Discovering Diverse Solutions in Deep Reinforcement Learning

Takayuki Osa\textsuperscript{1,2}, Voot Tangkaratt\textsuperscript{2} and Masashi Sugiyama\textsuperscript{2,3}

\textsuperscript{1}Kyushu Institute of Technology, Fukuoka, Japan
\textsuperscript{2}RIKEN Center for Advanced Intelligence Project, Tokyo, Japan
\textsuperscript{3}University of Tokyo, Tokyo, Japan

Abstract

Reinforcement learning (RL) algorithms are typically limited to learning a single solution of a specified task, even though there often exists diverse solutions to a given task. Compared with learning a single solution, learning a set of diverse solutions is beneficial because diverse solutions enable robust few-shot adaptation and allow the user to select a preferred solution. Although previous studies have showed that diverse behaviors can be modeled with a policy conditioned on latent variables, an approach for modeling an infinite set of diverse solutions with continuous latent variables has not been investigated. In this study, we propose an RL method that can learn infinitely many solutions by training a policy conditioned on a continuous or discrete low-dimensional latent variable. Through continuous control tasks, we demonstrate that our method can learn diverse solutions in a data-efficient manner and that the solutions can be used for few-shot adaptation to solve unseen tasks.

1 Introduction

Reinforcement learning (RL) has achieved significant success in various fields, including robotic manipulation [Levine et al. (2016); Bodnar et al. (2020)], board games [Silver et al. (2016)], and optimization of experimental setups [Sorokin et al. (2020)]. In RL, it is known that there can be multiple optimal policies that elicit the optimal value function [Sutton & Barto (2018)]. However, existing RL methods are typically limited to obtaining only one of the optimal policies in a stochastic manner. To overcome this shortcoming, RL methods for learning diverse solutions have been recently proposed [Kumar et al. (2020)]. However, these methods have only been evaluated for learning a finite number of solutions, and their effectiveness when learning an infinite number of solutions is unknown.

In this study, we propose a novel RL method that can learn an infinite number of diverse solutions in a sample-efficient manner. In our framework, the diversity of solutions is encoded in a low-dimensional latent variable. Hence, the user can intuitively analyze and select the desired behavior among the learned diverse behaviors.

Our contribution is the proposal of a practical algorithm for training a policy conditioned on continuous or discrete latent variables in RL. In our method, a policy conditioned on a latent variable is trained by directly maximizing a variational lower bound of the mutual information via back-propagation, instead of using the mutual information as unsupervised rewards as in previous studies [Eysenbach et al. (2019); Kumar et al. (2020); Sharma et al. (2020)]. We empirically demonstrate that our methods can learn diverse solutions for specified tasks in a more sample-efficient manner compared with existing RL methods for learning diverse solutions [Kumar et al. (2020)]. In addition, the few-shot robustness [Kumar et al. (2020)] of our method is empirically demonstrated, and our method outperforms baseline methods in few-shot adaptation tasks.
2 Related Work

Learning from expert demonstrations is efficient for obtaining complex and natural motions [Peng et al., 2018; Osa et al., 2018]. In terms of imitation learning, previous studies [Wang et al., 2017; Li et al., 2017; Sharma et al., 2019; Merel et al., 2019] addressed the problem of imitating diverse behaviors from expert demonstrations. In these methods, diverse behaviors are encoded in a latent variable. However, these methods assume the availability of observations of diverse behaviors performed by experts. In this study, instead of learning from demonstrations, we address the problem of encoding diverse behaviors by learning from a reward function through trial and error.

In terms of multimodal optimization, various black-box optimization methods have been developed to obtain multiple solutions in optimization problems [Goldberg & Richardson, 1987; D. P. Kroese & Rubinstein, 2006; Deb & Saha, 2010; Stoean et al., 2010]. Agrawal et al. (2014) proposed methods that learn diverse skills using a genetic algorithm for multimodal optimization. However, methods based on black-box optimization such as the cross-entropy method [de Boer et al., 2005] and CMA-ES [Hansen & Ostermeier, 1996] are limited in scalability.

Recently, Eysenbach et al. (2019) and Sharma et al. (2020) investigated methods for learning diverse behaviors using deep RL with unsupervised rewards. However, these methods are not aimed for learning a policy to solve a specific task, and they learn diverse behaviors randomly. [Kumar et al., 2020] proposed a method, SMERL, that learns multiple solutions for a specified task. However, a disadvantage of SMERL is that it requires information regarding the optimal return value before learning diverse solutions. Therefore, SMERL requires a separate RL algorithm to be executed in advance to estimate the optimal return value because the optimal return value is typically unknown. By contrast, our method does not require information regarding the optimal return value, and a separate RL algorithm need not be executed.

Hierarchical reinforcement learning (HRL) methods [Bacon et al., 2017; Florensa et al., 2017; Nachum et al., 2018, 2019; Osa & Sugiyama, 2018; Osa et al., 2019] learn a hierarchical policy structured by a latent variable. However, the latent variable in HRL algorithms do not provide multiple solutions for a specified task. Recently, Nachum et al. (2018, 2019) investigated goal-conditioned policies \( \pi(a|s, g) \), where \( g \) denotes the goal. The goal-conditioned policy can be regarded as a method of modeling diverse behaviors conditioned on a latent variable that has semantic meaning. However, learning a goal-conditioned policy requires the user to specify the goals in a supervised manner. By contrast, we aim to learn diverse behaviors in an unsupervised manner by using only reward signals.

In the field of deep learning, methods for learning latent representations from data have been developed, such as generative adversarial networks (GANs) [Goodfellow et al., 2014; Chen et al., 2016; Arjovsky et al., 2017] and variational autoencoders [Kingma & Welling, 2014; Higgins et al., 2016; Dupont, 2018]. In infoGAN Chen et al. (2016), a generator is trained based on a discriminator while the mutual information between the latent variable and the synthetic sample generated by the generator is maximized. The actor in our framework is analogous to the generator in infoGAN in that the actor is trained based on a critic while the mutual information between the latent variable and state-action pair is maximized.

3 Background

We consider a reinforcement learning problem under a Markov decision process (MDP) defined by a tuple \((S, A, P, r, \gamma, d)\) where \(S\) is the state space, \(A\) the action space, \(P(s_{t+1}|s_t, a_t)\) the transition probability density, \(r(s, a)\) the reward function, \(\gamma\) the discount factor, and \(d(s_0)\) the probability density of the initial state. A policy \(\pi(a|s): S \times A \rightarrow \mathbb{R}\) is defined as the conditional probability density over actions given states. The goal of RL is to identify a policy that maximizes the expected cumulative discounted reward \(\mathbb{E}[R_0|\pi]\) where \(R_t = \sum_{k=t}^{\infty} \gamma^{k-t} r(s_k, a_k)\).

