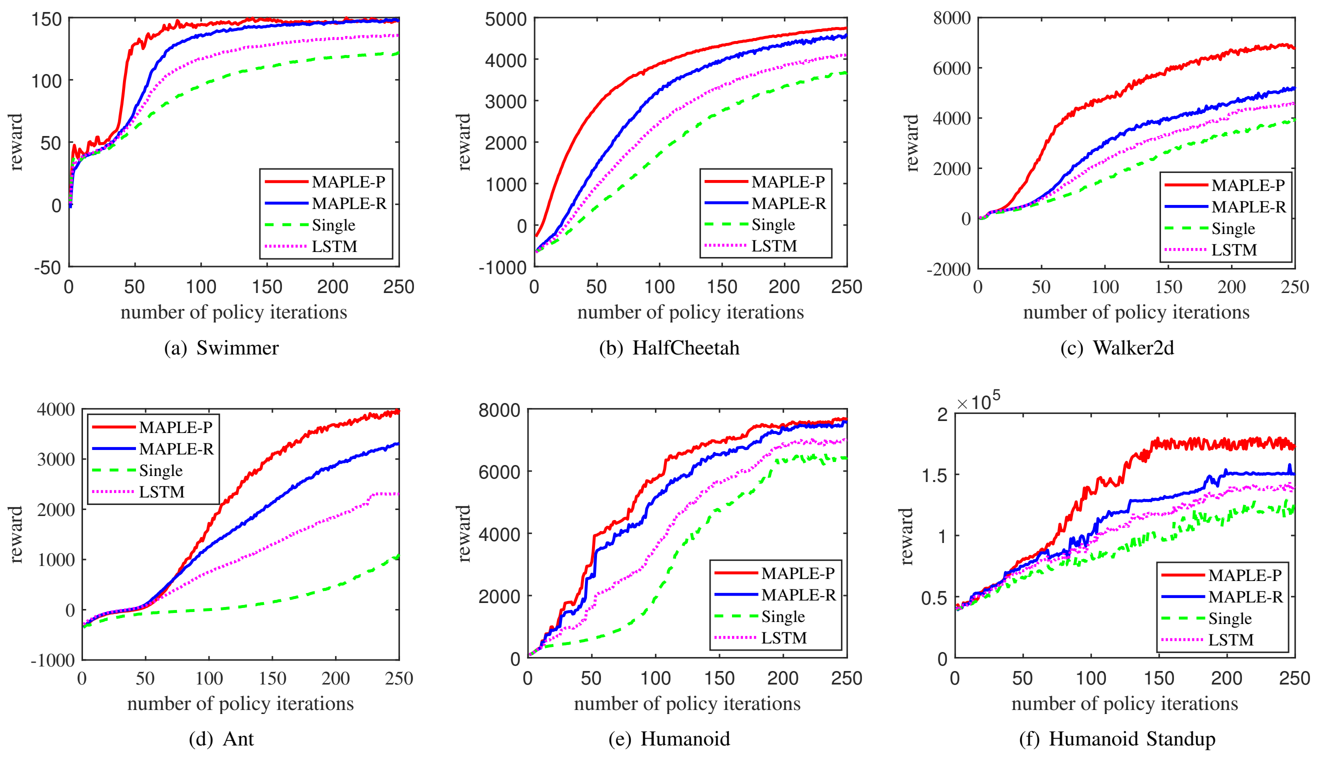


Fig. 1. Performance comparison on training tasks. The x-axis is the number of iterations of policy training.

HalfCheetah is to train a 2d Cheetah model to run. The environment distribution is created by varying "bthigh" from $(-1, 1.45)$ to $(0.5, 2.95)$ uniformly at random;

Walker2d is to train a bipedal model with two legs to walk forward. The task distribution is created by varying "thigh_left_joint" and "leg_left_joint" in $[-150, -135]$, "foot_left_joint" in $[-70, 45]$, and "foot_left_joint" in $[-45, 70]$ uniformly at random;

Ant is to train a four-legged creature to walk forward. The task distribution is created by varying "ankle_2" and "ankle_3" in $[-55, -64]$ uniformly at random;

Humanoid is to train a three-dimensional bipedal robot to walk forward. The task distribution is created by varying "right_hip_x" from $(-29, -20)$ to $(5, 14)$ uniformly at random;

Humanoid Standup is to train a three-dimensional bipedal robot to stand up. The task distribution is created by varying "right_hip_x" from $(-29, -20)$ to $(5, 14)$ uniformly at random.

The base reinforcement learning method is TRPO [15], which has been shown one of the best method for the controlling tasks. We will also compare with TRPO alone on each seen task, denoted as "Single". Besides, we compare with two cross-task methods, the "Neighbor" method that, for a test task, uses the policy from the nearest training task; employing an LSTM model has recently been shown to be able to adapt from training tasks to test tasks [41], which is compared and denoted as LSTM.

