Characterizing Policy Divergence for Personalized Meta-Reinforcement Learning

Michael Zhang
Harvard University
michael_zhang@college.harvard.edu

Abstract

Despite ample motivation from costly exploration and limited trajectory data, rapidly adapting to new environments with few-shot reinforcement learning (RL) can remain challenging, especially in personalized settings. We extend previous work in meta-learning with the notion that certain environments are more similar to each other than others in personalized settings, and propose a model-free meta-learning algorithm that prioritizes past experiences by relevance during gradient-based adaptation. Our algorithm characterizes past policy divergence with inverse reinforcement learning methods, and illustrates how such metrics effectively distinguish past policy parameters by the environment they were deployed in, leading to more rapid adaptation during evaluation. We also introduce a navigation testbed to specifically incorporate environment diversity across training episodes, and demonstrate that our approach outperforms meta-learning alternatives with respect to few-shot reinforcement learning in personalized settings.

1 Introduction

Swiftly learning to learn across a variety of tasks is a growing area of interest in reinforcement learning (RL), especially with regard to real-world applications. Each RL task may possess individual characteristics that parameterize an environment and define a unique Markov decision process (MDP). Similar to previous setups [1], the MDPs share the same state and action spaces, but differ in their transition dynamics. Additionally, for each MDP we assume there exists an optimal RL policy that has adapted to and thus "personalized" to each instance, capable of being learned from scratch, but which we wish to approximate in as few timesteps as possible. In the episodic few-shot learning framework, while previous meta-learning paradigms such as learning a prior initialization of parameters for fast adaptation demonstrate promise in the model-free setting [2], challenges remain when transferring similar approaches to more diverse environment dynamics. Inspired by these approaches but also motivated by these pitfalls, we note that so far no model-free approach seems to adequately address the notion that certain environments are more similar to each other than others in personalized settings, and accordingly their corresponding experiences should be given higher attention during adaptation. As an example, given a wide range of RL tasks, model-agnostic meta-learning (MAML) for policy gradients seeks to leverage the gradient updates for each task during the inner loop of training, reaching a point in parameter space relatively close to the optimal parameters of all potential tasks [3]. However, if the set of optimal policy parameters is easily organized into relatively balanced but distinct clusters, then initialization at a global optimal parameter "centroid" may be sub-optimal.

We thus propose an algorithm that explicitly seeks to determine which past parameters should be factored in during adaptation. Because the core idea of our adaptation relies on being able to quickly identify the most relevant past experiences, we rely on whole trajectories−which provide a richer source of information over the gradients, model weights, and single timesteps used in previous work [1, 3, 4]−and would like to call upon a diverse set of environment interactions for reference. Accordingly, we retain an episodic model-free approach in contrast to more recent model-based online-learning methods where each timestep is considered a new task [1]. To characterize these experiences and establish some notion of similarity between policies, we borrow from the inverse

3rd Workshop on Meta-Learning at NeurIPS 2019, Vancouver, Canada.

reinforcement learning (IRL) literature, which provides a distance metric for determining policy divergence. Our contributions center around how to (1) study personalization effectively in simulated contexts, (2) characterize the divergence of policies across a population of environments, and (3) learn effectively across different and possibly previously unseen agent types with minimum exploration cost. Towards this, we introduce a personalized 2D navigation testbed. We then propose how to characterize divergence of policies over time as they adapt to specific environments. Finally, given these insights, we introduce a K-medoids-inspired algorithm for policy adaptation and few-shot learning across multiple environments.

2 Characterizing policy divergence

For lexical consistency, we refer to the agent as the actual policy or decision-maker, and define an "entity" to be the object through which an agent interacts with the larger environment (the "patient" in our working example). Accordingly, we consider a modified version of a typical continuous state-space MDP, parameterized specifically by an individual entity type $\mathcal{T}_i$ among a population of possible types $\mathcal{T}$. We include a full description of our setup in the appendix, but each entity type then introduces an MDP $\mathcal{M}^i$ with unique transition probabilities $P^i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ and reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. Given shared state and action spaces S and A, we thus allow for diversity in behavior across entities from different types. Our goal across a population of entity types is then to learn some function $f : \mathcal{T} \rightarrow \Pi$ mapping from entity types to optimal policies, determined by maximizing the cumulative discounted reward $\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t)$ for T-length episodes.

