Context-Based Meta-Reinforcement Learning With Bayesian Nonparametric Models

Zhenshan Bing, Yuqi Yun, Kai Huang, and Alois Knoll, *Fellow, IEEE*

***Abstract*—Deep reinforcement learning agents usually need to collect a large number of interactions to solve a single task. In contrast, meta-reinforcement learning (meta-RL) aims to quickly adapt to new tasks using a small amount of experience by leveraging the knowledge from training on a set of similar tasks. State-of-the-art context-based meta-RL algorithms use the context to encode the task information and train a policy conditioned on the inferred latent task encoding. However, most recent works are limited to parametric tasks, where a handful of variables control the full variation in the task distribution, and also failed to work in non-stationary environments due to the few-shot adaptation setting. To address those limitations, we propose MEta-reinforcement Learning with Task Self-discovery (MELTS), which adaptively learns qualitatively different nonparametric tasks and adapts to new tasks in a zero-shot manner. We introduce a novel deep clustering framework (DPMM-VAE) based on an infinite mixture of Gaussians, which combines the Dirichlet process mixture model (DPMM) and the variational autoencoder (VAE), to simultaneously learn task representations and cluster the tasks in a self-adaptive way. Integrating DPMM-VAE into MELTS enables it to adaptively discover the multi-modal structure of the nonparametric task distribution, which previous methods using isotropic Gaussian random variables cannot model. In addition, we propose a zero-shot adaptation mechanism and a recurrence-based context encoding strategy to improve the data efficiency and make our algorithm applicable in non-stationary environments. On various continuous control tasks with both parametric and nonparametric variations, our algorithm produces a more structured and self-adaptive task latent space and also achieves superior sample efficiency and asymptotic performance compared with state-of-the-art meta-RL algorithms.**

***Index Terms*—Meta-reinforcement learning, task inference, task adaptation, Bayesian nonparametric model, robotic control.**

I. Introduction

Reinforcement learning (RL) methods have demonstrated their ability to learn specific tasks and even surpass human performance. However, they are still limited in generalizing knowledge to new tasks compared to humans. Humans can innately leverage past experiences to accomplish new and similar tasks. For example, those who are good at skiing may find it easy to learn snowboarding, as they have already mastered maneuvering on snow. Instead of training on each task individually, meta-RL trains on a group of tasks, usually a distribution of Markov decision processes (MDPs) with the same state and action space. Thereafter, it can learn to perform new tasks from a similar task distribution using only a small number of interactions. Meta-RL algorithms are usually classified into three types, namely, gradient-based [1], [2], recurrence-based [3], [4], and context-based [5], [6]. Across these three categories, recent context-based meta-RL algorithms, such as PEARL [5], have been shown to achieve superior sample efficiency and asymptotic performance [7].

Context-based meta-RL involves training a universal policy that is conditioned on a probabilistic task embedding, consisting of a task inference network for learning the embedding and a policy network conditioned on the inferred task belief. For instance, PEARL utilizes a variational auto-encoder to encode historical data into latent task representations, enabling online probabilistic inference for the tasks and policy network conditioning [5]. Task distributions are categorized as parametric or nonparametric [8], where parametric tasks involve specific parameter variations, while nonparametric tasks are qualitatively distinct and cannot be described by parameter variations. Previous context-based meta-RL approaches have been effective for parametric tasks but struggle with generalizing to diverse nonparametric tasks due to the limited expressiveness of single Gaussian priors in modeling multi-modal structure.

We can view learning task representations while discovering a latent structure for nonparametric tasks as a problem of *simultaneous representation learning and clustering*, also known as *deep clustering* [9]. By formalizing the concept of parametric and nonparametric variability using Gaussian mixture models (GMMs), our recent algorithm, CEMRL, is applicable to a variety of meta-RL settings, such as non-stationary environments and broad task distributions [7]. However, CEMRL is still limited in two aspects: (1) it assumes that the number of clusters (mixtures) is known, but this information may be hidden in meta-RL; (2) the number of clusters in GMMs is fixed so it cannot adapt dynamically when there is a new task cluster. Note that a stationary environment refers to an MDP that is fixed during an episode and only changed between episodes. A non-stationary environment, in contrast, refers to an MDP that can potentially change every timestep. Therefore, an algorithm must perform zero-shot adaptation to solve such an environment [10].

In this work, we design an approach to leverage deep clustering and Bayesian nonparametric models to tackle the challenge

Manuscript received 9 January 2023; revised 28 March 2024; accepted 7 April 2024. Date of publication 9 April 2024; date of current version 5 September 2024. Recommended for acceptance by M. Sugiyama. *(Corresponding author: Kai Huang.)*

Zhenshan Bing, Yuqi Yun, and Alois Knoll are with the Department of Informatics, Technical University of Munich, 80333 München, Germany (e-mail: bing@in.tum.de; yuqi.yun@tum.de; knoll@in.tum.de).

Kai Huang is with the School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou 510275, China (e-mail: huangk36@mail.sysu.edu.cn).

Digital Object Identifier 10.1109/TPAMI.2024.3386780

of meta-RL with broad nonparametric task distributions. We propose **ME**ta-reinforcement **l**earning with **T**ask **S**elf-discovery (MELTS), an off-policy context-based meta-RL algorithm that can discover both parametric and nonparametric tasks in a self-adaptive way. This algorithm also works in non-stationary environments and adapts in a zero-shot manner. The novelty of our algorithm is based on two concepts: (1) We propose a simultaneous representation learning and clustering method (DPMM-VAE) that combines the Dirichlet process mixture model (DPMM) and the variational autoencoder (VAE). Compared with previous deep clustering methods, our method can self-adapt to the varying complexity of the data, scale to an infinite mixture, and additionally learn the distribution of each cluster. DPMM-VAE enables MELTS to automatically identify nonparametric tasks and generate a more structured latent space than previous context-based meta-RL algorithms using a single Gaussian model or a Gaussian mixture model. (2) We propose a novel recurrent encoder-decoder strategy for learning task embeddings. We train our task-inference network by recurrently encoding several transitions and reconstructing the task's Markov decision process. Previous meta-RL methods using recurrent networks [3], [4], [6] maintain the hidden state over the whole transition history of a task. We demonstrate that enough task information can be learned from just a few transitions and subsequently make our recurrent encoder-decoder compatible with non-stationary environments.

We evaluate the DPMM-VAE on the MNIST dataset, and it achieves superior prediction accuracy compared with other types of VAE. We evaluate MELTS on continuous robotic control tasks with parametric and nonparametric distributions. Experimental results show that MELTS significantly outperforms state-of-the-art meta-RL methods in terms of sample efficiency and asymptotic performance. To the best of our knowledge, MELTS is the first meta-RL algorithm that can explicitly identify qualitatively distinct tasks without any prior information about the task distribution. We provide an open-source implementation of our framework and encourage readers to further explore and extend our work on this basis.[1]

II. Background

A. Meta-Reinforcement Learning

Meta-learning algorithms seek to learn how to perform various tasks without training for each task from scratch. Meta-learning is regarded as a promising step toward achieving strong and versatile AI [11], [12]. The fundamental principle underlying meta-learning is often referred to as "learning to learn". The objective is to train a meta-learner that can extract commonalities across a set of training tasks. By leveraging the acquired meta-information from these tasks, the meta-learner can then facilitate more efficient learning of related tasks. Typically, an inner loop employing a standard learner is employed to learn the specifics of each task, a process commonly referred to as "adaptation" in the literature. Simultaneously or iteratively, the outer loop focuses on training the meta-learner. The optimization of both the inner and outer loops depends on the specific algorithm being employed. In the context of meta-RL, a task is represented as a Markov Decision Process (MDP) and is assumed to be drawn from a task distribution denoted as $\mathcal{T}_i \sim p(\mathcal{T})$:

$$\mathcal{T}_i = \{S^i, A^i, R^i(s,a), p^i(s,a), p^i(s_0)\}. \tag{1}$$

Here, $\mathcal{T}_i$ represents the MDP corresponding to the i-th task, with S^i and A^i denoting the state and action spaces, respectively. The functions $R^i(s,a)$, $p^i(s,a)$, and $p^i(s_0)$ describe the reward function, transition dynamics, and initial state distribution for the i-th task, respectively. In meta-RL, we have separate task sets, $D_{\mathcal{T}_{train}}$ and $D_{\mathcal{T}_{test}}$, which contain different tasks with no overlap. Meta-RL usually contains two phases: During *meta-training*, we only access the tasks in $D_{\mathcal{T}_{train}}$. During *meta-testing*, we test how quickly and how well our meta-RL agent can adapt to tasks in $D_{\mathcal{T}_{test}}$ that are not seen during meta-training. Therefore, the optimization objective of the policy Π is based on the performance on $D_{\mathcal{T}_{test}}$:

$$\Pi^* = \underset{\Pi}{\operatorname{argmax}}\, \mathbb{E}_{\mathcal{T}\sim D_{\mathcal{T}_{test}}}\left[\mathbb{E}_{\boldsymbol{\tau}\sim p(\boldsymbol{\tau}|\Pi)}\left[\sum_t \gamma^t r_t\right]\right], \tag{2}$$

where $\boldsymbol{\tau} = (\boldsymbol{s_t}, \boldsymbol{a_t}, r_t, \boldsymbol{s_{t+1}})$ contains the state, action, reward, and the next state. γ represents the discount factor, with values ranging from 0 to 1.