For all of the six environments, the discount factor γ is 0.99. Similar as in [42], the policy model is a fully connected neural networks with two hidden layers each has 64 nodes. For MAPLE, the parameters of dd-CRP are 0.05 for α and 0.1 for ρ in all environments, the classification learner L_c is the random forest [43] for its advantage of less parameter tuning. The default depth of the shallow trails is 2 on each training task except for the experiments investigating this parameter.

B. Experiment Results

To address $Q1$, we first compare MAPLE with Single as well as LSTM on training tasks. 40 training tasks are sampled for each environment. Figure 1 shows the training performance, measured by the total reward averaged over all tasks, along with the training iterations. It can be observed that the four methods have the consistent rank of performance in all the environments. Single has the worst performance, as it trains on each task separately, indicating that information sharing among tasks can be helpful even at the training stage. MAPLE-P has the best performance, as it knows the ground-truth task parameters. MAPLE-R has an inferior performance to MAPLE-P as it does not use the ground-truth task parameters but shallow trail vector, while it still performs better than LSTM, which implies that the shallow trail can provide more useful task information than LSTM.

To address $Q2$, we then compare the methods on test tasks, after training on 40 training tasks. The policies trained by Single are not able to be used on test tasks directly, thus Neighbor is used instead which uses the task parameters to determine the nearest training task of a test task. Figure 2 shows the total reward averaged over all test tasks. The rank of the performance of the methods is consistent with the rank on the training tasks. Moreover, we can observe that Neighbor has a quite bad performance. It almost has no improvement on the total reward, implying that the policy learned by Single cannot generalize well to the neighbor tasks in the task parameter space. We also notice that the test task performance of MAPLE-P is very similar to its training task performance on Swimmer, HalfCheetah, Walker2d, and Ant, while the test task performance drops significantly on the more difficult tasks Humanoid and Humanoid Standup. But without knowing the ground-truth task parameters, MAPLE-R and LSTM both have dropped performance from training tasks to test tasks in all environments, while MAPLE-R is still significantly better than LSTM, implying the effectiveness of shallow trails.

To address $Q3$, we investigate the effect of each parameter. Figure 3 shows the performance with different shallow trail depth. It is consistent that deeper trail leads to better informative vector. Meanwhile, using deep trails increases the training cost and also raises the risk of overfitting.

We then investigate the usefulness of task grouping. By "NoGrouping", the method switched off the grouping process, and only one policy is trained over all training tasks, while the suffix "P" or "R" indicates that they are based on MAPLE-P and MAPLE-R respectively. From the results in Figure 4 we can observe that NoGrouping is consistently worse than MAPLE, implying putting too many tasks under one policy degrades the performance. Thus grouping is necessary.

We also investigate the effect of the number of training tasks. We show the results of comparing training tasks from 5 to 40 in Figure 5 for training task performance and Figure 6 for test task performance, where the subscript 100 or 200 indicates the number of training iterations. We firstly observe that increasing the training tasks is an effective way to improve the performance for MAPLE and LSTM, as they are improved on both training and test tasks when the number of training tasks increases. We also observe that the performance rank is consistent as the number of training tasks varies.

Finally, to address $Q4$, on test tasks we continue training the adapted policy by TRPO. The results are shown in Figure 7, where "None" denotes using a random policy as initial policy. We can observe that MAPLE-P and MAPLE-R lead to a better starting performance and can be further improved, which keep the best performance.

Moreover, we can observe that the ending points of None, the trained-from-scratch policies, have worse performance that the starting points of MAPLE, the adapted policies, on all environments except Humanoid Standup. This disclose that the obtained meta-policies can have strong performance even without test-task-specific training.

V. Conclusion

This paper proposes the MAPLE approach with the aim of learning a meta-policy that can be reused in unseen tasks. To distinguish the tasks and measure the similarity of two tasks, we introduce the method of shallow trail that probe