Towards this, we wish to characterize how policies adapt to their environments over time and establish a valid distance metric between various policies for policy parameter initialization in few-shot RL. Conceptually we can imagine that given pairs of MDPs and policies $(\mathcal{M}^i, \pi^i)$ for types $\mathcal{T}_i \in \mathcal{T}$, the policies start with an indistinguishable parameter initialization and diverge when adapting to their personalized environments after t timesteps of training. If adapted policies π'_i and π'_j are still similar to each other, then this suggests that their environments $\mathcal{M}^i$ and $\mathcal{M}^j$ may be similar as well, and accordingly we can use this information to form a representation of optimal policy parameter clusters, making sure to only reference relevant prior experiences when adapting to a new environment.

From inverse reinforcement learning (IRL), where the goal is often to learn a cost function explaining an expert policy's behavior [5, 6], we can uniquely characterize a policy by its occupancy measure:

$$\rho_\pi(s, a) = \pi(a|s) \sum_{t=1}^{T} P(S_t = s|\pi), \text{ where } \rho_\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R} \tag{1}$$

Essentially this metric serves as the state-action distribution for a policy. Using a symmetric measure such as the Jensen-Shannon (JS) divergence [7], we can thus compare the occupancy measures of multiple policies at specific time steps over training as a metric into policy divergence over time.

3 Personalizing policy learning

3.1 Practical divergence estimation

Instead of looking at an occupancy measure defined over all states as above, for computational purposes we propose an alternative measure reliant only on a sampled subset of states. We observe that two policies may intuitively be different if for each of various given states, they differ in their action distributions. However, the states must have a non-zero chance of occurring in both policies of interest. In the K-shot N-ways setting, where for each of N MDPs $\mathcal{M}_i$ we get K trajectories $\{\tau_i^1, \dots, \tau_i^K\}$ to observe before updating and an additional trajectory τ'_i after the update, we then calculate a kernel density estimate (KDE) $\hat{q}(\tau'_{1:N})$ based on the observed trajectories $\tau'_{1:N} = \{\tau'_1, \tau'_2, \dots \tau'_N\}$. We then sample states $\hat{s}_j \sim \hat{q}(\tau'_{1:N})$, with probability density for state $\hat{s}_j$ given by $\hat{q}(\tau'_{1:N})(\hat{s}_j)$.

Referring back to (1), for any policy π_i, we estimate the conditional probability $\pi(\cdot|s)$ by feeding in $\hat{s}_j$ to our model network and obtaining the probability vector $\pi'_i(\hat{s}_j) = (p_1^j, \dots, p_n^j)$ where p_a^j denotes the probability of taking action a given state s_j. Towards estimating $\sum_{t=1}^{T} P(S_t = s|\pi)$, we again calculate the KDE over our observed trajectories, but now only consider τ'_i for MDP $\mathcal{M}_i$.

Accordingly, for any set of policies we first obtain a respective observed batch of trajectories, calculate an estimated state-marginalized variant of occupancy measure $\hat{\rho}_s^i$ for $s \in \mathcal{S}$ and entity types $\mathcal{T}_i$, and compute pairwise symmetric divergences across entity types. In our case we use the Jensen-Shannon divergence [7]. As a final measure, we employ K-medoids clustering to group our policies using the pairwise $D_{\text{JS}}(\cdot, \cdot)$ as a distance metric. The overall procedure is summarized in Algorithm 1.