In this study, we focus on tasks where multiple, possibly infinitely many policies achieve the maximal expected return. This scenario can be observed in many practical tasks, particularly in robotics [Toussaint et al., 2018; Orthey et al., 2020]. For example, when learning the walking behavior of a robot, many walking motions arrive at the destination at the same speed, as shown in the top row of Figure 1.

3.1 Few-Shot Robustness

A recent study discussed the benefit of learning diverse solutions in terms of few-shot robustness [Kumar et al., 2020]. The protocol to evaluate few-shot robustness is described in [Kumar et al., 2020]. In the training phase, a policy is trained in a training MDP given by \(M = (S, A, P, r, \gamma, d)\). Subsequently, the policy is tested in a test MDP with \(M' = (S, A, P', r', \gamma, d')\), in which the state and action space are identical but the transition probability density, initial state density and reward functions differ from those of the training MDP. In the test MDP, the policy should be adapted after a limited number of trials and then return the best policy. The number of episodes in which the agent is allowed to interact with the test MDP is referred to as the budget. The performance of the policy after adaptation is evaluated under the test MDP. In this study, we evaluated the few-shot robustness of policies trained using our method.
3.2 Latent-Conditioned Policies

To model diverse behaviors, we consider a policy conditioned on a latent variable as in the previous studies [Eysenbach et al., 2019; Kumar et al., 2020; Sharma et al., 2020]. Herein, we introduce latent variable \( z \in \mathcal{Z} \), where \( \mathcal{Z} \) is a set of possible values of \( z \) that can be discrete or continuous. The conditional probability density of observing a trajectory \( \tau \) given the latent variable \( z \) is expressed as

\[
p(\tau | z) = d(s_0) \prod_{t=0}^{T} \pi(a_t | s_t, z) P(s_{t+1} | s_t, a_t).
\] (1)

Likewise, the Q-function conditioned on the latent variable is expressed as

\[
Q^\pi(s, a, z) = \mathbb{E}_z[R | s, a, z],
\] (2)

which represents the expected return when following a policy for the latent variable. In this study, we aim to learn a policy that generates various trajectories by changing the value of latent variable \( z \).

4 Learning Latent Representations of Policies

In this section, we formulate the problem of learning diverse solutions in RL. Subsequently, we describe the method to perform information maximization based on a variational lower bound.

4.1 Problem Formulation

We consider a policy of the form

\[
\pi(a | s) = \int \pi(a | s, z)p(z)dz,
\] (3)

where \( z \) is the latent variable and \( p(z) \) is the prior distribution of \( z \). The latent variable \( z \) can be either continuous or discrete. We can train the latent-conditioned policy by maximizing the expected return \( \mathbb{E}[R | \pi] \). However, such a policy may disregard the latent variable and would not be useful for learning diverse solutions. Hence, we propose maximizing the mutual information between the latent variable and the state-action variable. Specifically, to encode diverse behaviors in the latent-conditioned policy, we aim to solve the problem formulated as

\[
\max_\pi \left( \mathbb{E}[R | \pi] + I_\pi(s, a; z) \right),
\] (4)

where \( \mathbb{E}[R | \pi] \) is the expected return of \( \pi \) and \( I_\pi(s, a; z) \) is the mutual information between a state action pair \((s, a)\) and latent variable \( z \) under policy \( \pi \). In our framework, we maximize the mutual information \( I_\pi(s, a; z) \) defined as

\[
I_\pi(s, a; z) = \iint p_\pi(s, a, z) \log \left( \frac{p_\pi(s, a, z)}{p_\pi(s, a)p(z)} \right) dzda ds,
\] (5)

where \( p_\pi(s, a, z) = \beta(s, z)\pi(a | s, z) \), \( p_\pi(s, a) = \int p_\pi(s, a, z)dz \) and \( \beta(s, z) \) is the joint density of states and latent variables induced by the behavior policy.

In previous studies, the mutual information between the state and latent variable \( I(s; z) \) is typically maximized [Eysenbach et al., 2019; Kumar et al., 2020], thereby encouraging different skills to visit different states. By contrast, we attempt to maximize \( I(s, a; z) \) to encode the diversity of actions for a specified state into the latent variable. The maximization of \( I(s, a; z) \) encourages different skills to visit different states and perform different actions.

4.2 Information Maximization via Variational Lower-Bound

In practice, directly computing \( I_\pi(s, a; z) \) is challenging because it requires the density estimation of \( p_\pi(s, a, z) \) and \( p_\pi(s, a) \). Hence, we use a variational lower bound of mutual information as the surrogate objective. The variational lower bound of the mutual information between the state-action pair \((s, a)\) and the latent variable \( z \) can be derived as follows:

\[
I_\pi(s, a; z) = H(z) - H(z | s, a)
\]

\[
= \mathbb{E}_{(s, a, z) \sim p_\pi} \left[ \log p(z | s, a) \right] + H(z)
\]

\[
= \mathbb{E}_{(s, a) \sim \beta(s, a)} \left[ D_{KL}(p(z | s, a) || q_\phi(z | s, a)) \right] + \mathbb{E}_{(s, a, z) \sim p_\pi} \left[ \log q_\phi(z | s, a) \right] + H(z)
\]

\[
\geq \mathbb{E}_{(s, a, z) \sim p_\pi} \left[ \log q_\phi(z | s, a) \right] + H(z)
\] (6)

Herein, the density model \( q_\phi(z | s, a) \) parameterized by a vector \( \phi \) is introduced to approximate the posterior density \( p(z | s, a) \), and \( p_\pi(s, a, z) \) is the joint density of state \( s \), action \( a \), and latent variable \( z \) stored in replay buffer \( \mathcal{B} \). We regarded entropy \( H(z) \) as a constant with respect to the policy parameter, and we set \( p(z) \) to be uniform to achieve a high \( H(z) \).

It was demonstrated that this lower-bound is tight [Jord dan et al., 1999; Barber & Agakov, 2003]. Specifically, the surrogate on the right-hand side of (6) is equivalent to the mutual information \( I_\pi(s, a; z) \) for the optimal variational distribution \( q_\phi = \max_{q_\phi} \mathbb{E}[\log q_\phi(z | s, a)] + H(z) \). Therefore, we can replace the mutual information in (4) with \( \max_{q_\phi} \mathbb{E}[\log q_\phi(z | s, a)] + H(z) \), resulting in an optimization problem

\[
\max_{\pi, \phi} \left( \mathbb{E}[R | \pi] + \mathbb{E}[\log q_\phi(z | s, a)] \right),
\] (7)
In this section, we present two practical algorithms for training latent-conditioned policy. In the next section, we propose a deterministc policy as in DDPG [Lillicrap et al. 2016] and TD3 [Fujimoto et al. 2018]. Therefore, \( \pi(a|s,z) \) is a Dirac-delta function that satisfies
\[
\int Q(s,a,z)\pi(a|s,z)da = Q(s,\mu_{\theta}(s,z),z),
\]
where \( \mu_{\theta}(s,z) : S \times Z \mapsto A \) is a model parameterized by a vector \( \theta \) that determines action \( a \) for given state \( s \) in a deterministic manner.