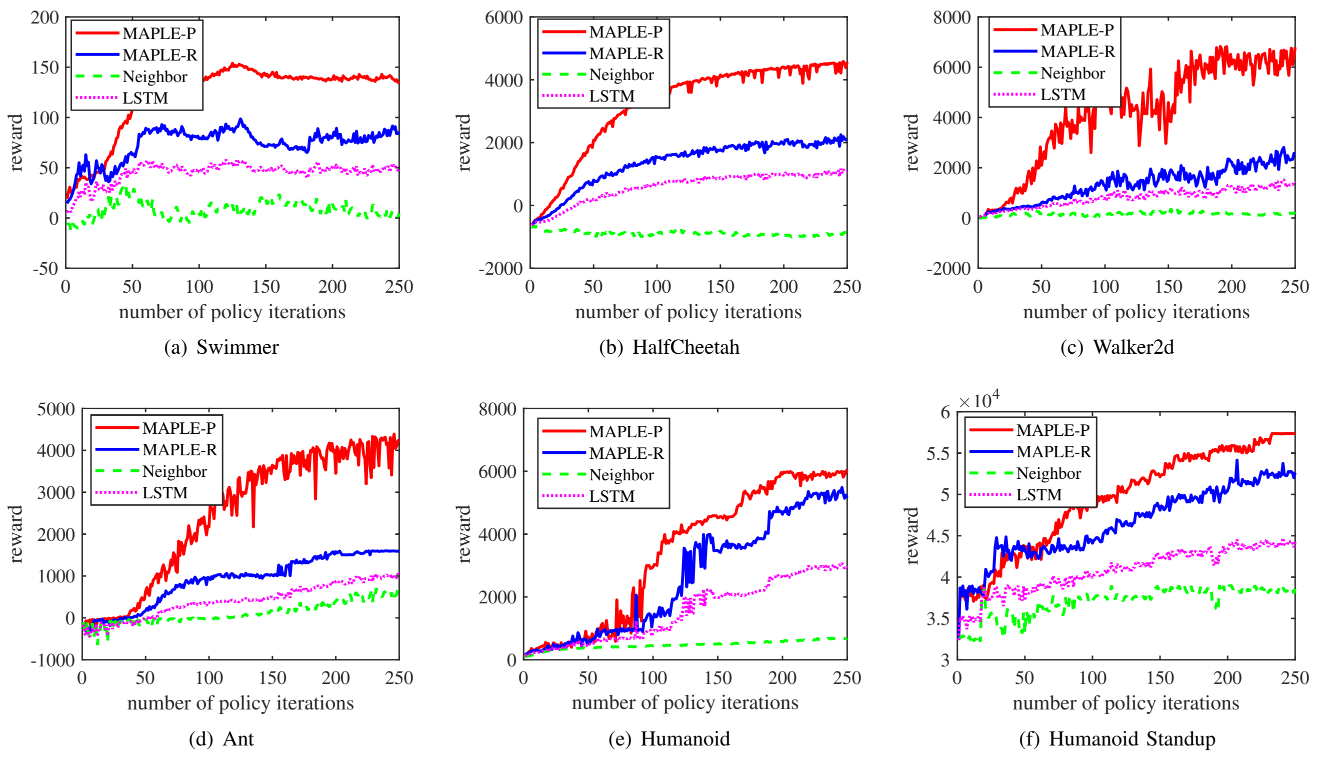


Fig. 2. Performance comparison on test tasks. The x-axis is the number of policy iterations at the training stage.