B. Parametric & Non-Parametric Task Distribution

The configurations of the task distribution can also vary, and we categorize them into two distinct groups.

- *Parametric task distribution:* In a task distribution characterized by parametric variation, a collection of Markov Decision Processes (MDPs) exhibits significant commonalities. The disparities among these MDPs, including the state space S, action space A, transition probability $p(s,a)$, and reward function $R(s,a)$, can be expressed through a set of continuous parameters. For instance, tasks involving a robot running at various velocities represent parametric variation, as their differences primarily rely on the reward functions, which can be fully described by the velocity parameter.
- *Non-parametric task distribution:* A task distribution demonstrating non-parametric variation is less uniform and presents greater challenges. This type of distribution encompasses qualitatively distinct tasks that cannot be distinguished solely by continuous parameters. For instance, in the Meta-World benchmark [8], the discrepancy between "reach puck" tasks and "open window" tasks represents non-parametric variation. Although we can parameterize all "reach puck" tasks using the puck position, the same parametrization cannot be applied to "open window" tasks.

To build a task distribution that facilitates training for generalization across a wide array of tasks, it is essential to include both parametric and non-parametric variations. In such a distribution, each qualitatively-different non-parametric *base task* is accompanied by a set of *subtasks* exhibiting parametric variation.

[1]https://github.com/Ghiara/MELTS

III. Related Work

A. Simultaneous Representation Learning and Clustering

Simultaneous representation learning and clustering, referred to as *deep clustering*, deals with the problem of learning through a deep neural network while discovering the clusters of the data at the same time. It is one of the most promising approaches in unsupervised learning [13]. The benefit of this method is usually shown in self-supervised learning, where we can generate pseudo-labels or surrogate tasks from unlabeled data based on the learned clustering assignments. We can categorize the methods in this domain by whether they use two-step optimization or one-step optimization. With two-step optimization, Xie et al. [14] use a k-means model with a Student's t-distribution as a kernel to measure the distance between feature embeddings and the cluster centroids, and then alternately optimize the neural network parameters and the cluster centroids, leveraging soft cluster assignments. DeepCluster [15] uses a k-means model, and it alternates between optimizing the clustering objective and improving the learned representations via cross-entropy loss.

The second category of methods based on one-step optimization does not seek to combine neural networks with a standard clustering framework like k-means. Instead, they design a single objective to push similar data closer together. Notable examples are invariant information clustering (IIC) [16], in which the objective is the maximal mutual information between the cluster assignment of data pairs, and each data pair consists of the original data point and the augmented version representing the same instance. Deep robust clustering (DRC) [17] turns maximizing mutual information into minimizing constructive loss and shows that maximizing the similarities for positive pairs and minimizing them for negative pairs can lead to robust clustering results. Although the one-step optimization approach circumvents the problem of error propagation in the alternative two-step optimization, it has the drawback of relying mainly on data augmentation to construct data pairs, which can be difficult when learning in an RL setting. It also does not produce parametrization for the learned cluster, which reduces the method's ability to reason on uncertainties. For this reason, we also use the two-step optimization strategy in our work.

Bayesian nonparametric models have been incorporated into *deep clustering*. Stick-breaking VAE [18] replaces the parameters of the isotropic Gaussian prior with the stick-breaking proportions of a Dirichlet process. Each latent dimension is a line segment of the stick and can therefore be viewed as its own cluster. However, the line segment informs only about the cluster membership, not the shape and density of the individual cluster. Besides, the decision to embed a Dirichlet process directly in the network architecture means that the variational inference methods for learning the Dirichlet process are limited to stochastic gradient variational Bayes, which may suffer from noisiness and a local optimum [19]. Deep nonparametric Bayes (DNB) [20] directly incorporates a Dirichlet process mixture model and adopts the two-step optimization approach. It has an additional step of self-labeling, and the network parameters are updated using the loss with pseudo-labels. Our method is also based on DPMM, but we eliminate the self-labeling step and keep the reconstruction loss of the standard VAE. In our work, the learned cluster assignments only modulate the VAE through the KL divergence.

B. Meta-Reinforcement Learning

In recent works, most meta-RL algorithms fall into three main categories.

Recurrence-based methods utilize recurrent neural networks (RNNs) such as LSTM or GRU to implicitly retain task information in hidden states. They can work in conjunction with on-policy RL algorithms. An example of this category is RL^2 [3], which uses GRU hidden states to encode relevant task information over time steps. Our work employs a GRU-based encoder as well, but we do not maintain hidden states over episodes like RL^2. Instead, we obtain "fresh" hidden states from local contexts of a few time steps, enabling applicability to non-stationary environments. In general, recurrence-based meta-RL algorithms do not have an explicit task-belief, and they are on-policy and comparably sample-inefficient.

Gradient-based meta-RL algorithms learn effective gradient descent rules to facilitate quick adaptation to new tasks during meta-testing. MAML [1] is a prominent representative, aiming to learn a highly sensitive weight initialization for rapid adaptation with few gradient steps. In this way, the model is expected to learn a good initialization of the neural network without overfitting on any particular task. This initialization serves as an inductive bias for fast adaptation to new tasks. MAML [1] has been extended to facilitate continuous adaptation in non-stationary environments [2], [21]. In the GrBAL algorithm [2], the transition from episodic adaptation to zero-shot adaptation involves considering each segment of M time steps as a distinct task. We employ a similar idea to achieve zero-shot adaptation in our work.

Context-based meta-RL explicitly models the belief over the task distribution based on essential task information. This approach enables online probabilistic inference to identify new tasks and act optimally accordingly. Many recent works have leveraged the advantages of learning latent embeddings to capture diverse skills. Notable algorithms in this category include PEARL [5] and VariBAD [6], decouple the learning of task representation from the policy training. MetaCURE [22] further separates the policy learning of PEARL into an explorer and an exploiter, where the explorer specializes in collecting experiences with rich task-relevant information to improve task inference. These context-based methods use off-policy learning, substantially improving sampling efficiency. A limitation of the aforementioned models is their reliance on isotropic Gaussian priors, which may prove inadequate when dealing with complex tasks exhibiting nonparametric variation. One possible approach to address this limitation is to employ a graph neural network architecture in the encoder, as proposed by Wang et al. [23]. Alternatively, Ren et al. [24] suggested the use of Dirichlet random variables for the task latent space to accommodate the multi-modality of task distributions. However, using the Dirichlet prior requires a pre-defined number of base tasks K. Ren's model was evaluated on a point-robot navigation task, which only involved parametric variation. Our work extends on this approach by proposing a model where the number of base tasks is unknown and can be self-discovered by the model, allowing for evaluation with more complex nonparametric task distributions.

Our work is also related to other studies that combine Bayesian nonparametrics with meta-learning. While MAML focuses on learning a single set of weight initialization during meta-training, there are extensions proposed by Nagabandi et al. [25], Jerfel et al. [26], and Wang et al. [27] that incorporate a mixture model to handle different tasks' model parameters. These works employ Bayesian nonparametric methods, specifically the Chinese restaurant process (CRP), to control the mixture model. Additionally, they optimize the mixture model jointly with the MAML algorithm using the expectation-maximization (EM) procedure.

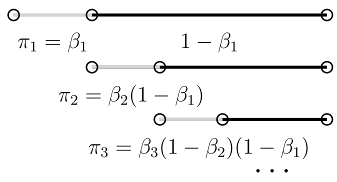
IV. Bayesian Nonparametrics Preliminaries

This section first presents a brief introduction to Bayesian nonparametrics, which serves as the theoretical foundation of our deep clustering method DPMM-VAE. Specifically, we start with the Dirichlet process and define it constructively via the stick-breaking process. We then show how to extend the finite mixture to an infinite mixture with the Dirichlet process mixture model. Finally, we explain the memoized online variational inference, which is used in large-scale training of Bayesian nonparametric models.