Algorithm 1 Policy Divergence Estimation through Observed Trajectories

```
Require: Distribution over entity types $p(\mathcal{T})$, initial policies $\pi_\theta^i \in \Pi$ and episode horizon $T$
Require: Initialize update counter $t = 0$, and batch size $n$ for computing estimates
 1: while not done do
 2:     Sample entity type $\mathcal{T}_i \sim p(\mathcal{T})$ and initialize policy $\pi_\theta^i$
 3:     Sample $K$ trajectories $\tau^i = \{s_1, a_1, \ldots, s_T\}$ following $\pi_\theta^i$
 4:     Update $\theta'$ through vanilla policy gradient with REINFORCE [8]
 5:     Save trajectory $\tau_i' = \{s_1, a_1, \ldots, s_T\}$ using updated $\pi_{\theta'}^i$
 6:     Increment $t \leftarrow t + 1$
 7:     if $t = n$ then
 8:         Compute type-specific KDE $\hat{q}(\tau_i')$ for all saved $\tau_i^{'}$
 9:         Obtain concatenated updated trajectories $\tau_{1:N}' = \{\tau_1', \ldots, \tau_N'\}$
10:         Compute overall KDE $\hat{q}(\tau_{1:N}')$ and generate samples $s \sim \hat{q}(\tau_{1:N}')$
11:         for all $s$ do
12:             Calculate $\pi^i(a|s)$ for all policies $\pi^i$ and compute state probability densities $\hat{q}(\tau_i')(s)$
13:             Estimate $\hat{\rho}_s^i = \pi^i(a|s)\hat{q}(\tau_i')(s)$ for given state $s$
14:         end for
15:         Compute time-indexed pairwise Jensen-Shannon divergences $D_{\text{js}}(\hat{\rho}_s^i, \hat{\rho}_s^j)$ for all $\mathcal{T}_i, \mathcal{T}_j \in \mathcal{T}$
16:         Obtain divergence metric $\sum_{s \sim \hat{q}(\tau_{1:N}')} D_{\text{js}}(\hat{\rho}_s^i, \hat{\rho}_s^j)\hat{q}(\tau_{1:N}')(s)$
17:         Reset $t = 0$
18:     end if
19: end while
```

3.2 Cluster-Adapting Meta-Learning

Given our policy divergence estimators, we hope to better organize our previous experiences for adaptation in new environments. In this section, we describe a K-medoids-inspired meta-learning algorithm called cluster-adapting meta-learning (CAML). Similar to serial versions of meta-learning algorithms such as MAML, CAML iteratively learns an initialization for parameters of a neural network model, such that given new MDP $\mathcal{M}^i$, after a few trajectory rollouts and a single batch update during test time, our policy performs competitively to those pretrained on the same environment. However, while previous algorithms update a single set of optimal parameters drawn from parameter space, CAML maintains parameters representative of larger clusters. We show that for $K = 10$ trajectory rollouts CAML performs favorably. As described, we implement clustering using the K-medoids algorithm [9], although in practice any clustering method may be used.

At the end of our training iterations, we obtain k medoid policies. During evaluation, we then view fast-adaptation to each individual MDP as a bandit problem, where given k available arms (the policies) and K arm pulls (the few number of shots, i.e. episodes allowed), we want to maximize the corresponding reward from following the arm policy for an entire episode. While more sophisticated multi-armed bandit methods exist [10], we found that simply sampling policies and saving their cumulative associated rewards for each rollout $r \in \{1, 2, \ldots, K\}-$then initializing with the medoid policy parameters corresponding to the highest reward$-$performed competitively against baselines. The full method is described in Algorithm 2 in the appendix.

4 Experiments

4.1 Evaluating fast adaptation to personalized environments

Given training on a set support types $\mathcal{T}^S$, our learning algorithms should quickly perform well on unseen query type set $\mathcal{T}^Q$. In-line with K-shot RL, for each support type $\mathcal{T}_i^S \in \mathcal{T}^S$, our policies are

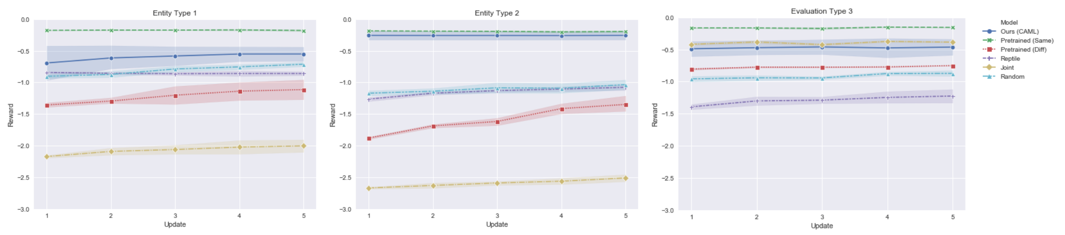

Figure 1: **Evaluation of learning algorithms on personalized environments.** CAML performs comparatively well across various evaluation entity types.

allowed K rollouts for adaptation during each training iteration. Towards evaluation, for each learning algorithm we train with 100 iterations using its policy-specific training algorithm. All underlying gradients were calculated with VPG, updating with batch size 10. For evaluation, due to the on-policy nature of policy gradient, we first allow K rollouts to collect new samples from unseen type $\mathcal{T}_i^Q \in \mathcal{T}$. Allocating these trajectories as an initialization period, for each $\mathcal{T}_i^Q$ we then evaluate fast adaptation by comparing rewards across policies for up to five updates with fine-tuning via VPG.