We train the policy \( \pi(a|s,z) \) by maximizing
\[
J(\theta, \phi) = J_Q(\theta) + J_{info}(\theta, \phi),
\]
where \( J_Q(\theta) \) is the term to maximize the expected return based on the estimated Q-function, and it is given by
\[
J_Q(\theta) = E(s,z) \sim B [Q_{\theta}(s,\mu_{\theta}(s,z),z)],
\]
and the term \( J_{info}(\theta, \phi) \) is
\[
J_{info}(\theta, \phi) = E(s,z) \sim B [W \log q_{\phi}(z|s,\mu_{\theta}(s,z))].
\]
We refer to our algorithm based on TD3 as the latent-conditioned TD3 (LTD3).

5.2 SAC-based Algorithm
In the SAC-based implementation, a stochastic policy \( \pi_{\theta}(a|s,z) \) is modeled with a neural network using the reparameterization trick as follows:
\[
\tilde{a} = \mu_{\theta}(s,z) + \epsilon \sigma_{\theta}(s,z),
\]
where \( \mu_{\theta} \) and \( \sigma_{\theta} \) are the mean and standard deviation of the Gaussian policy, respectively. \( \epsilon \) is a diagonal matrix whose diagonal elements are a noise vector sampled from a Gaussian distribution. We use the soft Q-function instead of the standard Q-function, and the target value for the soft-Q function is computed as
\[
y = r + \gamma \left( \min_{i=1,2} Q_{w_i}(s',\tilde{a}',z) \right) - \alpha \log \pi(\tilde{a}'|s',z),
\]
where \( \alpha \) is a hyperparameter.
Algorithm 1 Latent-Conditioned Off-Policy Actor-Critic

Input: Dimension of latent variable z
for each episode do
  Sample latent variable z ∼ p(z)
  for t = 0 to T do
    Select action with exploration noise
    Observe reward r and new state s′
    Store tuple (s, a, s′, r, z) in B
    Sample mini-batch from B
    Update the critics by minimizing $L_{\text{critic}}(w_i)$ in (12)
    if $t \mod d_{\text{atrr}}$ then
      Update the actor by maximizing $J_q(\theta)$ in (17) or (21)
    end if
    if $t \mod d_{\text{info}}$ then
      Update $\phi$ and $\theta$ by maximizing $J_{\text{info}}(\theta)$ in (18)
    end if
  end for
end for

Sample mini-batch from $B$
for $z \sim p(z)$ do
  Observe reward $r_t$
  if $t \mod d_{\text{atrr}}$ then
    Update $\theta_{\text{critic}}$ by minimizing $\mathcal{L}_{\text{critic}}(w_i)$ in (12)
  end if
  if $t \mod d_{\text{info}}$ then
    Update $\phi$ and $\theta$ by maximizing $J_{\text{info}}(\theta)$ in (18)
  end if
end for

Figure 2: Network architecture used in our implementation of LTD3.

where $\alpha$ is the temperature parameter. The term for maximizing the expected return is as follows:

$$J_q(\theta) = \mathbb{E}(s, z) \sim B, \epsilon \sim \mathcal{N}(0,1) \left[ Q_{\text{soft}}(s, a, z) \right]$$

where $Q_{\text{soft}}^\alpha(s, a, z)$ is the soft Q-function conditioned on the latent variable. In the SAC-based implementation, we can use the reparameterization trick to maximize the information term. However, we observed in our preliminary experiments that using the reparameterization trick for maximizing the information term results in a mediocre learning performance. Hence, we maximize the information term in (18) and update only the mean of the policy when maximizing the information term. We refer to our algorithm based on SAC as the latent-conditioned SAC (LSAC).

The proposed algorithm is summarized in Algorithm 1. The network architecture used in our implementation of LTD3 is shown in Figure 2. In the network that models the latent-conditioned policy, the latent variable is input to a fully connected layer and then concatenated with the state vector. In the implementation, the conditional density $q_\phi(z|s, a)$ is modeled by a neural network. We assume that the latent variable $z$ is given by $z = [z_{\text{cont}}, z_{\text{disc}}]$ where $z_{\text{cont}}$ and $z_{\text{disc}}$ are continuous and discrete, respectively. As in infoGAN, we regard $p(z_{\text{cont}}|s, a)$ as a factored Gaussian. For the categorical latent variable $z_{\text{disc}}$, the categorical distribution $q_\phi(z_{\text{disc}}|s, a)$ is modeled with a softmax layer. Whereas existing methods [Kumar et al. (2020), Sharma et al. (2020)] regards the mutual information term as unsupervised rewards, we directly maximize the information term via back-propagation in our method. The gradient of the information term is back-propagated from the posterior approximator to the policy network.

6 Experiments

We used tasks based on OpenAI Gym [Brockman et al. (2016)] with Mujoco physics engine [Todorov et al. (2012)]. The tasks were based on the Hopper, Walker2d and Humanoid tasks, as shown in Figure 3.

6.1 Learning Diverse Solutions

In the reward function of the original tasks in OpenAI Gym, a velocity term $r_{\text{vel}}$ is used to encourage the agent to walk faster, and $r_{\text{vel}}$ is given by

$$r_{\text{vel}} = \frac{(x_t - x_{t-1})}{\Delta t},$$

where $x_t$ represents the horizontal position of the agent at time $t$, and $\Delta t$ is the time step size in the simulation. As in the study in [Kumar et al. (2020)], we modified the velocity term to the following:

$$r_{\text{vel}} = \min \left( \frac{(x_t - x_{t-1})}{\Delta t}, v_{\text{max}} \right),$$

where $v_{\text{max}}$ is a constant term that defines the upper bound of $r_{\text{vel}}$. Therefore, the agent is required to walk at the maximum speed $v_{\text{max}}$ with minimal control cost instead of walking as fast as possible. We refer to the modified version of the Hopper, Walker2d and Humanoid tasks as HopperVel, Walker2dVel, and HumanoidVel tasks, respectively. Details of these tasks are described in Appendix A.1.