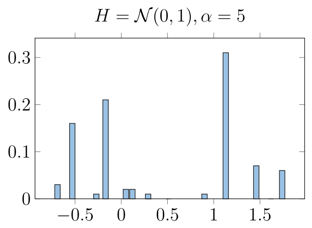
A. Bayesian Nonparametrics

Traditional parametric probabilistic models are concerned with learning a finite set of parameters within a model class to explain the observed data. They have been a popular choice for machine learning. However, they can only have a fixed number of parameters, which leads to several limitations. First, in many parametric models, we require the distributions to lie within a certain parametric family. Second, model misspecification in these cases can lead to inconsistent estimators of parameters. Third, the number of parameters in the model is also usually fixed. This means that we need to select the proper number of parameters via model selection, which can be cumbersome. Concrete examples include choosing the number of components in a Gaussian mixture model or choosing k for k-means clustering. If we specify the wrong number of components, the model can be over-fitting or under-fitting. Moreover, if we observe more data that provide additional information, a parametric model with a fixed number of parameters cannot adjust its model complexity to adapt to the increased data complexity or the changing data structure. Therefore, these models are not a natural choice for meta-learning, continual learning, or lifelong learning.

Bayesian nonparametric models have been developed to overcome these limitations. "Nonparametric" here does not mean the model has no parameters. Rather, the model has the capacity to use an infinite set of parameters. The prior distributions for the nonparametric models usually have the space of all distributions as their support and therefore are not limited to particular parametric families [28]. With the possible set of parameters being infinite, the nonparametric models can decide how many parameters to utilize given the data, and the model should be able to adapt with the growing data. In the following, we briefly introduce the Bayesian nonparametric models and the variational inference methods, and readers are encouraged to read the references for more details.


$\pi_1 = \beta_1$
$1-\beta_1$
$\pi_2 = \beta_2(1-\beta_1)$
$\pi_3 = \beta_3(1-\beta_2)(1-\beta_1)$
...

(a) Stick-breaking process.



(b) Draw from a Dirichlet process.

Fig. 1. (a) Stick-breaking process, where we break π_i off the stick at the i-th iteration. (b) Histogram based on draws from a Dirichlet process with $\alpha = 5$ and $H = \mathcal{N}(0, 1)$.

B. Dirichlet Process and Stick-Breaking

The Dirichlet process (DP) is a distribution over probability measures, where its marginal distribution has to be Dirichlet-distributed. Hence, draws from a DP are random distributions. Let H be a distribution over the sample space Θ and α be a positive scalar. A random probability distribution G is distributed according to a DP with the base distribution H and concentration parameter α, i.e., $G \sim \mathrm{DP}(\alpha, H)$. We refer to [28] for a complete explanation of the basic properties of DPs.

Dirichlet process mixture models (DPMMs) are commonly used to tackle the problem of the infinite mixture. For $k \geq 1$, we define a DP constructively using the so-called stick-breaking (SB) process [29]:

$$\beta_k \sim \mathcal{B}(1, \alpha), \qquad \pi_k = \beta_k \prod_{i=1}^{k-1} (1 - \beta_i). \tag{3}$$

This constructive definition derives from imaginatively breaking up a unit-length stick into an infinite number of segments π_k, with α being a positive scalar. First, we sample $\beta_1 \sim \mathcal{B}(1, \alpha)$ from a Beta distribution and break the stick at $\pi_1 = \beta_1$. Then we sample β_2, and the length of the second segment will be $\pi_2 = \beta_2(1 - \beta_1)$. Continuing this process, we will have $\sum_{k=1}^{\infty} \pi_k = 1$, and the resulting π follows a Griffiths-Engen-McCloskey (GEM) distribution $\pi \sim \mathrm{GEM}(\alpha)$. Fig. 1(a) demonstrates the SB process.

A random distribution G drawn from a DP is discrete, so we can write it as a weighted sum of point masses [30], [31]:

$$G = \sum_{k=1}^{\infty} \pi_k \delta_{\theta_k^*}, \tag{4}$$

where $\delta_{\theta_k^*}$ is the point mass located at θ_k^*: it equals 1 at θ_k^* and equals 0 everywhere else. If we sample the weights π_k according to (3) and sample θ_k^* from a base distribution $\theta_k^* \sim H$, then we can say that $G \sim \mathrm{DP}(\alpha, H)$, that is to say, G is a

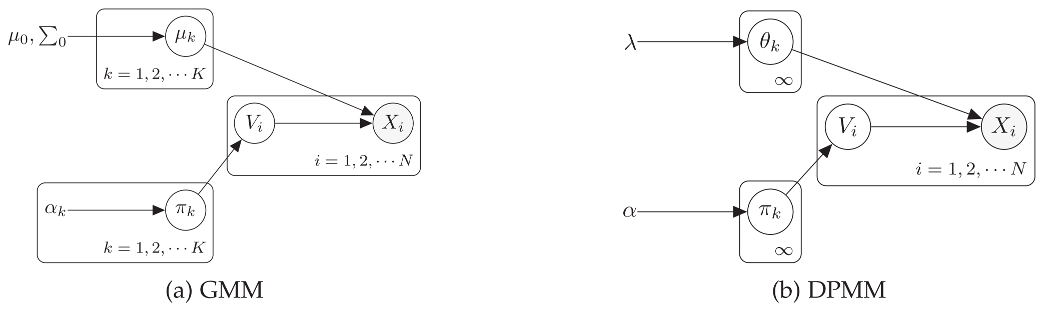


Fig. 2. Probabilistic graphic model of (a) the Gaussian mixture model and (b) the Dirichlet process mixture model. Compared with GMM that can has fixed numbers of clusters, DPMM can model an infinite number of clusters.

Dirichlet process with the base distribution H and concentration parameter α. Fig. 1(b) shows a draw from a DP with $\alpha = 5$.

C. Dirichlet Process Mixture Model

The Dirichlet process mixture model (DPMM) is one of the most widely used Bayesian nonparametric models. It is an infinite mixture model where the number of cluster components is decided by the data instead of being pre-specified. A set of observations $\boldsymbol{x} = \{x_1, \ldots, x_N\}$ is modeled by a set of latent parameters $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_N\}$, where θ_i is independently draw G. Here we assume x_i has distribution $F(\theta_i)$ parameterized by θ_i. Then we have:

$$\begin{aligned} G|\alpha, H &\sim \text{DP}(\alpha, H), \\ \theta_i|G &\sim G, \\ x_i|\theta_i &\sim F(\theta_i). \end{aligned} \tag{5}$$

Because of the discreteness and the clustering property of the Dirichlet process G, θ_i will take on repeated values, and we will obtain a clustering on all the θ_i's. Then all x_i's drawn with the same value of θ_i can be seen as one cluster. The number of unique values of θ_i's is the active number of cluster components for a DPMM. During inference, the active number of components can be dynamically inferred from the observed data.

Let v_i be a cluster assignment variable, which takes on value k with probability π_k that is drawn from a categorical distribution (Cat). Then (5) can be equivalently expressed via the stick-breaking process [28] as follows:

$$\begin{aligned} \theta_k^*|H &\sim H, \\ \pi|\alpha &\sim \text{GEM}(\alpha), \\ v_i|\pi &\sim \text{Cat}(\pi), \\ x_i|v_i, &\sim F(\theta_{v_i}^*). \end{aligned} \tag{6}$$

Here we assume x_i has distribution $F(\theta_{v_i}^*)$, and $\theta_{v_i}^*$ are the parameters that parameterize F. In mixture modeling terminology, π is the mixing proportion, θ_k^* are the cluster parameters, $F(\theta_{v_i}^*)$ is the distribution over data in cluster k, and H is the prior over cluster parameters. Typically, F is a Gaussian distribution. The main difference between GMM and DPMM is that for DPMM, the mixing proportion $\boldsymbol{\pi}$ is sampled from a GEM distribution, and the prior $H(\lambda)$ over the cluster parameters is the base distribution of an underlying Dirichlet process $G(\alpha, H)$. Fig. 2 illustrates this generative model of the GMM and DPMM.

D. Variational Inference for DPMM

So far, we have introduced the Dirichlet process mixture model (DPMM) as a generative model. In practice, however, we want to approximate the posterior density for the models based on the observed data. For a Gaussian mixture model, we can use the expectation-maximization (EM) algorithm to find the maximum likelihood estimates of the parameters. Inference for Bayesian nonparametric models is not as easy in comparison. Various methods have been developed using split-merge sampling [32], sequential Monte Carlo [33], and variational inference (VI) [34]. We focus on the variational methods since they are typically faster and more scalable compared with Monte Carlo methods.