On a modified RL testbed inspired by classic 2D navigation [3], incorporating environment diversity (Appendix 6.2), we compare CAML to five other methods: (1) pretraining a policy by randomly sampling from all support environments during training, similar to joint training [11], (2) pretraining a policy in the same query environment that we use for evaluation, (3) pretraining a policy in a different environment, (4) training with Reptile, a meta-learning algorithm comparable to MAML in performance but much simpler in computation that performs $K > 1$ steps of stochastic gradient descent for randomly sampled support environments before directly moving initialization weights in the direction of the weights obtained during SGD [12], and (5) randomly initializing weights. For all experiments, we use vanilla policy gradient (VPG) to train a single layer 68 node neural net.

Figure 1 suggests CAML is able to more quickly adapt to unseen entity types after one update during evaluation. We note that while no policies reach the target point within the provided number of rollouts, given three different environments, CAML consistently outperforms a support type-pretrained policy and a randomly-initialized policy, and also outperforms Reptile to varying degrees as well.

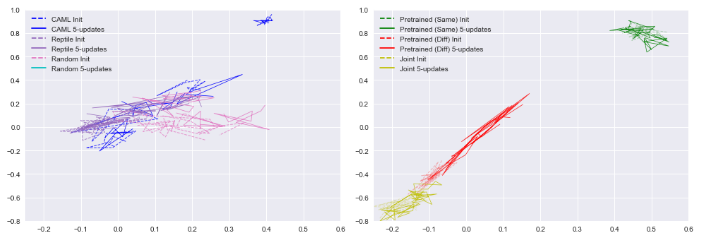

Figure 2: **Performance on 2D navigation.** Qualitative comparisons marking the ending location of particles at the end of episodes. Entities begin at the origin and try to reach point $(1, 1)$.

Lastly we acknowledge the potential challenges of pretraining to personalized environments. Observing the ending coordinates of various policies in Figure 2, we note that in certain diverse-enough situations, methods such as joint training or pretraining with another environment may initiate detrimental behavior such as moving in the wrong direction.

5 Discussion and Future Work

We tackle meta-learning for fast adaptation in personalized environments, where environments may share the same reward, but differ dramatically in state transition probabilities. Using ideas from characterizing policy divergence, we introduce competitive improvements to a promising meta-learning algorithm paradigm, better initializing to a set of optimal policy parameters by better organizing past experiences into relevant clusters. While we believe further baselines and experiments are needed, such as running comparisons with original MAML and other flat and non-flat hierarchical approaches, as well as characterizing the added complexity maintaining multiple initializations, our results suggest that meta-learning between all potential optimal parameters may be sub-optimal when trying to adapt to diverse-enough environments. Our method builds on this original motivating curiosity, and suggests one possible alternative to better approach personalized reinforcement learning.

References

[1] Ignasi Clavera, Anusha Nagabandi, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt: Meta-learning for model-based control. *CoRR*, abs/1803.11347, 2018.

[2] Abhishek Gupta, Benjamin Eysenbach, Chelsea Finn, and Sergey Levine. Unsupervised meta-learning for reinforcement learning. *CoRR*, abs/1806.04640, 2018.

[3] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017.

[4] Anusha Nagabandi, Chelsea Finn, and Sergey Levine. Deep online learning via meta-learning: Continual adaptation for model-based RL. *CoRR*, abs/1812.07671, 2018.

[5] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[6] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *CoRR*, abs/1606.03476, 2016.

[7] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, January 1991.

[8] Richard S Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.

[9] Xin Jin and Jiawei Han. *K-Medoids Clustering*, pages 564–565. Springer US, Boston, MA, 2010.