1 Background color is changed from default setting for visibility.
Learning Curve We compared the proposed methods with SMERL [Kumar et al. (2020)] and a variant of DIAYN [Eysenbach et al. (2019)], which we refer to as SAC-DIAYN and was used as a baseline in the study by [Kumar et al. (2020)]. In SAC-DIAYN, the policy is trained to maximize the sum of the task reward and an unsupervised reward given by

\[ r_{\text{diayn}} = \log q_{\psi}(z|s) \]  

(24)

where \( q_{\psi}(z|s) \) is a parameterized model that discriminates the value of the latent variable. Although previous studies reported results with discrete latent variables, this unsupervised reward can be applied to the continuous latent variable if we assume a factored Gaussian distribution. In SMERL, \( r_{\text{diayn}} \) is given only when the agent achieves a return close to the optimal return. Therefore, an off-the-shelf RL method must be used to determine the optimal return before running SMERL. Our implementation of TD3 and SAC was adapted from the implementation in OpenAI Spinning Up [Achiam 2018].

We show a comparison of the learning curve of the baseline methods in Figure 4. TD3 and SAC required approximately 1 million time steps to learn the optimal behavior on the tasks. As shown in Figure 4, the sample-efficiency of LSAC is comparable to those of TD3 and SAC, whereas LTD3 required approximately 2 million time steps until the convergence. However, SMERL did not reach the maximal return in 2 million time steps. Therefore, an off-the-shelf RL method must be used to determine the optimal return. Hence, we conclude that LTD3 and LSAC are more sample-efficient than SMERL. The learning curve of LTD3 with different values of the clipping parameter is shown in Figure 5. The use of the importance weight improved the sample-efficiency of LTD3, particularly on the HopperVel task.

Qualitative Analysis We show the behaviors learned by LTD3 on the HopperVel task in Figure 6. In these figure, the screens were capture at the same time steps, and the variance of the position indicated the variance of the walking speed. As shown, LTD3 discovered diverse behaviors for solving given tasks. It is challenging to manually design the reward function that specifically elicit one of these behaviors.

Figure 6(a) shows the results with a two-dimensional continuous latent variable on the HopperVel task. The agent on the HopperVel task has a leg with two knee joints, and the latent variable encoded various styles of using these joints for hopping. The lower knee joint was bent when \( z_0 = -0.9 \), whereas the lower knee joint was straight when \( z_0 = 0.9 \). When \( z_1 = 0.9 \), the upper knee joint was bent when \( z_0 = -0.9 \). Figure 6(b) shows the results with a one-dimensional continuous latent variable and the three categorical latent variables on the HopperVel task. When the discrete latent variable changed, the walking behavior changes discontinuously. Regardless of the value of the discrete latent variable, the angle of the upper joint changed continuously when the value of the continuous latent variable changed continuously. This result shows that LTD3 successfully learned diverse solutions that
achieved the task, and that the user might select a preferred solution by selecting the value of the latent variable. The learned solutions for the Walker2Dvel and Humanoid Vel tasks are provided in Appendix B, and they supports the discussion in this section.

6.2 Few-Shot Adaptation

We evaluated the few-shot robustness of the policy trained using LTD3 ans LSAC, based on the protocol proposed by Kumar et al. [2020]. A policy was trained on a training environment for 3 million times steps, and the trained policy was adapted to the test environment in \( k \) episodes. We used the HopperVel and WalkerVel tasks presented in the previous section as the training MDPs. In the test MDPs, parts of the agent body, such as position of the knee and the length of a link, are modified as shown in Figure 7. For example, in the W-ShortRed task, the red leg of the agent was shorter than the original Walker2d agent’s leg. Likewise, the orange leg of the agent was shorter than the original Walker2d agent’s leg in the W-ShortOrange task. To adapt a policy to both tasks, a wide range of walking skills must be learned in the training MDP. Details of the tasks in the test MDPs are described in Appendix A.2.

As a baseline, we evaluated SMERL and SAC-DIAYN as in the previous experiment. We performed few-shot adaptation with \( k = 5 \) and \( k = 25 \). In this experiment, we evaluated LTD3, LSAC and SMERL with two-dimensional continuous, one-dimensional continuous, and discrete latent variables. For SAC-DIAYN, we present the results with the discrete latent variable. For the discrete latent variables, the dimension of the latent variable was set to \( k \) as \( |Z| = k \). Therefore, \( k \) skills were learned on the training environment, and each learned skill was tested once in the test environment. For methods with continuous latent variables, the latent variables were uniformly sampled from \([-1, 1]\], and the performance of the policy with different values of the latent variable was tested in \( k \) episodes. Subsequently, the value of the latent variable with the best performance was used to evaluate the performance after the few-shot adaptation.

The results of the few-shot adaptation tasks with the Walker2d agent are shown in Table 1. LTD3 and LSAC outperformed SMERL and SAC-DIAYN in this experiment. LTD3 with the discrete latent variable outperformed LTD3 with the continuous latent variables on a half of the tasks, although the policy conditioned on the continuous latent variable can model more diverse solutions than the policy conditioned on the discrete latent variable. Regarding the comparison of LTD3 and LSAC, if the latent variable is discrete, then LTD3 outperforms LSAC. Meanwhile, the performances of LTD3 and LSAC are comparable when the latent variable is continuous.

LTD3 with the discrete variable showed performances that were superior or comparable to those of SAC-DIAYN and SMERL with the discrete latent variable on all the few-shot adaptation tasks. Although the performance of SMERL with the discrete latent variable improved when the budget was increased from \( k = 5 \) to \( k = 25 \), the performance of LTD3 with \( k = 5 \) is better or comparable to that of SMERL with \( k = 25 \). In addition,
SMERL required running an off-the-shelf RL method to know the value of the return of the optimal policy, which is not necessary for LTD3 and LSAC. In this sense, LTD3 and LSAC are more sample-efficient in the training phase. These results show that the proposed algorithms LTD3 and LSAC outperformed baseline methods in few-shot adaptation tasks and indicate that the proposed algorithms can learn more diverse skills in training MDPs than the baseline methods. The results of the few-shot adaptation tasks with the Hopper agent are shown in Appendix B, and they also support the discussion in this section.

7 Discussions

Our method can be interpreted as an approach for learning a generative model of actions that maximizes the expected return. A limitation of our method is that the trained model does not necessarily encompass all possible behaviors. Consequently, the learned behaviors are often different when using different random seeds in the training of the neural networks, although this is a typical limitation of deep models. Therefore, the desired types of behaviors are not necessarily included in the trained model.

In addition to the few-shot robustness, we believe that learning diverse solutions can be beneficial. For example, learning the latent space of motor skills is useful when editing the motions of characters for application to computer graphics. Our method enables us to obtain diverse behaviors and allow users to select preferable behaviors among the obtained behaviors. Recently, Shen et al. (2020) proposed a method to allow the user to edit face images generated by a GAN with predefined attributes, such as age and facial expressions. The extension of the latent-conditioned policy to allow the user to edit the motor skills using specific attributes will be an interesting research direction.