The basic idea of variational inference is to convert the inference problem into an optimization problem. From (3) and (6), we write the joint probability for DPMM as:

$$\begin{aligned} p(\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta}) = &\prod_{n=1}^{N} F(x_n|\theta_{v_n}) \text{Cat}(v_n|\boldsymbol{\pi}(\boldsymbol{\beta})) \\ &\times \prod_{k=1}^{\infty} \mathcal{B}(\beta_k|1, \alpha) H(\theta_k|\lambda). \end{aligned} \tag{7}$$

When the true posterior $p(\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta}|\boldsymbol{x})$ is intractable, we aim to find the best variational distribution $q(\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta})$, such that the KL divergence with the exact conditional is minimized:

$$\begin{aligned} q^*(\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta}) = &\operatorname*{argmin}_q \mathbb{KL}(q(\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta}) \\ &\times ||p(\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta}|\boldsymbol{x})), \end{aligned} \tag{8}$$

$$\begin{aligned} \mathbb{KL}(q(\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta})||p(\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta}|\boldsymbol{x})) &= \mathbb{E}[\log q(\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta})] \\ &\quad - \mathbb{E}[\log p(\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta}|\boldsymbol{x})] \\ &= \mathbb{E}[\log q(\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta})] \\ &\quad - \mathbb{E}[\log p(\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta}, \boldsymbol{x})] \\ &\quad + \log p(\boldsymbol{x}). \end{aligned} \tag{9}$$

Notice that $\log p(\boldsymbol{x})$ does not depend on q, so instead of minimizing the KL divergence directly, we maximize the so-called evidence lower bound (ELBO), which is:

$$\text{ELBO}(q) = \mathbb{E}[\log p(\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta})] - \mathbb{E}[\log q(\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta})]. \quad (10)$$

We can rearrange the terms in the ELBO and rewrite it as follows:

$$\begin{aligned}\text{ELBO}(q) &= \mathbb{E}[\log p(\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta})] + \mathbb{E}[\log p(\boldsymbol{x}|\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta})] \\ &\quad - \mathbb{E}[\log q(\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta})] \\ &= \mathbb{E}[\log p(\boldsymbol{x}|\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta})] - \mathbb{KL}(q(\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta})||p(\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta})). \end{aligned} \quad (11)$$

The first term, $\mathbb{E}[\log p(\boldsymbol{x}|\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta})]$, is the expected log-likelihood of the data. It encourages the variational distribution to favor parameter values that best explain the observed data. The second term, $\mathbb{KL}(q(\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta})||p(\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta}))$, is the KL divergence between two priors $p(\boldsymbol{v})$ and $q(\boldsymbol{v})$, which forces the variational distribution to stay close to the prior. Therefore, we can interpret the optimization of the ELBO as finding a solution to sufficiently explain the observed data with minimal deviation from the prior.

For the DPMM model, based on the idea of variational inference, we define the variational distribution q following the mean-field assumption, where each latent variable has its variational factor and is mutually independent of each other [19]:

$$q(\boldsymbol{v}, \boldsymbol{\theta}, \boldsymbol{\beta}) = \prod_{n=1}^{N} q(v_n|\hat{r}_n) \prod_{k=1}^{K} q(\beta_k|\hat{\alpha}_{k_1}, \hat{\alpha}_{k_0}) q(\theta_k|\hat{\lambda}_k), \quad (12)$$

$$\begin{aligned} q_{v_n} &= \text{Cat}(v_n|\hat{r}_{n_1:n_K}), \\ q_{\beta_k} &= \mathcal{B}(\beta_k|\hat{\alpha}_{k_1}, \hat{\alpha}_{k_0}), \\ q_{\theta_k} &= H(\theta_k|\hat{\lambda}_k). \end{aligned} \quad (13)$$

We mark variables with hats to distinguish parameters of variational factors q from parameters of the generative model p. The categorical factor q_{v_n} is limited to only K components to make the computation possible. Since the variational distribution is merely approximate, the true posterior remains infinite. If K is large, the variational distribution can be reasonably close to the infinite posterior through ELBO optimization.

We consider a special case where both H and F in (6) belong to the exponential family:

$$p_H(\theta_k|\lambda_0) = \mathbb{E}[\lambda_0^T t_0(\theta_k) - a_0(\lambda_0)], \quad (14)$$

$$p(x_n|\theta_k) = \mathbb{E}[\theta_k^T t(x_n) - a(\theta_k)]. \quad (15)$$

Hughes and Sudderth [19] showed that in this case, we can express the ELBO in terms of the expected mass $\hat{N}_k$ and the expected sufficient statistic $s_k(x)$ of each component k:

$$\begin{aligned}\text{ELBO}(q) = \sum_{k=1}^{K} \Bigg(& \mathbb{E}_q[\theta_k]^T s_k(x) - \hat{N}_k[a(\theta_k)] + \hat{N}_k[\log \pi_k(\beta)] \\ & - \sum_{n=1}^{N} \hat{r}_{nk} \log \hat{r}_{nk} \\ & + \mathbb{E}_q\left[\log \frac{\mathcal{B}(\beta_k|1, \alpha)}{q(\beta_k|\hat{\alpha}_{k_1}, \hat{\alpha}_{k_0})}\right] + \mathbb{E}_q\left[\log \frac{H(\theta_k|\lambda)}{q(\theta_k|\hat{\lambda}_k)}\right]\Bigg). \end{aligned} \quad (16)$$

Then each variational factor can be updated individually in an iterative manner. First, we update the *local* variational parameters in the categorical factor q_{v_n} for each cluster assignment:

$$\tilde{r}_{nk} = \exp(\mathbb{E}_q[\log \pi_k(\beta)] + \mathbb{E}_q[\log p(x_n|\theta_k)]), \quad (17)$$

$$\hat{r}_{nk} = \frac{\tilde{r}_{nk}}{\sum_{l=1}^{K} \hat{r}_{nl}}. \quad (18)$$

Next, we update the *global* parameters in the factors for the stick-breaking proportions q_{β_k} and the factors for the base distribution q_{θ_k}:

$$\hat{N}_k = \sum_{n=1}^{N} \hat{r}_{nk}, \quad \alpha_{k1} = \alpha_1 + \hat{N}_k, \quad \alpha_{k0} = \alpha_0 + \sum_{l=k+1}^{N} \hat{N}_l,$$

$$s_k(x) = \sum_{n=1}^{N} \hat{r}_{nk} t(x_n), \quad \hat{\lambda}_k = \lambda_0 + s_k(x). \quad (19)$$

The calculation of the summary statistics N_k and $s_k(x)$ requires the full dataset. For inference on a large dataset, we need a batch-based approach such as stochastic variational Bayes (soVB), which is based on stochastic gradient updates, and memoized online variational Bayes (memoVB) [19], which breaks the summary statistics of the entire dataset into a sum of the summary statistics of each batch.

The nonparametric nature of DPMM allows the model to adapt to the changing number of clusters. Therefore, we can develop additional heuristics to dynamically insert new clusters and remove redundant clusters. This is particularly helpful in escaping a local optimum when using batch-based variational inference approaches.

In memoVB, a birth move to create new clusters is made by first collecting subsamples x' that are poorly described by a single cluster when passing through each batch. After passing through all the batches, a separate DPMM model is fitted on x' with K' initial clusters. Assuming that the active number of clusters before the birth move is K, we can either accept or reject the new cluster proposals by comparing the result of assigning x' to $K + K'$ with that of assigning x' to K. To complement the birth move, a merge move potentially combines a pair of clusters into one. Two clusters are merged if the merge improves the ELBO objective, leaving $K - 1$ clusters after the merge. The birth move helps to escape the local maximum in ELBO optimization, which can occur in gradient-based soVB. The birth and merge moves control the number of clusters in this infinite model while maintaining a non-decreasing ELBO.

V. Simultaneous Representation Learning and Clustering

We introduce our methods in two parts. In this section, we introduce DPMM-VAE, a deep clustering framework that combines Bayesian nonparametric models and a variational autoencoder. In the next section, we focus on our meta-RL algorithm MELTS, where we adapt our DPMM-VAE framework as part of the task self-discovery module.