[10] Andreas Merentitis, Kashif Rasul, Roland Vollgraf, Abdul-Saboor Sheikh, and Urs Bergmann. A bandit framework for optimal selection of reinforcement learning agents. *CoRR*, abs/1902.03657, 2019.

[11] Wenhao Yu, Greg Turk, and C. Karen Liu. Multi-task learning with gradient guided policy specialization. *CoRR*, abs/1709.07979, 2017.

[12] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *CoRR*, abs/1803.02999, 2018.

6 Appendix

6.1 Personalized Markov decision processes (MDPs)

We consider a modified version of a typical continuous state-space MDP, parameterized specifically by entity type $\mathcal{T}_i \in \mathcal{T}$, which we denote $\mathcal{M}^i \triangleq (\mathcal{S}, \mathcal{A}, P^i, r^i, \gamma, s_0^i)$. As described earlier, we let $\mathcal{S}$ be the set of observable continuous states and deal with discrete action space $\mathcal{A}$. Our characterization of a typical MDP diverges with the introduction of lowercase i which denotes an individual entity type among set of possible types $\mathcal{T}$.

For any two different agent types $\mathcal{T}_i \neq \mathcal{T}_j$, P^i may not necessarily differ from P^j, but crucially at the onset of training for some subset of unseen types $\mathcal{T}_i \subseteq \mathcal{T}$ we make no assumption that these transition dynamics are the same. Finally, γ is a discount factor assumed to be the same for all agents, and s_0^i is the initial state distribution for agent type i. Our goal across a population of agent types is then to learn some function $f : \mathcal{T} \to \Pi$ mapping from agent types to policies. For each agent type given s_0^i and a minimal number of timesteps t, we can then output an optimal policy π_θ^i with regard to maximizing the cumulative discounted reward $\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t)$ for T-length episodes, where θ denotes the parameters for personalized policy π_i taking action $a_t \in \mathcal{A}$ given state $s_t \in \mathcal{S}$.

6.2 A personalized particle environment for 2D navigation

We begin our study of divergence in personalizing policies with a 2D continuous episodic gridworld environment, where a point agent must move to a target position in a constrained amount of time. The state space is given by the agent's current 2D position, and at each timestep the agent may choose to go left, right, up, or down one pixel unit. Following these commands, the agent tries to maximize its reward, calculated to be the negative squared distance to the goal after each timestep. Finally episodes end every 100 timesteps, or when the agent is within 0.01 units of the goal. Particles start at origin $(0, 0)$, and try to reach target at $(1, 1)$ before the episode ends. $\mathcal{S}$ is defined by the coordinates of the particle at timestep $t \in \{0, 1, \ldots T\}$. Actions are discrete a, corresponding to moving one unit left, right, down, up, i.e. $a_t \in \{(-1, 0), (1, 0), (0, -1), (0, 1)\}$.

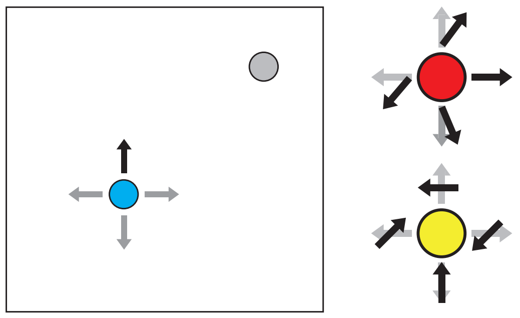
Figure 3: **2D personalized particles.** Policy π tries to move entity (blue, red, yellow) to target (gray), but entities behave in different ways unknown to the policy. Right: Two remapping schemes, where some cardinal direction e.g. 'up' corresponds to a different transition vector.

We personalize this environment with the introduction of a population of entities, where we may either stick with one throughout the policy's entire training run or randomly introduce new entities into the world at the start of every episode. Each agent is initialized with a personalization function $\mathcal{F} : \mathcal{A} \to \mathcal{A}'$, which first remaps the cardinal directions of each action and then imposes additional variance. For example, given default action $a_r = (1, 0)$, telling a default agent to go right, our policy may encounter an agent who upon receiving action input $(1, 0)$ actually moves with action $(0.42, -1)$. This is achieved by first remapping a_r to $(0, -1)$, and then imposing further variance on the x-coordinate. We can think of this as having the same set of interventions for all agents, but facing different responses in the form of varied actual transition outcomes.