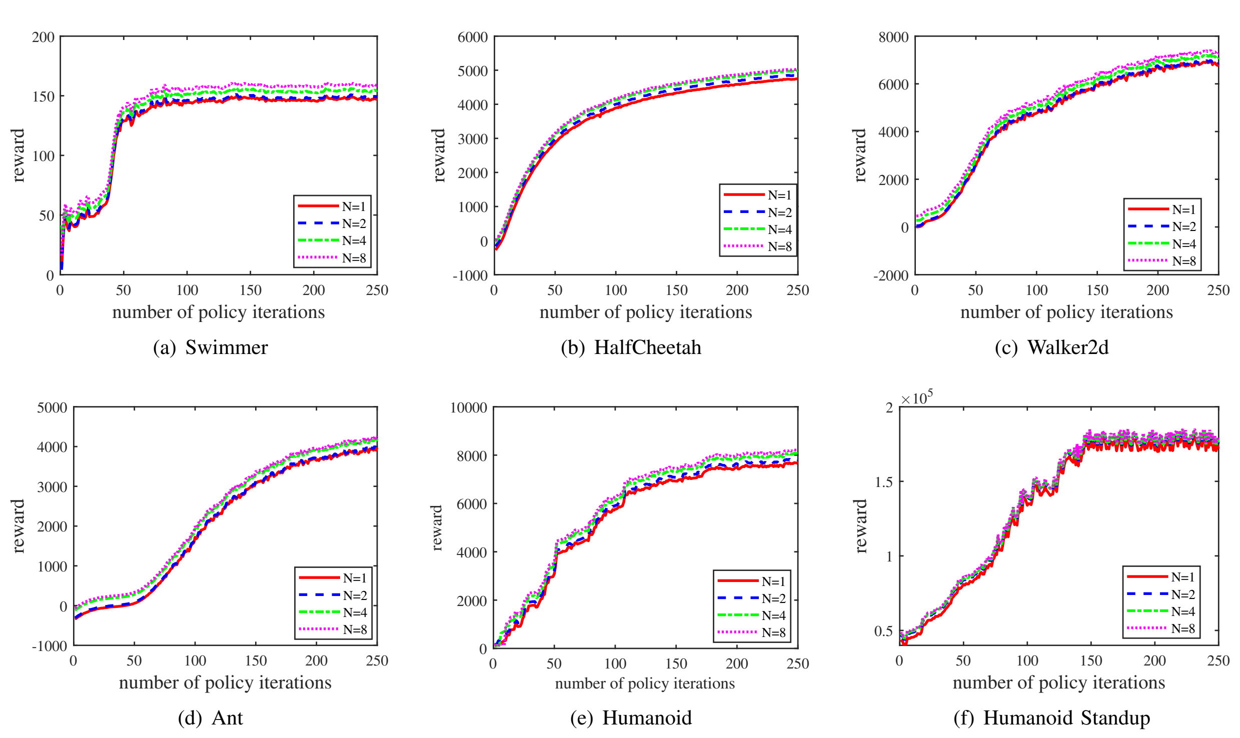


Fig. 3. Performance of different shallow trail depth N of MAPLE.

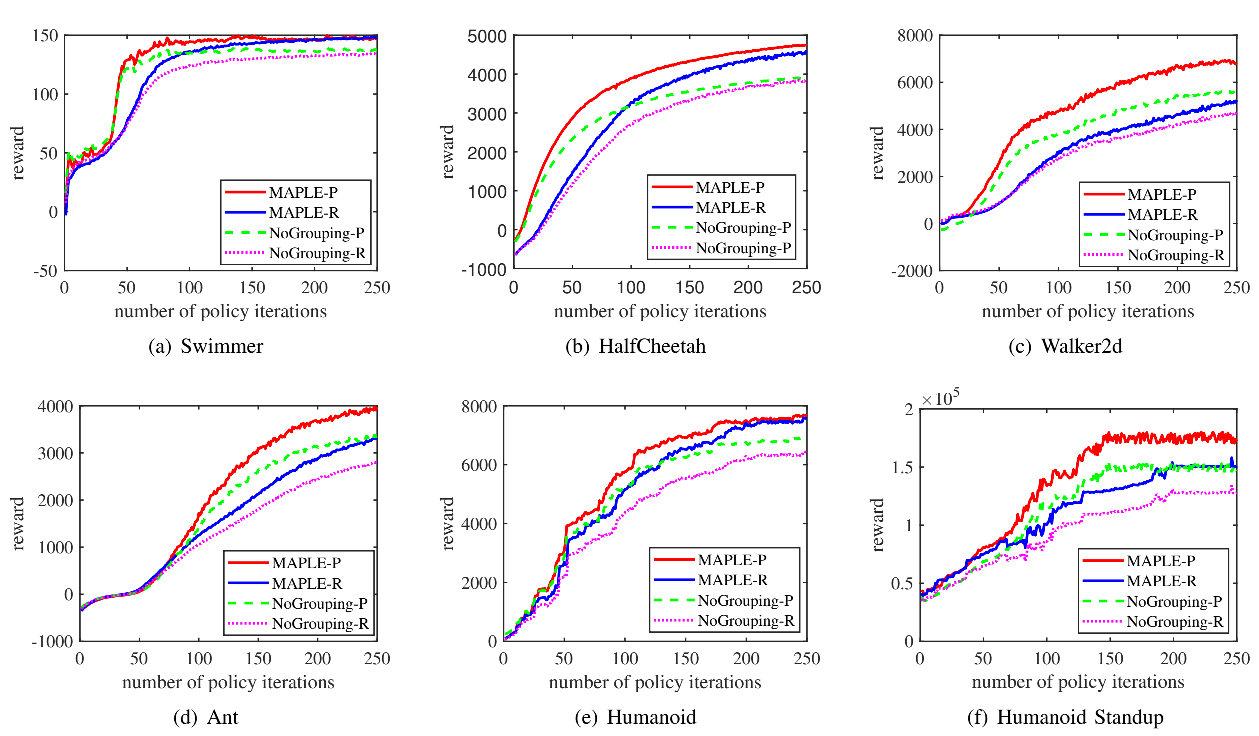


Fig. 4. On the effect of task grouping in MAPLE.