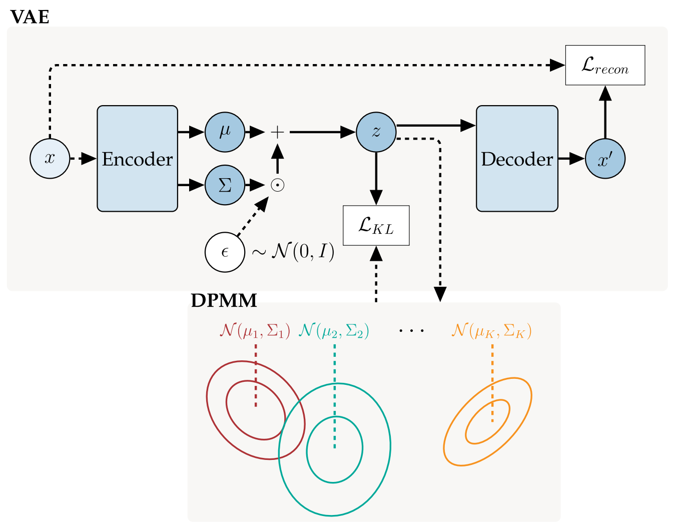

VAE
$\mathcal{L}_{recon}$
x
Encoder
μ
Σ
$+$
$\odot$
z
Decoder
x'
ϵ $\sim \mathcal{N}(0, I)$
$\mathcal{L}_{KL}$
DPMM
$\mathcal{N}(\mu_1, \Sigma_1)$
$\mathcal{N}(\mu_2, \Sigma_2)$
$\cdots$
$\mathcal{N}(\mu_K, \Sigma_K)$

Fig. 3. Overview of the DPMM-VAE. We optimize the DPMM and the VAE alternately. When updating the DPMM, we use the current latent sample z obtained from the VAE. When updating the VAE, we assign the outputs of the encoder to the clusters of the current DPMM and minimize the KL divergence with respect to the assigned cluster.

Given an unlabeled set of data with n points $\{x_i\}_{i=1}^n \in X$, we design a deep clustering method, DPMM-VAE, that simultaneously learns: (1) The latent representation z_i of each x_i in a latent space Z and the corresponding mapping from the data space X to Z, namely $f_\theta : X \to Z$ (the dimension of Z is a fixed hyperparameter D). (2) The number of clusters K and the distribution on each cluster. (Since we assume Gaussian distribution, we learn the mean μ_k and covariance matrix Σ_k for each cluster k.) (3) The cluster assignment v_i of each x_i, where $v_i \in \{1, \ldots, K\}$.

DPMM-VAE combines a standard variational auto-encoder (VAE) for learning representations and a Dirichlet process mixture model for learning the total number of clusters and the cluster distributions. The cluster assignments depend jointly on the learned representation and the learned distribution of each cluster. Our algorithm uses an alternative optimization scheme: First, we update the DPMM model using the latent variables z_i, which are sampled from the encoder using the reparametrization trick during the last VAE update. Then we fix the parameters of the DPMM model and update the parameters of the VAE. Here we use the DPMM model to assign each z_i to a previously learned cluster and minimize the KL divergence with respect to the assigned cluster. An overview of our proposed method is illustrated in Fig. 3.

A. Updating the DPMM Model

When updating the parameters of the DPMM model, we fit a DPMM model on the latent samples z_i's obtained from the VAE. We define the generative process of assigning data points to clusters according to the stick-breaking process. We assume a generative process as follows:

1) The mean μ_k and covariance matrix Σ_k of each cluster k are drawn from a normal-Wishart distribution. We assume a normal-Wishart prior because normal-Wishart distribution is the conjugate prior for a diagonal Gaussian with unknown means and unknown covariance matrix. We first sample Σ_k from the Wishart distribution with scale matrix $\mathbf{W}$ and degree of freedom ν. Then we sample μ_k from a multivariate normal distribution with mean μ_0 and variance $(\lambda\Sigma_k)^{-1}$:

$$\Sigma_k \sim \mathcal{W}(\mathbf{W}, \nu), \tag{20}$$

$$\mu_k \sim \mathcal{N}(\mu_0, (\lambda\Sigma_k)^{-1}). \tag{21}$$

For simplicity, we assume Σ_k is a diagonal covariance matrix, assuming the latent space is D-dimensional.

$$\mu_k = \begin{pmatrix} \mu_{k1} \\ \mu_{k2} \\ \vdots \\ \mu_{kD} \end{pmatrix}, \Sigma_k = \begin{pmatrix} \lambda_{k1}^{-1} & & & \\ & \lambda_{k2}^{-1} & & \\ & & \ddots & \\ & & & \lambda_{kD}^{-1} \end{pmatrix}. \tag{22}$$

2) The probabilities of each cluster are drawn from a GEM distribution with concentration parameter α. In practice, we view it as a stick-breaking process where we first sample $\beta_i \sim \mathcal{B}(1, \alpha)$ and then $\pi_k = \beta_i \prod_{i<k}(1 - \beta_i)$.

$$\pi \sim \text{GEM}(\alpha). \tag{23}$$

3) The cluster assignment v_n is drawn from a discrete distribution $\text{Cat}(\pi)$ based on the cluster probabilities we previously obtained.

$$v_n \sim \text{Cat}(\pi). \tag{24}$$

4) In our DPMM model, when the latent variable z_i is assigned to a cluster $v_n = k$, we assume that it is generated from a multivariate Gaussian with mean μ_k and variance Σ_k.

$$z_i | v_n = k \sim \mathcal{N}(\mu_k, \Sigma_k). \tag{25}$$

5) Finally, the original data are assumed to be generated by the decoder, namely $f_\theta(z_i) = x_i$, where θ are the parameters of the decoder neural network.

Given the overall generative process, the DPMM has a joint probability:

$$\begin{aligned} p(\boldsymbol{z}, \boldsymbol{v}, \boldsymbol{\beta}, \boldsymbol{\mu}, \Sigma) = & \prod_{i=1}^{N} \mathcal{N}(z_i | \mu_{v_n}, \Sigma_{v_n}) \text{Cat}(v_n | \boldsymbol{\pi}(\boldsymbol{\beta})) \\ & \prod_{k=1}^{\infty} \mathcal{B}(\beta_k | 1, \alpha) \\ & \prod_{k=1}^{\infty} \mathcal{N}(\mu_k | \mu_0, (\lambda\Sigma_k)^{-1}) \mathcal{W}(\Sigma_k | \mathbf{W}, \nu). \end{aligned} \tag{26}$$

We then use variational inference to find the posterior estimates for the parameters. We construct the variational distribution q

Algorithm 1: DPMM-VAE.

```
Require: Dataset $\mathcal{D}$, batch size $B$, DPMM optimization steps $T$
Ensure: Learned parameters $\eta$ of the encoder and $\phi$ of the decoder, total number of clusters $K$, learned parameters of each
 DPMM cluster $\pi_{1:K}$, $\mu_{1:K}$, $\sigma_{1:K}$
1:  Initialize parameters $\eta$ and $\phi$ for the VAE
2:  Initialize the DPMM with $K = 1$, including initializing the parameters $(\mu_0, \lambda, \mathcal{W}, \nu)$ for the NW-distribution and the
    parameter $\alpha$ for the Beta-distribution
3:  repeat
4:  Sample mini-batch of data $\mathcal{M} = \{x_0, \ldots, x_B\} \sim \mathcal{D}$
5:    With the VAE, obtain latent variables $\{z_0, \ldots, z_B\}$ and save to a buffer $\mathcal{Z} = \mathcal{Z} \bigcup \{z_0, \ldots, z_B\}$
6:    With the current DPMM model, obtain the cluster assignments $\{v_0, \ldots, v_B\}$ and the corresponding assignment
      probabilities for each latent variable.
7:    Compute $\mathcal{L}_{KL}$ using (28) for hard assignment or (29) for soft assignment
8:    Compute $\mathcal{L}_{recon}$ and update $\eta$ and $\phi$ with gradients of the ELBO according to the training of standard VAE
9:    if it is time to update the DPMM (e.g., at the end of an epoch) then
10:     for step $i = 1, 2, \ldots, T$ do
11:         Use data in $\mathcal{Z}$ as input and the current DPMM model as prior
12:           Update random variables of the DPMM, $\hat{\mu}_0, \hat{\lambda}, \hat{\mathcal{W}}, \hat{\nu}, \hat{r}_{n_1:n_k}, \hat{\alpha}_{k_1}, \hat{\alpha}_{k_0}$
13:           Attempt birth moves to potentially add new clusters
14:           Attempt merge moves to potentially combine pairs of clusters
15:       Reset buffer $\mathcal{Z} = \emptyset$
16:   until convergence
```

with the following factorization:

$$\begin{aligned} q(\boldsymbol{v}, \boldsymbol{\beta}, \boldsymbol{\mu}, \Sigma) &= \prod_{n=1}^{N} q(v_n) \prod_{k=1}^{K} q(\beta_k) q(\mu_k, \Sigma_k) \\ &= \prod_{n=1}^{N} \text{Cat}(v_n | \hat{r}_{n_1}, \ldots, \hat{r}_{n_k}) \prod_{k=1}^{K} \mathcal{B}(\beta_k | \hat{\alpha}_{k_1}, \hat{\alpha}_{k_0}) \\ &\quad \prod_{k=1}^{K} \mathcal{N}(\mu_k | \hat{\mu}_0, (\hat{\lambda}\Sigma_k)^{-1}) \mathcal{W}(\Sigma_k | \hat{\mathcal{W}}, \hat{\nu}). \end{aligned} \quad (27)$$