Recent studies indicate that there are often multiple solutions in robot motion planning Toussaint et al. (2018); Orthey et al. (2020); Osa (2020), but techniques for modeling diverse plans are not established yet. We will investigate how to leverage our method in these applications.

8 Conclusions

We proposed RL algorithms, LTD3 and LSAC, which can train the latent-conditioned policy to represent diverse solutions. In our approach, diverse behaviors are encoded in continuous or discrete latent variables by variational information maximization. The experimental results indicated that our method can learn diverse solutions in continuous control tasks, and that our method
was sample-efficient compared with existing methods for learning diverse solutions. In addition, our method outperformed the existing methods on few-shot adaptation tasks. We will further investigate methods to leverage diverse solutions in various applications.

Acknowledgment

TO was supported by KAKENHI 19K20370, and MS was supported by KAKENHI 17H00757.

References

Achiam, J. Spinning Up in Deep Reinforcement Learning. 2018.

Agrawal, S., Shen, S., and v. d. Panne, M. Diverse motions and character shapes for simulated skills. IEEE Transactions on Visualization and Computer Graphics, 20(10):1345–1355, 2014.

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein generative adversarial networks. In Proceedings of the 34th International Conference on Machine Learning, volume 70 of Proceedings of Machine Learning Research, pp. 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

Bacon, P. L., Harb, J., and Precup, D. The option-critic architecture. In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), 2017.

Barber, D. and Agakov, F. V. The IM algorithm: A variational approach to information maximization. In Advances in Neural Information Processing Systems, 2003.

Bodnar, C., Li, A., Hausman, K., Pastor, P., and Kalakrishnan, M. Quantile qt-opt for risk-aware vision-based robotic grasping. In Robotics and Science and Systems, 2020.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. arXiv preprint arXiv:1606.01540, 2016.

Chen, X., Duan, Y., Huthoof, R., Schulman, J., Sutskever, I., and Abbeel, P. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In Advances in Neural Information Processing Systems, 2016.

D. P. Kroese, S. P. and Rubinstein, R. Y. The cross-entropy method for continuous multi-extremal optimization. Methodology and Computing in Applied Probability, 8:383–407, 2006.

de Boer, P. T., Kroese, D. P., Mannor, S., and Rubinstein, R. Y. A tutorial on the cross-entropy method. Annals of Operations Research, 134:19–67, 2005.

Deb, K. and Saha, A. Finding multiple solutions for multimodal optimization problems using a multi-objective evolutionary approach. In Proceedings of the 12th annual conference on Genetic and evolutionary computation, 2010.

Dupont, E. Learning disentangled joint continuous and discrete representations. In Advances in Neural Information Processing Systems 31 (NIPS 2018), 2018.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. Diversity is all you need: Learning skills without a reward function. In Proceedings of the International Conference on Learning Representations (ICLR), 2019.

Florensa, C., Duan, Y., and Abbeel, P. Stochastic neural networks for hierarchical reinforcement learning. In Proceedings of the International Conference on Learning Representations (ICLR), 2017.

Fujimoto, S., van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In Dy, J. and Krause, A. (eds.), Proceedings of the International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pp. 1587–1596. PMLR, 2018. URL http://proceedings.mlr.press/v80/fujimoto18a.html.

Goldberg, D. E. and Richardson, J. Genetic algorithms with sharing for multimodal function optimization. In Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application, 1987.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In Advances in Neural Information Processing Systems (NeurIPS), 2014.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Proceedings of the International Conference on Machine Learning (ICML), 2018.

Hansen, N. and Ostermeier, A. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In Proceedings of the IEEE International Conference on Evolutionary Computation, 1996.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. beta-vae: Learning basic visual concepts with a constrained variational framework. In Proceedings of the International Conference on Learning Representations (ICLR), 2016.
Osa, T. and Sugiyama, M. Hierarchical policy search via return-weighted density estimation. In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), 2018.

Osa, T., Pajarinen, J., Neumann, G., Bagnell, J. A., Abbeel, P., and Peters, J. An algorithmic perspective on imitation learning. Foundations and Trends® in Robotics, 7(1-2):1–179, 2018.

Osa, T., Tangkaratt, V., and Sugiyama, M. Hierarchical reinforcement learning via advantage-weighted information maximization. In Proceedings of the International Conference on Learning Representations (ICLR), 2019.

Peng, X. B., Abbeel, P., Levine, S., and van de Panne, M. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. ACM Trans. Graph., 37(4):143:1–143:14, July 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201311. URL http://doi.acm.org/10.1145/3197517.3201311

Sharma, A., Gu, S., Levine, S., Kumar, V., and Hausman, K. Dynamics-aware unsupervised discovery of skills. In Proceedings of the International Conference on Learning Representations (ICLR), 2020.

Sharma, M., Sharma, A., Rhinehart, N., and Kitani, K. M. Directed-info gail: Learning hierarchical policies from unsegmented demonstrations using directed information. In Proceedings of the International Conference on Learning Representations (ICLR), 2019.

Shen, Y., Gu, J., Tang, X., and Zhou, B. Interpreting the latent space of gans for semantic face editing. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 9240–9249, 2020.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. Nature, 529(7587):484–489, 2016.

Sorokin, D., Ulanon, A., Sazhina, E., and Lvovsy, A. Interferobot: aligning an optical interferometer by a reinforcement learning agent. In Advances in Neural Information Processing Systems, 2020.

Stoean, C., Preuss, M., Stoean, R., and Dumitrescu, D. Multimodal optimization by means of a topological speciesconservation algorithm. IEEE Transactions on EvolutionaryComputation, 14(6):842–864, 2010.

Sutton, R. S. and Barto, A. G. Reinforcement Learning: An Introduction. MIT Press, second edition, 2018.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 5026–5033, 2012.
When plotting the learning curve, we performed the 10 episodes without exploration. $r$ was computed once every 5000 time steps by executing $\text{yielded the best performance on the test MDP was de-}$

$\text{fined as}$ $\text{and is given by}$ $\text{Hopper task in OpenAI Gym is given by}$ $\text{reward functions of the MuJoCo tasks were modified}$ $\text{based on the study by Kumar et al. (2020).}$ $\text{In the training MDPs, we used the modified version}$ $\text{five episodes}. In few-shot adaptation tasks, we trained a latent-conditioned policy on a training MDP five times with different random seeds. In a test MDP, five policies trained with different random seeds were adopted. For each random seed, the value of the latent value that yielded the best performance on the test MDP was determined after $k$ episodes in the test MDP; subsequently, the performance of the selected policy was evaluated in five episodes.