6.3 Matching policy divergence to training environments

Towards evaluating both the dynamics of our testbed and the strength of our policy divergence distance metric, we first sought to see if clustering on trained VPG trajectories could recover their corresponding initial environmental dynamics. To do so, we first initiated six latent entity types by remapping controls completely (e.g. the yellow particle in Figure 1). Afterwards, for each latent type, we generated four variants by further introducing uniform variance for given entity and action while preserving the defining cardinal direction (red particle in Figure 3). In total we ended up with 24 entity types. To eludidate divergence across optimal policies, for each type we then trained an individual VPG policy from scratch over 40 updates performing a batch parameter update every 10 iterations, and calculating estimated occupancy-measure-based distances according to Algorithm 1.

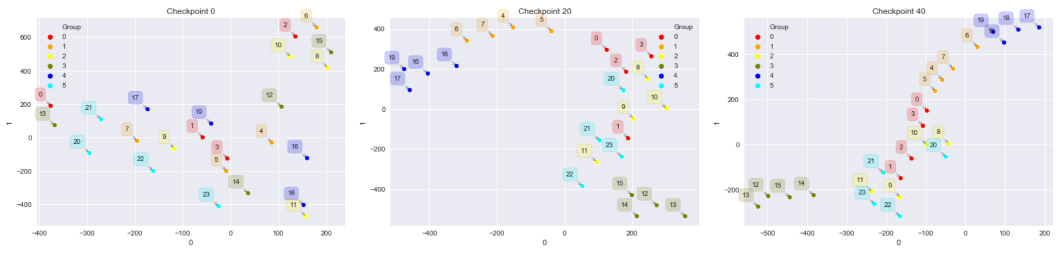

Figure 4: **Policy divergence over time.** T-SNE visualization of policy divergences over 40 training updates. Colors denote original latent group. Computing pairwise distances using our divergence metric leads to noticeable grouping over time.

As observed in Figure 4, based on estimated occupancy measure, trained policies show increasing signs of neighboring with members of their respective groups over training updates. One takeaway is that our occupancy measure-based divergence is effective at measuring policy divergence, where directed through their respective environments, we can thus evaluate how different two policies are based on their trajectories.

When comparing policy divergence with training progress, we note that convergence in performance over time suggests that expert policies trained on similar environments are close to each other in occupancy measure space. Accordingly, in the few-shot RL setting, one valid strategy for producing an optimal policy without having to train from scratch would be to identify prior expert policies trained on similar environments. Taking this one step further, because our occupancy measures directly derive from $\pi(a|s)$ which is itself derived from a policy network's parameters, this provides another interpretation into the effectiveness of finding nearby parameters in policy space.

6.4 Full algorithm for Cluster-Adapting Meta-Learning

Algorithm 2 Training with CAML (K-medoids version)

```
Require: Distribution over entity types $p(\mathcal{T})$, initial policy parameters $\theta$, episode length $T$
Require: Number of medoids $k$, batch size to cluster with $n$
1: for initial iterations $1, \ldots, n$ do
2:     Sample entity type $\mathcal{T}_i \sim p(\mathcal{T})$ and initialize new policy $\pi_\theta^i$
3:     Obtain and save updated $\pi_{\theta'}^i$ and $\tau_i'$ through Steps $3, 4, 5$ in Algorithm 1
4: end for
5: Compute distance matrix $D$ with pairwise distances on the saved $\tau_i'$ using Algorithm 1.
6: Perform $K$-medoids clustering on $D$, saving set of $k$ corresponding medoid policies $\Pi_\theta^k$ and
   trajectories $\tau_{1:N}^k$. Save trajectories $\tau^k \in \tau_{1:N}^k$ to compute next iteration of occupancy measures.
7: for iterations $n+1, n+2 \ldots$ do
8:     Sample entity type $\mathcal{T}_i \sim p(\mathcal{T})$ and randomly sample medoid policy $\pi_\theta \in \Pi^k$
9:     Obtain and save updated $\pi_{\theta'}^i$ and $\tau_i'$ through Steps $3, 4, 5$ in Algorithm 1
10:    if number of save policies equals $n$ then
11:        Repeat steps 5 and 6, updating $k$ medoid policies and trajectories.
12:    end if
13: end for
```