In each optimization step, we update the parameters $\hat{\mu}_0$, $\hat{\lambda}$, $\hat{\mathcal{W}}$, $\hat{\nu}$ of the normal-Wishart distribution. We then use the memoized online variational Bayes method described in Section IV-D to update the parameters for the stick-breaking process, $\hat{r}_{n_1:n_k}$, $\hat{\alpha}_{k_1}$, and $\hat{\alpha}_{k_0}$, to estimate the cluster probabilities and the cluster assignments. Finally, throughout the DPMM optimization steps, we also dynamically adjust the total number of clusters K by adding birth-and-merge moves described in Section IV-D. A complete update for our DPMM model thus breaks down to updating three sets of parameters.

We perform the updates on the DPMM model after each training epoch of the VAE. Because we alternate between updating the DPMM model and the VAE, the DPMM model is not required to converge in each update, nor do we need to fit a new DPMM model from scratch every time. In each update, we initialize the DPMM model with the parameters learned from the previous updates and apply this model to new latent samples produced by the updated VAE. This makes sure that we keep updating the same DPMM model while incorporating the latest changes in the latent space mappings.

B. Updating the VAE

When training the VAE, we jointly minimize the reconstruction loss $\mathcal{L}_{recon}$ and the KL divergence loss $\mathcal{L}_{\text{KL}}$. The reconstruction loss $\mathcal{L}_{recon}$ is the mean squared error between the observed data x and the decoder's output. We keep the reconstruction loss from the standard VAE. The $\mathcal{L}_{\text{KL}}$ is simply the KL-divergence between two isotropic Gaussian distributions. Detailed explanations can be found at [35].

1) Hard Assignment: For the calculation of $\mathcal{L}_{\text{KL}}$, we first obtain the cluster assignment $v_i = k$ of each latent sample z_i using the current DPMM model. According to the DPMM, the mean and covariance matrix for assigned cluster k is μ_k and Σ_k. On the other hand, according to the VAE, $z_{id} = \mu_d(\boldsymbol{x}; \boldsymbol{\eta}) + \sigma_d(\boldsymbol{x}; \boldsymbol{\eta})\epsilon$ for $d \in \{1, 2, \ldots, D\}$, where $\mu(x_i; \boldsymbol{\eta})$ and $\sigma(x_i; \boldsymbol{\eta})$ are the outputs of the VAE's encoder and $\epsilon \sim \mathcal{N}(0, 1)$. The KL divergence between two multivariate Gaussian distributions is calculated as follows:

$$\begin{aligned} D_{\text{KL}_{ik}} = \frac{1}{2} \Bigg[&\log \frac{|\Sigma_k|}{|\Sigma(x_i; \boldsymbol{\eta})|} - D + \text{tr}\{\Sigma_k^{-1} \Sigma(x_i; \boldsymbol{\eta})\} \\ &+ (\mu_k - \mu(x_i; \boldsymbol{\eta}))^T \Sigma_k^{-1} (\mu_k - \mu(x_i; \boldsymbol{\eta})) \Bigg]. \end{aligned} \quad (28)$$

We name this method of assigning a data point to the most probable cluster *hard assignment* because it only involves one single cluster in the KL calculation.

2) Soft Assignment: The cluster assignment made by the hard assignment might not always be correct, and the wrongly-assigned samples will introduce errors into the training by calculating the KL divergence with respect to the wrong cluster. To eliminate the harm of prematurely assigning a sample to a definite cluster, we may additionally take into account the probabilities of the cluster assignments. From the DPMM model,

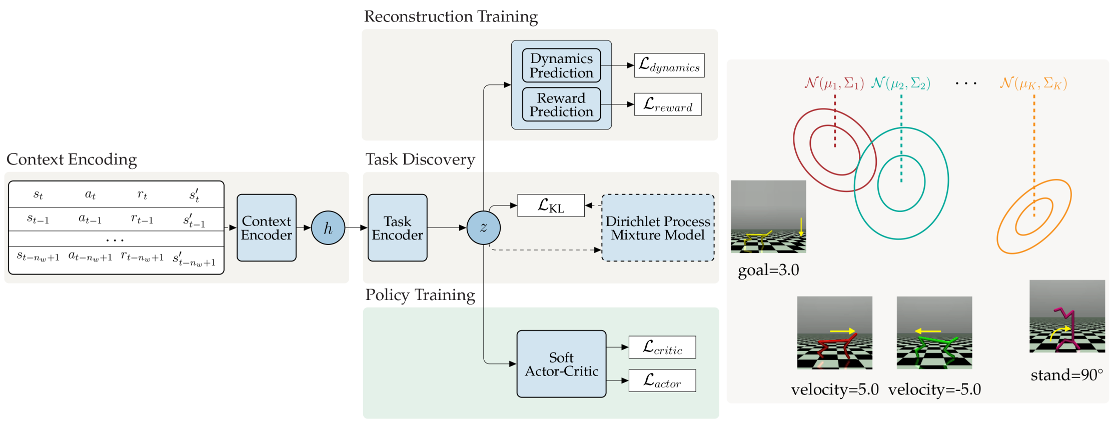


Fig. 4. Overview of our meta-RL algorithm MELTS, which includes four components, namely, the context encoding, the task discovery, the reconstruction training, and the policy training.

we compute the probability p_{ik} of assigning the latent sample z_i to cluster k for all possible $k \in \{1, 2, \ldots, K\}$. Then we define the KL divergence as a weighted sum of the KL divergences calculated with respect to each cluster:

$$D_{\mathrm{KL}_i} = \sum_{k=1}^{K} p_{ik} D_{\mathrm{KL}_{ik}}. \tag{29}$$

Although one can use a more complex weighting strategy, we found this simple weighting by probabilities to be sufficient empirically. We summarize our algorithm for DPMM-VAE in Algorithm 1.

VI. Meta-RL

In this section, we introduce our meta-RL algorithm, MELTS. It decouples the learning of latent task encoding and the learning of a universal policy conditioned on the inferred task. The model for latent task encoding can incorporate the DPMM-VAE we proposed in Section V to get a more structured latent space, or a standard isotropic Gaussian VAE for comparison. The DPMM-VAE-based task encoding allows the agent to adaptively discover nonparametric tasks in a self-guided way. Note that, for robotic tasks the mixtures in the Dirichlet process mixiture model are used to model the non-parametric base tasks and each component is used to model the parametric sub-tasks conditioned on its base task category.

A. Overview

MELTS is a context-based off-policy meta-RL algorithm. It works in both stationary and non-stationary environments and can solve task distributions with both parametric and nonparametric variations. Fig. 4 shows a graphic representation of the algorithm. We decompose our algorithm into four modules: context encoding, task self-discovery, reconstruction training, and policy training. The first three modules together form our task inference network. We will discuss each module in detail in the following sections, but first, we provide an overview of the meta-training and meta-testing procedure as follows. Note that the training tasks $D_{\mathcal{T}_{train}}$ and the testing tasks $D_{\mathcal{T}_{test}}$ are sampled from the task distribution $p(\mathcal{T})$ with no overlaps.

1) Meta-Training: During a meta-training epoch, we first randomly sample a subset of training tasks from $D_{\mathcal{T}_{train}}$ and roll out each task for several episodes to collect replay experiences. We maintain the current context from one episode and obtain a task encoding from the context encoder on a per-time-step basis while rolling out the task. The policy during roll-outs is conditioned on the task encoding and the collected episodes are appended to the replay buffer.

Second, we update the task inference network, which consists of the context encoding module, the task self-discovery module, and the reconstruction training module.