A.1 Details of Training MDPs

In the training MDPs, we used the modified version of MuJoCo tasks implemented in OpenAI Gym. The reward functions of the MuJoCo tasks were modified based on the study by Kumar et al. (2020).

**HopperVel task** The reward function of the original Hopper task in OpenAI Gym is given by

$$r = r_{\text{ctrl}} + r_{\text{vel}} + r_{\text{survive}},$$  \hspace{1cm} (25)

where $r_{\text{ctrl}}$ is the control cost defined as

$$r_{\text{ctrl}} = -1e^{-3} \|a\|^2_2$$  \hspace{1cm} (26)

$r_{\text{vel}}$ is the term for encouraging the agent to walk faster and is given by

$$r_{\text{vel}} = (x_t - x_{t-1})/\Delta t$$  \hspace{1cm} (27)

$r_{\text{survive}}$ is the term for avoiding falling down and is defined as

$$r_{\text{survive}} = \begin{cases} 1 & \text{if } s_t \text{ is not terminal} \\ 0 & \text{otherwise} \end{cases}.$$  \hspace{1cm} (28)

For the HopperVel task, we modified $r_{\text{vel}}$ as follows:

$$r_{\text{vel}} = \min((x_t - x_{t-1})/\Delta t, 1).$$  \hspace{1cm} (29)

**Walker2dVel task** The reward function of the original Walker2d task in OpenAI Gym is given by

$$r = r_{\text{ctrl}} + r_{\text{vel}} + r_{\text{survive}},$$  \hspace{1cm} (30)

where $r_{\text{ctrl}}$ is the control cost given by

$$r_{\text{ctrl}} = -1e^{-3} \|a\|^2_2$$  \hspace{1cm} (31)

$r_{\text{vel}}$ is the term for encouraging the agent to walk faster and is defined as

$$r_{\text{vel}} = (x_t - x_{t-1})/\Delta t$$  \hspace{1cm} (32)

$r_{\text{survive}}$ is the term for avoiding falling down and is defined as

$$r_{\text{survive}} = \begin{cases} 1 & \text{if } s_t \text{ is not terminal} \\ 0 & \text{otherwise} \end{cases}.$$  \hspace{1cm} (33)

For the Walker2dVel task, we modified $r_{\text{vel}}$ as follows:

$$r_{\text{vel}} = \min((x_t - x_{t-1})/\Delta t, 2).$$  \hspace{1cm} (34)

**HumanoidVel task** The reward function of the original Humanoid task in OpenAI Gym is given by

$$r = r_{\text{ctrl}} + r_{\text{vel}} + r_{\text{survive}} + r_{\text{impact}},$$  \hspace{1cm} (35)

where $r_{\text{ctrl}}$ is the control cost defined as

$$r_{\text{ctrl}} = -0.1 \|a\|^2_2$$  \hspace{1cm} (36)

$r_{\text{vel}}$ is the term for encouraging the agent to walk faster and is given by

$$r_{\text{vel}} = 0.25(x_t - x_{t-1})/\Delta t$$  \hspace{1cm} (37)

$r_{\text{survive}}$ is the term for avoiding falling down and is defined as

$$r_{\text{survive}} = \begin{cases} 5 & \text{if } s_t \text{ is not terminal} \\ 0 & \text{otherwise} \end{cases}.$$  \hspace{1cm} (38)

$r_{\text{impact}}$ is given by

$$r_{\text{impact}} = -\min(5e^{-6} \| f_{\text{ext}} \|^2_2, 10),$$  \hspace{1cm} (39)

$f_{\text{ext}}$ is the force exerted by the floor. For the HumanoidVel task, we modified $r_{\text{vel}}$ to

$$r_{\text{vel}} = 0.25 \min((x_t - x_{t-1})/\Delta t, 4).$$  \hspace{1cm} (40)

A.2 Description of Test MDPs for Few-Shot Adaptation

The agents used in the test MDPs are shown in Figure 8.

**Hopper-Short task** In the Hopper-Short task, the link under the lower joint is shorter than that of the agent in the original Hopper task in OpenAI Gym.
**Hopper-ShortShort task** In the Hopper-ShortShort task, the link under the lower joint is shorter than that of the agent in a original Hopper task in OpenAI Gym. In addition, the link between the upper and lower joints is shorter than that of the agent in the original Hopper task in OpenAI Gym.

**Hopper-HighKnee task** In the Hopper-HighKnee task, the link under the lower joint is longer, and the link between the upper and lower joints is shorter than that of the agent in a original Hopper task in OpenAI Gym.

**Hopper-LowKnee task** In the Hopper-LowKnee task, the link under the lower joint is shorter, and the link between the upper and lower joints is longer than that of the agent in the original Hopper task in OpenAI Gym.

**Hopper-LongHead task** In the Hopper-LongHead task, the link above the upper joint is longer than that of the agent in the original Hopper task in OpenAI Gym.

**Walker2d-Short task** In the Walker2d-Short task, the link under the knee joint of the red leg is shorter than that of the agent in the original Walker2d task in OpenAI Gym.

**Walker2d-ShortOrange task** In the Walker2d-ShortOrange task, the link under the knee joint of the orange leg is shorter than that of the agent in the original Walker2d task in OpenAI Gym.

**Walker2d-Asym1 task** In the Walker2d-Asym1 task, the position of the knee joint of the red and orange legs are higher and lower than that of the agent in the original Walker2d task in OpenAI Gym, respectively.

**Walker2d-Asym2 task** In the Walker2d-Asym2 task, the position of the knee joint of the red and orange legs are lower and higher than that of the agent in the original Walker2d task in OpenAI Gym, respectively.

**Walker2d-LowKnee task** In the Walker2d-LowKnee task, the positions of the knee joints of both the red and orange legs are lower than that of the agent in the original Walker2d task in OpenAI Gym.

**B.2 Diverse Behaviors on Walker2dVel**

Figure 10 shows the results with a two-dimensional continuous latent variable as well as those with a one-dimensional continuous latent variable and the three categorical latent variables on the Walker2dVel task. In the behaviors shown in Figure 10(a), channel $z_0$ encoded the knee angle of the red leg, whereas channel $z_1$ encoded the swing of two legs. The knee of the red leg was bent when $z_0 = -0.9$, whereas the knee of the red leg was straight when $z_0 = 0.9$. When $z_1 = 0.9$, the agent only stepped with the orange leg, whereas the agent walked with two legs when $z_1 = -0.9$. In the behaviors shown in Figure 10(b), the categorical latent variable encoded different types of walking behaviors. For all categorical latent variables, the agent leaned more forward when the value of the continuous latent variable increased.