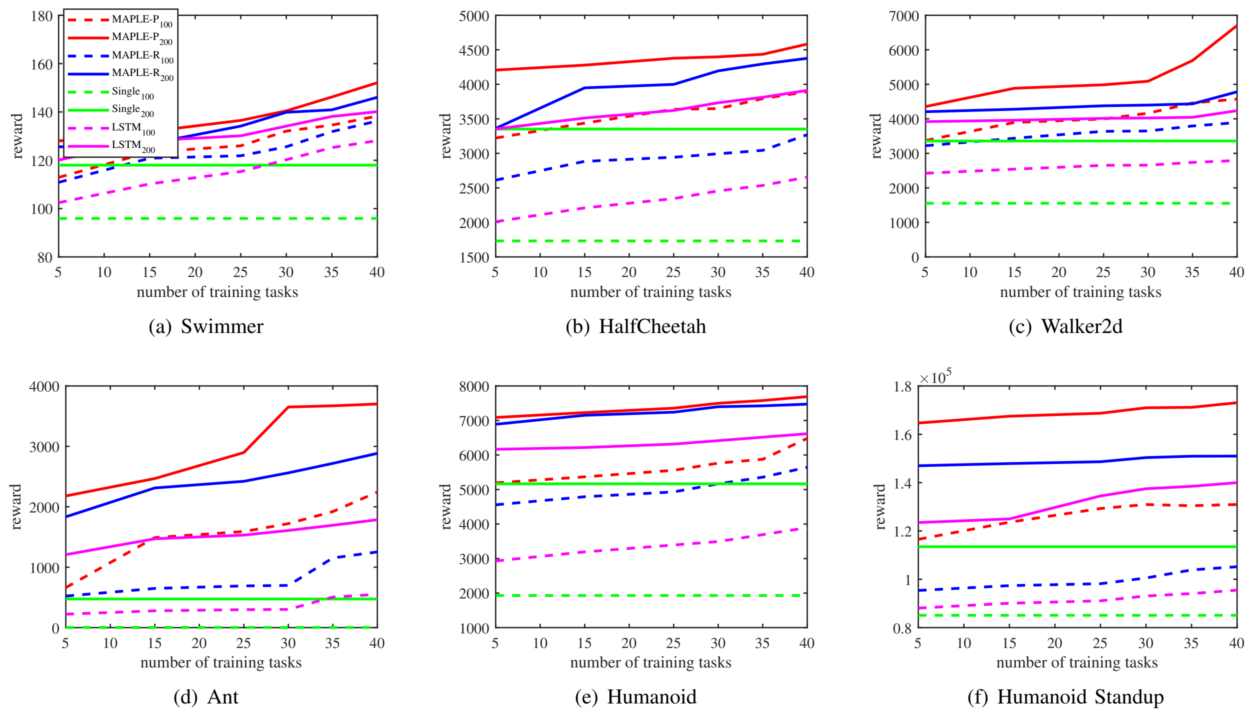
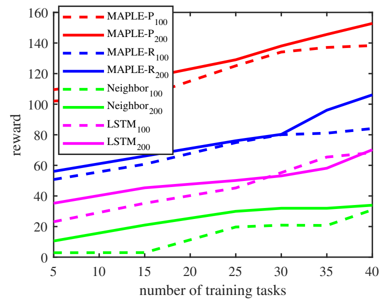
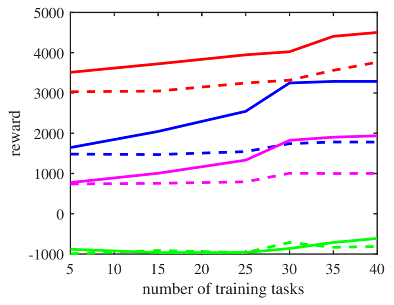


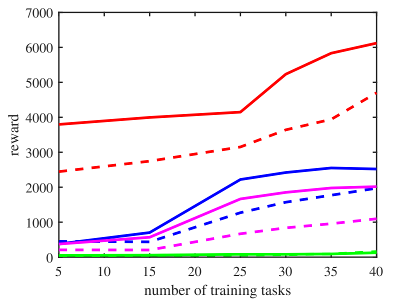
Fig. 5. Performance on training tasks of approaches with different numbers of training tasks.