- For each iteration in the training epoch, we sample a set of contexts from the replay buffer according to a sampling strategy S_c (see Section VI-B2). These samples are fed to an encoder based on a gated recurrent unit (GRU) network, and we obtain the task encoding z via posterior sampling.
- An information bottleneck constrains the task encoding to stay close to a (learned) prior. We have developed two variants for the prior (see Section VI-C). (1) In our vanilla VAE setting, the prior is the standard Gaussian $\mathcal{N}(0, I)$. (2) In the DPMM-VAE setting, the encoding is assigned to a cluster by the DPMM model from the last epoch, and the prior is the distribution of the cluster.
- We use a dynamics prediction network and a reward prediction network as the decoder of the task inference module (see Section VI-D). The two networks reconstruct the next state s_{t+1} and the reward r_t from the state, action, and inferred task encoding (s_t, a_t, z_t) of one time step. The reconstruction loss encourages the encoding to capture the most informative information about the current task.

We update the DPMM model at the end of the training epoch. The input data to the DPMM are the task encodings obtained from the last training step of the task inference network, and the rules to update the DPMM are the same as in Section V-B. In policy training, we sample from the replay buffer again to train the agent using the soft actor-critic (SAC) algorithm [36]. We

Algorithm 2: Meta-Reinforcement Learning With Task Self-Discovery (MELTS).

```
Require: Training and test tasks $\mathcal{D}_{\mathcal{T}_{train}}$ and $\mathcal{D}_{\mathcal{T}_{test}} \sim p(\mathcal{T})$
1:  Initialize encoder $q_\eta$, decoder $p_\phi$, policy $\pi_\nu$, Q networks $Q_{1_w}, Q_{2_w}$, target Q networks $Q^T_{1_w}, Q^T_{2_w}$ and replay buffer $\mathcal{B} = \emptyset$
2:  repeat
3:    Roll out randomly sampled tasks from $\mathcal{D}_{\mathcal{T}_{train}}$ using the policy $\pi_\nu$ and append episodes into $\mathcal{B}$
4:    for each task-inference training step do
5:        Sample mini-batch $\mathcal{M}_1 = \{\mathcal{C}_1, \mathcal{C}_2, \cdots \mathcal{C}_{M_1}\} \sim B$ with sampling strategy $S_c$
6:        Obtain context encodings $h(C_i)\ \forall i \in \{1, 2, \ldots, M_1\}$          ▷Section VI-B
7:        Obtain task latent samples $z_i \sim q_\eta(z_i|h(\mathcal{C}_i))\ \forall i \in \{1, 2, \ldots, M_1\}$
8:        Compute $\mathcal{L}_{KL}$ according to the type of the VAE          ▷Section VI-C
9:        Compute $\mathcal{L}_{recon}$ with (37)          ▷Section VI-D
10:        Update $\theta, \phi$ with gradients of the ELBO
11:   if Use DPMM-VAE for task self-discovery then
12:        Fit DPMM on $z_i$'s from the last task-inference training step
13:        Update the DPMM model          ▷Section V-B
14:   for each policy training step do
15:        Randomly sample mini-batch $\mathcal{M}_2 \sim \mathcal{B}$
16:        Infer task encoding $z_i = q_\eta(z_i|h(\mathcal{C}_i))\ \forall i \in \{1, 2, \ldots, M_2\}$
17:        Concatenate $z_i$ to the SAC inputs and update $\pi_\nu, Q_{1_w}, Q_{2_w}, Q^T_{1_w}, Q^T_{2_w}$
18:  until SAC convergence
```

obtain the task encoding from the task inference network online and use it as an auxiliary input to the SAC.

2) Meta-Testing: We evaluate each task in $D_{\mathcal{T}_{test}}$ with zero-shot adaption, which means we infer a new task encoding at each time step when rolling out the task-conditioned policy. We summarize our algorithm MELTS in Algorithm 2.

B. Context Encoding

In context-based meta-RL, the context refers to the raw experience collected from the roll-out, which contains rich information about the task being executed. We denote the experience obtained at a single transition as a tuple $c_t = (s_t, a_t, r_t, s_{t+1})$. For a given time step $t < N$ from a trajectory $\mathcal{T}_{0:N}$, we define the context as the experiences we collected over a time window of the n_w most recent transitions ($n_w < N$) and denote it as

$$\mathcal{C}_t = (c_{t-n_w+1}, c_{t-n_w+2}, \ldots, c_{t-1}, c_t). \quad (30)$$

We use a single replay buffer for MELTS to access all the past contexts in an off-policy training. We append the newly obtained trajectory to the end of the replay buffer when we execute a training task. To get a training sample during meta-training, we sample an index t from the replay buffer and retrieve the context $\mathcal{C}_t$. It can be the case that, if t is sampled at the beginning of a task trajectory, there are fewer transitions than n_w for the sampled task, and $\mathcal{C}_i$ contains some transitions from the previous task. In practice, we choose n_w to be much smaller than the length of the task trajectory N so that noisy samples occur only occasionally. Another reason for keeping n_w small is to ensure the zero-shot adaptation capability of the algorithm. Tasks can switch at any time step in a non-stationary environment, so ideally, n_w should be smaller than the switching interval. Therefore, we push our algorithm to learn to infer the task from a minimal amount of past experiences.

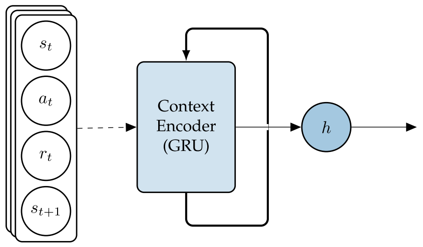


Fig. 5. Context encoding. We pass $c_t = (s_t, a_t, r_t, s_{t+1})$ sequentially to a GRU-based context encoder. The context encoding of the whole context $\mathcal{C}_t = (c_{t-n_w+1}, \ldots, c_t)$ is the final hidden state h after passing the step t.

1) Feature Extraction With GRU: Our context is high-dimensional, so it can be difficult to directly encode the contexts into latent task representations. We want to provide the subsequent task encoder with more succinct features. Thus, as visualized in Fig. 5, we train a GRU network to extract features from the context. Since contexts are samples from sequential interactions between the agent and the environment, the GRU can learn the correlation among elements within the sequence. After we pass the whole context step-by-step up to c_t, we take the final hidden state as the input for the task encoder. Our decoder network design differs from that of RL^2 [3] or VariBAD [6] in terms of handling variable-length contexts. In [3] and [6], the context consists of the full history up to the current time step, and the hidden state is not reset between episodes. However, this setup has limited adaptability in facing non-stationary scenarios. Here we hypothesize that enough information about the task can be learned from just a few time steps. This is why we use GRU over LSTM, as GRU operates directly on the hidden state and does not require an additional internal memory unit as in LSTM. Our proposed feature extraction method only use few recent transitions for context encoding. This departure from stringent stationary assumptions enhances its adaptability in addressing non-stationary scenarios. Furthermore, it enables fast task discovery and performance convergence even in zero-shot trials.

2) Sampling With Priority: We sample from the same replay buffer for training the context encoding and training the policy. However, we use different sampling strategies. For policy training, we sample uniformly from the entire replay buffer. The transitions in the replay buffer are highly-correlated. A mini-batch of random samples alleviates the problems with non-i.i.d data and improves the stability of the policy training. On the other hand, for context encoding, we sample with a linearly weighted sampling strategy S_c. According to S_c, when the replay buffer has a total of T transitions, the probability of sampling the ith transition is:

$$w_i = \frac{i}{\sum_{t=0}^{T} t} = \frac{2i}{T(T+1)}. \tag{31}$$

The sampling strategy is important because of the difference between training a VAE in the usual unsupervised learning setting or the RL setting. In unsupervised learning, the training data at each epoch is the whole training dataset. In the RL setting, although the task inference network resembles the structure of a VAE, the training data at each epoch are different samples from the replay buffer. Data in the replay buffer are built cumulatively with a policy that is updated constantly during training. The VAE will be insufficiently trained on the most recent samples if we use a uniform sample strategy, which will also harm the stability of the task encoding inference during adaptation.

C. Task Self-Discovery

Similar to other context-based meta-RL algorithms, the policy in MELTS is conditioned on the task encoding z, where z is unknown and has to be learned through the agent-environment interactions. Formally, we define a generative process where we assume each transition $c_t = (s_t, a_t, r_t, s_{t+1})$ is generated from a latent variable z_t with a likelihood of $p(c_t|z_t)$.