**B.3 Diverse Behaviors on HumanoidVel**

Figure 11 shows the results with a two-dimensional continuous latent variable and those with a one-dimensional continuous latent variable and the three categorical latent variables on the HumanoidVel task. In the behaviors shown in Figure 11(a), channel $z_0$ encoded the walking direction whereas channel $z_1$ encodes the angle of the right arm. For example, the right arm was stretched when $z = [0.0, -0.9]$, whereas the right arm was bent when $z = [0.0, 0.9]$.

**B.4 Learning Curve with Different Dimensionality of Latent Variable**

The learning curves of LTD3, LSAC, SMERL, and SAC-DIAYN with different dimensionalities of the latent variable are shown in Figures 12–15, respectively. In SMERL, increasing the dimensionality of the latent variable resulted in an increase in the sample-complexity in the training phase, as shown in Figure 14. By contrast, in LTD3, increasing the dimensionality of the latent variable did not necessarily increase the sample complexity in the training phase. In the HumanoidVel task, the variance of the return of the policy with the two-dimensional latent variable was lower than that of the policy with the one-dimensional latent variable. However, when the latent variable was discrete, training with LSAC was relatively unstable compared with training with LTD3.
B.5 The Clipping Parameter for Importance Weight

The learning curves of LTD3 with different values of the clipping parameter are shown in Figures 16 and 17.

B.6 The frequency of minimizing the information loss

The learning curves of LTD3 with different values of $d_{\text{info}}$ are shown in Figure 18. We observed a trade-off between the average return and the diversity of the solutions. If the information loss $J_{\text{info}}$ is updated more often, then more diverse behaviors are obtained, but the average return decreases.

B.7 Few-shot Adaptation Task

The results on few-shot adaptation tasks with the Hopper and Walker2d agents are shown in Table B.7. In both few-shot adaptation tasks, LTD3 and LSAC outperformed the baseline methods.

B.8 Hyperparameters for Training

The hyperparameters of our methods and the baseline methods used in the experiments are summarized in Tables 3–7. The hyperparameters used for SMERL and SAC-DIAYN were based on the study by Kumar et al. (2020).
Table 2: Results of few-shot adaptation. Mean and standard deviation of return on test MDPs are shown. “H-” and “W-” indicate “Hopper” and “Walker2d,” respectively.

| Test MDP | Budget k = 5 | Budget k = 25 |
|----------|--------------|---------------|
|          | H-Short      | H-ShorShort | H-HighKnee | H-LowKnee | H-LongHead |
| LTD3 (2d)| 1486.5 ± 348.1 | 1196.1 ± 115.2 | 1434.0 ± 650.7 | 1643.4 ± 565.8 | 854.6 ± 621.9 |
| LTD3 (1d)| 1692.8 ± 105.1 | 1215.6 ± 204.4 | 1077.1 ± 625.7 | 1892.0 ± 334.8 | 911.3 ± 659.5 |
| LTD3 (disc)| 1707.1 ± 341.1 | 1296.5 ± 273.1 | 1330.7 ± 687.9 | 1722.0 ± 586.7 | 1831.2 ± 351.0 |
| LSAC (2d)| 1828.7 ± 97.5 | 1014.3 ± 407.7 | 1593.3 ± 570.2 | 1967.6 ± 17.5 | 1113.6 ± 704.1 |
| LSAC (1d)| 1701.4 ± 285.0 | 939.7 ± 266.5 | 1531.2 ± 614.2 | 1978.6 ± 4.4 | 1275.4 ± 783.2 |
| LSAC (disc)| 20.1 ± 25.0 | 43.3 ± 69.3 | 23.4 ± 10.0 | 5.7 ± 1.5 | 17.4 ± 15.0 |
| SMERL (2d)| 1475.4 ± 379.8 | 921.6 ± 233.2 | 725.8 ± 235.8 | 1265.1 ± 401.2 | 646.7 ± 299.6 |
| SMERL (1d)| 1517.8 ± 620.7 | 627.5 ± 375.8 | 614.4 ± 366.8 | 1075.5 ± 524.2 | 422.6 ± 75.2 |
| SMERL (disc)| 1733.8 ± 201.1 | 1231.2 ± 238.7 | 489. ± 70.0 | 1728.1 ± 340.1 | 504.4 ± 86.9 |

| Test MDP | W-ShortRed | W-ShortOrange | W-Asym1 | W-Asym2 | W-LowKnee |
|----------|-------------|---------------|---------|---------|-----------|
|          | Budget k = 5 | Budget k = 25 |         |         |           |
| LTD3 (2d)| 1063.8 ± 822.1 | 1033.7 ± 863.2 | 1768.6 ± 1054.2 | 1405.0 ± 929.8 | 2546.4 ± 692.6 |
| LTD3 (1d)| 409.9 ± 419.8 | 1084.2 ± 731.9 | 2279.8 ± 899.9 | 2449.3 ± 676.3 | 2364.4 ± 959.6 |
| LTD3 (disc)| 790.4 ± 595.3 | 1195.6 ± 931.2 | 2266.5 ± 944.1 | 2589.7 ± 636.1 | 2725.7 ± 458.1 |
| LSAC (2d)| 573.2 ± 379.1 | 341.6 ± 106.7 | 2359.4 ± 953.1 | 2815.8 ± 62.0 | 2826.0 ± 83.8 |
| LSAC (1d)| 420.2 ± 262.5 | 884.5 ± 522.6 | 2835.5 ± 59.4 | 2848.4 ± 21.6 | 2868.7 ± 24.9 |
| LSAC (disc)| 265.2 ± 133.2 | 680.5 ± 957.9 | 2346.2 ± 755.1 | 2307.6 ± 795.1 | 2562.3 ± 744.7 |
| SMERL (2d)| 868.9 ± 582.6 | 675.4 ± 400.5 | 817.7 ± 267.1 | 828.6 ± 198.9 | 720.6 ± 244.6 |
| SMERL (1d)| 664.6 ± 578.7 | 658.7 ± 488.9 | 731.0 ± 342.2 | 1115.6 ± 827.0 | 1092.1 ± 727.5 |
| SMERL (disc)| 844.9 ± 608.0 | 679.8 ± 379.4 | 1512.7 ± 723.2 | 1534.8 ± 727.9 | 1279.9 ± 687.9 |