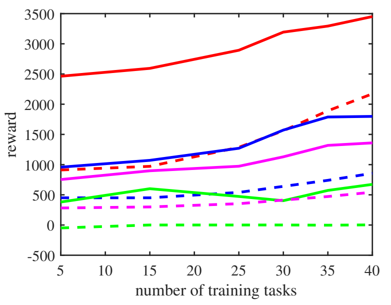
(a) Swimmer



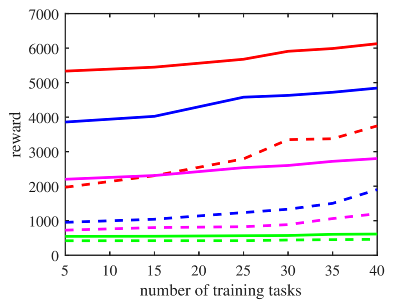
(b) HalfCheetah



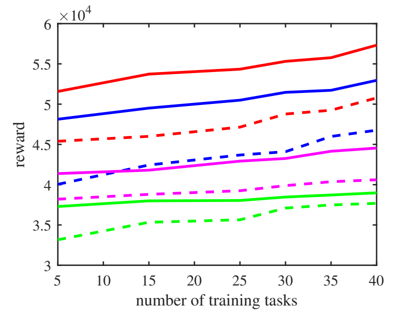
(c) Walker2d



(d) Ant

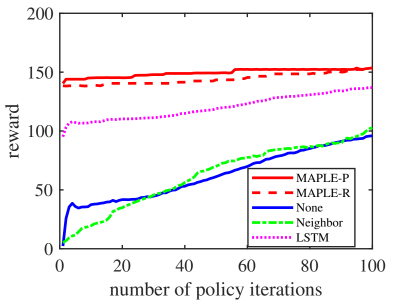


(e) Humanoid

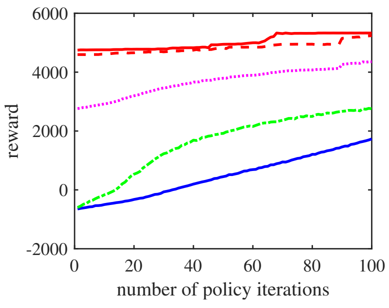


(f) Humanoid Standup

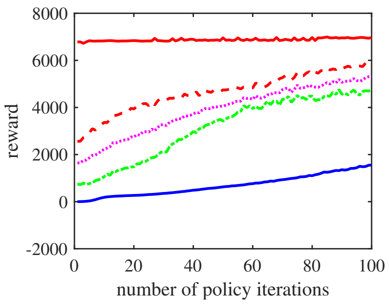
Fig. 6. Performance on test tasks of approaches with different numbers of training tasks.



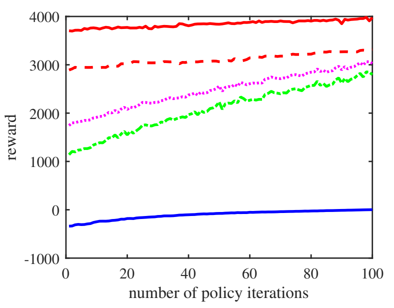
(a) Swimmer



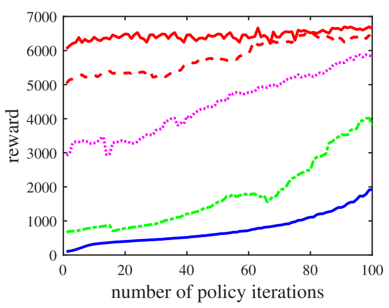
(b) HalfCheetah



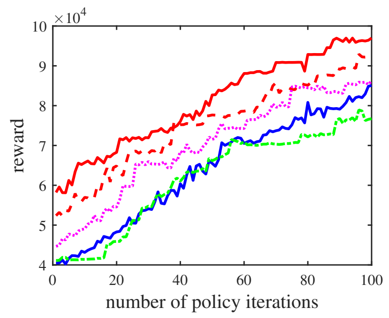
(c) Walker2d



(d) Ant



(e) Humanoid



(f) Humanoid Standup

Fig. 7. Performance of training on test tasks from scratch (None), with initialization by Neighbor, with initialization by LSTM and with initialization by MAPLE.

the property of a task by running roughly trained prototype policies which is obtained through a policy search algorithm for a few iterations. MAPLE solves the issue of possible negative transfer when some tasks are irrelevant or opposite in large task space, by restricting a policy within a group of similar tasks based on the shallow trails. Besides, when the task parameters are unknown, the outcome of shallow trails can be directly used to represent the task features. As a result, MAPLE can learn meta-policies that can be well reused in unseen tasks, even if the task parameters are not available.

Experiment results verify that MAPLE evidently has a good performance on both training and test tasks. Moreover, the performance of MAPLE improves as the number of training tasks grows, while no performance degradation is observed for MAPLE, since it the task similarity is measured and managed in groups based on the shallow trails.

MAPLE is restricted to be reused in tasks with the same state and action spaces. In the future, we will study how to reuse a meta-policy to fit changed state and action spaces, approaching the goal of learnware [5].

ACKNOWLEDGMENT

The authors want to thank Dr. Zongzhang Zhang for his helpful comments. This research was supported by the National Science Foundation of China (61751306) and Jiangsu Science Foundation (BK20170013).

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[2] Z.-H. Zhou, "A brief introduction to weakly supervised learning," *National Science Review*, 2018. [Online]. Available: https://doi.org/10.1093/nsr/nwx106

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[5] Z.-H. Zhou, "Learnware: On the future of machine learning," *Frontiers of Computer Science*, vol. 10, no. 4, pp. 589–590, 2016.

[6] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, vol. 10, pp. 1633–1685, 2009.

[7] A. Lazaric and M. Restelli, "Transfer from multiple MDPs," in *Advances in Neural Information Processing Systems 24*, Granada, Spain, 2011, pp. 1746–1754.

[8] N. Mehta, S. Natarajan, P. Tadepalli, and A. Fern, "Transfer in variable-reward hierarchical reinforcement learning," *Machine Learning*, vol. 73, no. 3, pp. 289–312, 2008.

[9] E. Parisotto, L. J. Ba, and R. Salakhutdinov, "Actor-Mimic: Deep multitask and transfer reinforcement learning," *CoRR*, vol. abs/1511.06342, 2015. [Online]. Available: http://arxiv.org/abs/1511.06342

[10] R. Glatt, F. L. da Silva, and A. H. R. Costa, "Towards knowledge transfer in deep reinforcement learning," in *Proceedings of the 5th Brazilian Conference on Intelligent Systems*, Recife, Brazil, 2016, pp. 91–96.

[11] A. Lazaric and M. Ghavamzadeh, "Bayesian multi-task reinforcement learning," in *Proceedings of the 27th International Conference on Machine Learning*, Haifa, Israel, 2010, pp. 599–606.

[12] A. Mujika, "Multi-task learning with deep model based reinforcement learning," *CoRR*, vol. abs/1611.01457, 2016. [Online]. Available: http://arxiv.org/abs/1611.01457

[13] L. Torrey and J. Shavlik, "Transfer learning," *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, vol. 1, pp. 242–264, 2009.

[14] B. C. da Silva, G. Konidaris, and A. G. Barto, "Learning parameterized skills," in *Proceedings of the 29th International Conference on Machine Learning*, Edinburgh, UK, 2012.

[15] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning*, Lille, France, 2015, pp. 1889–1897.

[16] Z. Zhang, Q. Fu, X. Zhang, and Q. Liu, "Reasoning and predicting POMDP planning complexity via covering numbers," *Frontiers of Computer Science*, vol. 10, no. 4, pp. 726–740, 2016.

[17] R. I. Brafman and M. Tennenholtz, "R-MAX - A general polynomial time algorithm for near-optimal reinforcement learning," *Journal of Machine Learning Research*, vol. 3, pp. 213–231, 2002.

[18] N. K. Jong and P. Stone, "Model-based exploration in continuous state spaces," in *Proceedings of the 7th International Symposium on Abstraction, Reformulation, and Approximation*, Whistler, Canada, 2007, pp. 258–272.

[19] J. Baxter and P. L. Bartlett, "Infinite-horizon policy-gradient estimation," *Journal of Artificial Intelligence Research*, vol. 15, pp. 319–350, 2001.

[20] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

[21] K. Kersting and K. Driessens, "Non-parametric policy gradients: a unified treatment of propositional and relational domains," in *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008, pp. 456–463.

[22] Y. Hu, H. Qian, and Y. Yu, "Sequential classification-based optimization for direct policy search," in *Proceedings of the 21th AAAI Conference on Artificial Intelligence*, San Francisco, CA, 2017, pp. 2029–2035.

[23] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems 12*, Denver, CO, 1999, pp. 1057–1063.

[24] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[25] H. Qian, Y. Hu, and Y. Yu, "Derivative-free optimization of high-dimensional non-convex functions by sequential random embeddings," in *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, New York, NY, 2016, pp. 1946–1952.

[26] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[27] M. E. Taylor, N. K. Jong, and P. Stone, "Transferring instances for model-based reinforcement learning," in *Proceedings of the 2008 European Conference Machine Learning and Knowledge Discovery in Databases, Part II*, Antwerp, Belgium, 2008, pp. 488–505.

[28] M. Sharma, M. P. Holmes, J. C. Santamaría, A. Irani, C. L. I. Jr., and A. Ram, "Transfer learning in real-time strategy games using hybrid CBR/RL," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 2007, pp. 1041–1046.

[29] F. Fernández and M. M. Veloso, "Probabilistic policy reuse in a reinforcement learning agent," in *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, Hakodate, Japan, 2006, pp. 720–727.

[30] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *Advances in Neural Information Processing Systems 27*, Montreal, Canada, 2014, pp. 3320–3328.

[31] H. Li, X. Liao, and L. Carin, "Multi-task reinforcement learning in partially observable stochastic environments," *Journal of Machine Learning Research*, vol. 10, pp. 1131–1186, 2009.

[32] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," in *Proceedings of the 1st Annual Conference on Robot Learning*, Mountain View, CA, 2017, pp. 262–270.

[33] M. P. Deisenroth, P. Englert, J. Peters, and D. Fox, "Multi-task policy search for robotics," in *Proceedings of 2014 IEEE International Conference on Robotics and Automation*, Hong Kong, China, 2014, pp. 3876–3881.