| Test MDP | W-ShortRed | W-ShortOrange | W-Asym1 | W-Asym2 | W-LowKnee |
|----------|-------------|---------------|---------|---------|-----------|
|          | Budget k = 25 |              |         |         |           |
| LTD3 (2d)| 1360.9 ± 1040.8 | 1585.0 ± 876.4 | 1991.0 ± 993.4 | 1696.2 ± 1008.8 | 2502.0 ± 597.5 |
| LTD3 (1d)| 363.8 ± 266.3 | 1147.8 ± 784.7 | 2322.1 ± 831.2 | 2417.2 ± 771.8 | 2719.7 ± 474.0 |
| LTD3 (disc)| 1519.8 ± 948.1 | 1685.0 ± 963.8 | 2428.6 ± 851.9 | 2603.7 ± 491.3 | 2790.7 ± 304.5 |
| LSAC (2d)| 650.5 ± 593.3 | 448.7 ± 137.6 | 2579.3 ± 695.3 | 2651.0 ± 666.6 | 2874.1 ± 28.1 |
| LSAC (1d)| 539.1 ± 486.4 | 919.0 ± 703.0 | 2782.6 ± 248.5 | 2865.4 ± 23.5 | 2871.4 ± 26.1 |
| LSAC (disc)| 1243.4 ± 1129.0 | 919.2 ± 707.5 | 2321.2 ± 962.6 | 2688.3 ± 601.8 | 2648.9 ± 557.8 |
| SMERL (2d)| 663.7 ± 304.9 | 656.9 ± 504.2 | 777.1 ± 263.4 | 779.7 ± 554.0 | 934.1 ± 414.7 |
| SMERL (1d)| 586.6 ± 505.2 | 543.5 ± 359.1 | 746.6 ± 720.4 | 1351.7 ± 851.7 | 1147.6 ± 769.8 |
| SMERL (disc)| 803.3 ± 197.5 | 1050.5 ± 646.4 | 2083.7 ± 558.5 | 1727.1 ± 828.9 | 1828.5 ± 878.3 |
(a) Results with two-dimensional continuous latent variable.

(b) Results with one-dimensional continuous latent variable and three categorical latent variable.

Figure 9: Behavior learned by LTD3 for HopperVel Task.

Figure 10: Behavior learned by LTD3 for Walker2dVel task.

Table 3: Hyperparameters used for TD3

| Parameter                  | Value         |
|----------------------------|---------------|
| Optimizer                  | Adam          |
| Learning Rate              | $3 \cdot 10^{-4}$ |
| Discount factor $\gamma$   | 0.99          |
| Replay buffer size         | $10^6$        |
| Number of hidden layers    | 2             |
| Number of hidden units per layer | 256 |
| Number of samples per minibatch | 256 |
| Activation Function        | ReLU          |
| Target smoothing coefficient| 0.005        |
| Gradient steps per time step | 1           |
| Interval for updating the policy | 2   |

Table 4: Hyperparameters used for SAC

| Parameter                  | Value         |
|----------------------------|---------------|
| Optimizer                  | Adam          |
| Learning Rate              | $3 \cdot 10^{-4}$ |
| Discount factor $\gamma$   | 0.99          |
| Replay buffer size         | $10^6$        |
| Number of hidden layers    | 2             |
| Number of hidden units per layer | 256 |
| Number of samples per minibatch | 256 |
| Activation Function        | ReLU          |
| Target smoothing coefficient| 0.005        |
| Gradient steps per time step | 1           |
| Temperature                | 0.2           |
Table 5: Hyperparameters used for SAC-DIAYN

| Parameter                        | Value       |
|----------------------------------|-------------|
| Optimizer                        | Adam        |
| Learning rate                    | $3 \cdot 10^{-4}$|
| Discount factor $\gamma$         | 0.99        |
| Replay buffer size               | $10^6$      |
| Number of hidden layers          | 2           |
| Number of hidden units per layer | 256         |
| Number of samples per minibatch  | 256         |
| Activation function              | Relu        |
| Target smoothing coefficient     | 0.005       |
| Gradient steps per time step     | 1           |
| Temperature for SAC              | 0.1         |
| Weight on unsupervised reward    | 0.5         |

Table 6: Hyperparameters used for SMERL

| Parameter                        | Value       |
|----------------------------------|-------------|
| Optimizer                        | Adam        |
| Learning rate                    | $3 \cdot 10^{-4}$|
| Discount factor $\gamma$         | 0.99        |
| Replay buffer size               | $10^6$      |
| Number of hidden layers          | 2           |
| Number of hidden units per layer | 256         |
| Number of samples per minibatch  | 256         |
| Activation function              | Relu        |
| Target smoothing coefficient     | 0.005       |
| Gradient steps per time step     | 1           |
| Temperature for SAC              | 0.1         |
| Weight on unsupervised reward    | 10.0        |
| Value of $\epsilon$              | 0.1         |

Table 7: Hyperparameters used for LTD3

| Parameter                        | Value       |
|----------------------------------|-------------|
| Optimizer                        | Adam        |
| Learning rate                    | $3 \cdot 10^{-4}$|
| Discount factor $\gamma$         | 0.99        |
| Replay buffer size               | $10^6$      |
| Number of hidden layers          | 2           |
| Number of hidden units per layer | 256         |
| Number of samples per minibatch  | 256         |
| Activation function              | Relu        |
| Target smoothing coefficient     | 0.005       |
| Gradient steps per time step     | 1           |
| Clipping param. for IW           | 0.3         |
| Interval for maximizing $J_Q(\theta)$ | 2          |
| Interval for maximizing $J_{info}(\theta)$ | 4          |

Table 8: Hyperparameters used for LSAC

| Parameter                        | Value       |
|----------------------------------|-------------|
| Optimizer                        | Adam        |
| Learning rate                    | $3 \cdot 10^{-4}$|
| Discount factor $\gamma$         | 0.99        |
| Replay buffer size               | $10^6$      |
| Number of hidden layers          | 2           |
| Number of hidden units per layer | 256         |
| Number of samples per minibatch  | 256         |
| Activation function              | Relu        |
| Target smoothing coefficient     | 0.005       |
| Gradient steps per time step     | 1           |
| Temperature for entropy           | 0.1         |
| Clipping param. for IW           | 0.2         |
| Interval for maximizing $J_Q(\theta)$ | 2          |
| Interval for maximizing $J_{info}(\theta)$ | 3          |
Figure 12: Learning curve of LTD3 with different dimensionalities of latent variable.

Figure 13: Learning curve of LSAC with different dimensionalities of latent variable.

Figure 14: Learning curve of SMERL with different dimensionalities of latent variable.

Figure 15: Learning curve of SAC-DIAYN with different dimensionalities of latent variable.
Figure 16: Effect of clipping parameter on LTD3. Results the two-dimensional continuous latent variable.

Figure 17: Effect of clipping parameter on LSAC. Results with two-dimensional continuous latent variable.

Figure 18: Effect of $d_{\text{info}}$ in LTD3. Results with two-dimensional continuous latent variable. Clipping parameter for the importance weight was $c_{\text{clip}} = 0.3$.

Figure 19: Effect of $d_{\text{info}}$ in LSAC. Results with two-dimensional continuous latent variable. Importance weight is not used.