[34] J. Park, G. Yoo, and E. Lee, "Proactive self-healing system based on multi-agent technologies," in *Proceedings of the 3rd ACIS International*

Conference on Software Engineering Research, Management and Applications, Mt.Pleasant, MI, 2005, pp. 256–263.
[35] L. G. Valiant, "A theory of the learnable," *Communications of the ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.
[36] W. Gao and Z. Zhou, "Dropout rademacher complexity of deep neural networks," *SCIENCE CHINA Information Sciences*, vol. 59, no. 7, pp. 072 104:1–072 104:12, 2016.
[37] L. Rokach, "A survey of clustering algorithms," in *Data Mining and Knowledge Discovery Handbook, 2nd ed*, 2010, pp. 269–298.
[38] H. Park and C. Jun, "A simple and fast algorithm for k-medoids clustering," *Expert Systems with Applications*, vol. 36, no. 2, pp. 3336–3341, 2009.
[39] D. M. Blei and P. I. Frazier, "Distance dependent Chinese restaurant processes," in *Proceedings of the 27th International Conference on Machine Learning*, Haifa, Israel, 2010, pp. 87–94.
[40] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura, Portugal, 2012, pp. 5026–5033.
[41] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," *CoRR*, vol. abs/1710.06537, 2017. [Online]. Available: http://arxiv.org/abs/1710.06537
[42] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proceedings of the 33nd International Conference on Machine Learning*, New York, NY, 2016, pp. 1329–1338.
[43] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

Yang Yu (M'11) received the BSc and PhD degrees in computer science from Nanjing University, China, in 2004 and 2011, respectively. He joined the Department of Computer Science & Technology at Nanjing University as an assistant researcher in 2011, and is currently an associate professor. His research interests are in artificial intelligence, including reinforcement learning, machine learning, and derivative-free optimization. He has published over 20 papers in leading international journals and conferences, including *Artificial Intelligence*, IJCAI, AAAI, NIPS, etc. He has won several awards/honors including the National Outstanding Doctoral Dissertation Award, China Computer Federation Outstanding Doctoral Dissertation Award, PAKDD'08 Best Paper Award, GECCO'11 Best Paper (Theory Track), etc. He is a Junior Associate Editor of *Frontiers of Computer Science*, and an Area Chair of ACML'17, IJCAI'18, and ICPR'18.

Shi-Yong Chen received the BSc degree in computer science from Jilin University, China, in 2015. He is currently pursuing the MSc degree in the Department of Computer Science and Technology at Nanjing University, China. His research interests are reinforcement learning and machine learning.

Qing Da received the BSc and MSc degrees in computer science from Nanjing University, China, in 2010 and 2013 respectively. He is currently a senior algorithm expert in the AI team of Department of Search at Alibaba Group. His research interests are reinforcement learning and applications of machine learning.

Zhi-Hua Zhou (S'00-M'01-SM'06-F'13) received the BSc, MSc and PhD degrees in computer science from Nanjing University, China, in 1996, 1998 and 2000, respectively, all with the highest honors. He joined the Department of Computer Science & Technology at Nanjing University as an Assistant Professor in 2001, and is currently Professor and Standing Deputy Director of the National Key Laboratory for Novel Software Technology; he is also the Founding Director of the LAMDA group. His research interests are mainly in artificial intelligence, machine learning and data mining. He has authored the books *Ensemble Methods: Foundations and Algorithms* and *Machine Learning* (in Chinese), and published more than 150 papers in top-tier international journals or conference proceedings. He has received various awards/honors including the National Natural Science Award of China, the PAKDD Distinguished Contribution Award, the IEEE ICDM Outstanding Service Award, the Microsoft Professorship Award, etc. He also holds 22 patents. He is an Executive Editor-in-Chief of the *Frontiers of Computer Science*, Associate Editor-in-Chief of the *Science China Information Sciences*, Action or Associate Editor of the *Machine Learning*, *IEEE Transactions on Pattern Analysis and Machine Intelligence* , *ACM Transactions on Knowledge Discovery from Data* , etc. He served as Associate Editor-in-Chief for *Chinese Science Bulletin* (2008-2014), Associate Editor for *IEEE Transactions on Knowledge and Data Engineering* (2008-2012), *IEEE Transactions on Neural Networks and Learning Systems* (2014-2017), *ACM Transactions on Intelligent Systems and Technology* (2009-2017), Neural Networks (2014-2016), *Knowledge and Information Systems* (2003-2008), etc. He founded ACML (Asian Conference on Machine Learning), served as Advisory Committee member for IJCAI (2015-2016), Steering Committee member for ICDM, PAKDD and PRICAI, and Chair of various conferences such as General co-chair of PAKDD 2014 and ICDM 2016, Program co-chair of SDM 2013 and IJCAI 2015 Machine Learning Track, and Area chair of NIPS, ICML, AAAI, IJCAI, KDD, etc. He is/was the Chair of the IEEE CIS Data Mining Technical Committee (2015-2016), the Chair of the CCF-AI (2012-), and the Chair of the Machine Learning Technical Committee of CAAI (2006-2015). He is a foreign member of the Academy of Europe, and a Fellow of the ACM, AAAI, AAAS, IEEE, IAPR, IET/IEE, CCF, and CAAI.