As in a standard VAE, the nonlinearity of neural networks means that the exact inference on the posterior $p(z_t|c_t)$ is intractable. Therefore, we use a variational distribution $q(z_t|c_t)$ as the approximate posterior. $q(z_t|c_t)$ is parameterized by a neutral network (task encoder), which we train on the output of the context encoder. The context encoder takes a sequence of transitions $\mathcal{C}_t = (c_{t-n_w+1}, c_{t-n_w+2}, \dots, c_{t-1}, c_t)$ and outputs a single hidden state $h(\mathcal{C}_t)$. When we feed $h(\mathcal{C}_t)$ to the task encoder, we get a single latent variable z_t. Since we make the implicit assumption that all transitions in $\mathcal{C}_t$ are generated from the same task, we can also assume that z applies to all transitions, namely $z = z_{t-n_w+1} = z_{t-n_w+2} = \dots = z_{t-1} = z_t$. The likelihood $p(c_t|z_t)$ is also parameterized by the decoder network. We can then express the evidence lower bound (ELBO) as:

$$\begin{aligned}\mathcal{L}(c_t, \boldsymbol{\eta}) &= \mathbb{E}_{z_t \sim q_{\boldsymbol{\eta}}(z_t|c_t)}[\log p_{\boldsymbol{\phi}}(c_t|z_t)] - \mathbb{KL}(q_{\boldsymbol{\eta}}(z_t|c_t)||p(z_t)) \\ &\leq p(c_t), \end{aligned} \tag{32}$$

where $\boldsymbol{\eta}$ are the parameters of the encoder network (including both the context encoder parameters $\boldsymbol{\eta}_{context}$ and the task

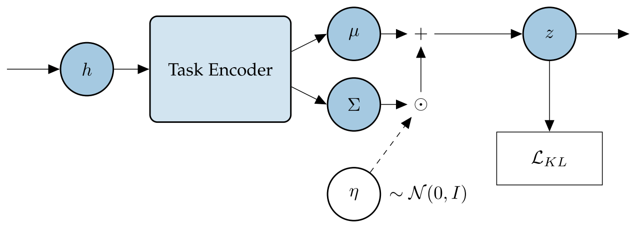


Fig. 6. Task self-discovery (Gaussian-VAE). The context encoding $h = h(\mathcal{C}_t)$ is fed into a task encoder to obtain the single task encoding z, representing the task for all time steps in $\mathcal{C}_t$. The training of the task encoder and the calculation of $\mathcal{L}_{\text{KL}}$ resemble a standard VAE.

encoder parameters $\boldsymbol{\eta}_{task}$), and $\boldsymbol{\phi}$ are the parameters of the decoder network. Incorporating the fact that all transitions from the trajectory $\mathcal{T}_{0:N}$ of a single task should share the same task encoding and taking into account that the encoder network's input is the hidden state $h(\mathcal{C}_t)$, we can optimize the whole task inference network by maximizing:

$$\begin{aligned}\mathcal{L}(\mathcal{C}_t, \boldsymbol{\eta}) = & \underbrace{\sum_{t=0}^{N-1} \mathbb{E}_{z_t \sim q_{\boldsymbol{\eta}}(z_t|h_{\boldsymbol{\eta}}(\mathcal{C}_t))}[\log p_{\boldsymbol{\phi}}(c_t|z_t)]}_{\mathcal{L}_{recon}} \\ & - \beta_{\text{KL}} \underbrace{\mathbb{KL}(q_{\boldsymbol{\eta}}(z_t|h_{\boldsymbol{\eta}}(\mathcal{C}_t))||p(z_t))}_{\mathcal{L}_{\text{KL}}}, \end{aligned} \tag{33}$$

where β_{KL} controls the degree of regularization by the KL divergence. We focus on $\mathcal{L}_{\text{KL}}$ in this section and discuss $\mathcal{L}_{recon}$ in Section VI-D.

We regard task self-discovery as the model's ability to correctly capture the whole structure of the task distribution, including both parametric and nonparametric variations. For a task distribution containing both types of variation, we name the qualitatively distinctive tasks with nonparametric variation "base tasks". The tasks within a base task that have only parametric variations are called "subtasks".

The prior $p(z_t)$ in $\mathcal{L}_{\text{KL}}$ expresses our beliefs about the task distribution before seeing any data. In this study, we design alternative priors to better capture the complex task distributions in many real-world meta-RL settings. Alternative priors have also been proposed to model the latent task space in context-based meta-RL [7], [24]. However, our work is the first that can discover the number of base tasks dynamically based on the data and aims to solve the non-parametric task environment.

To explore the influence of different priors on task self-discovery ability, we propose two compositions for the task encoder and $\mathcal{L}_{\text{KL}}$:

1) Gaussian-VAE, which is the standard VAE with a fixed $\mathcal{N}(0, I)$ prior.
2) DPMM-VAE, which we adapt from our own deep clustering framework proposed in Section V.

1) With Fixed Gaussian Prior: Our first configuration is the uninformative and fixed standard Gaussian prior $\mathcal{N}(0, I)$. Fig. 6 depicts this configuration. Since the context encoder has already extracted the features, our task encoder has only one MLP layer and outputs the mean μ and variance Σ of the posterior

Gaussian. Our hypothesis is that with Gaussian-VAE, we can capture the parametric variation of tasks but may not capture the nonparametric variation very well due to the lack of clustering capability in the Gaussian-VAE.

2) With the DPMM Model: Our final composition with the DPMM-VAE adds a parametrized distribution for each cluster on top of the stick-breaking process. Our DPMM-VAE uses a distributional prior, meaning that we can just replace the prior $p(z)$ and keep the architecture of the standard VAE. In addition, with the *hard assignment* approach, we obtain a configuration where data belonging to different clusters can occupy truly disjoint subspaces in the latent space, which matches our belief in a disjoint base task distribution.

In contrast to the previous two compositions using a fixed prior to compare the observations with our initial knowledge, the prior in DPMM-VAE is learned from the data itself. The prior for the latent $q(z|h(\mathcal{C}_t))$ is one of the clusters of DPMM, which is essentially an infinite mixture of Gaussians. We update the mixture regularly during training based on the posterior z obtained from the previous optimization step, including adjusting the number of clusters, the distribution of the individual cluster, and the cluster assignments. In the meta-RL setting, we can interpret the generative process of the DPMM-VAE in Section V in a new way to match the structure of the task distribution, where the cluster assignment represents the assignment to the distinct base task. The task encoding is sampled from the distribution of subtasks of the assigned base task.

$$v \sim p(v) = \text{Cat}(\pi), \qquad \text{base task distribution} \tag{34}$$

$$z \sim p(z|v) = \mathcal{N}(\mu(v), \Sigma(v)). \qquad \text{sub-task distribution} \tag{35}$$

We hypothesize that with the DPMM-VAE, we can capture both the parametric variation and the nonparametric variation in the task distribution. The clustering controlled by a stick-breaking process can capture the nonparametric variation. In addition, each cluster is parametrized by an individual Gaussian distribution, implying that we can also capture the parametric variation within a cluster.

D. Reconstruction Training

We need a way to encourage the task encoding to capture sufficient information so that it can distinguish one task from another. PEARL [5] uses the Bellman error from the SAC to indirectly supervise the learning of the task encoding. Here we decouple the task inference training from the policy training via dedicated decoder networks to reconstruct the task MDP given the task encoding as our recent algorithm [7]. For this reason, we design two neural networks to approximate these functions: One dynamics prediction network with parameters $\phi_{dynamics}$ to predict the next state s_{t+1}, and one reward prediction network with parameters ϕ_{reward} to predict the reward r_t. Both use the state s_t, action a_t, and the sampled task encoding z_t from time step t as inputs. Both networks are implemented with MLP layers, and the complete decoder is illustrated in Fig. 7.

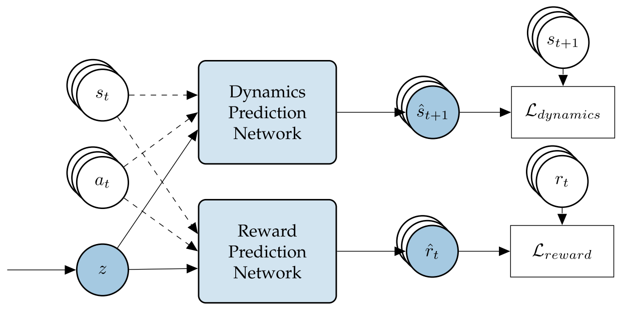


Fig. 7. Reconstruction training. We concatenate the task encoding z to the state s_t and action a_t as input to two decoder networks. The dynamics prediction network predicts the next state s_{t+1}, and the reward prediction network predicts the reward r_t.

Based on the above observation, we can express the log-likelihood of the reconstruction term $\log p(\mathcal{C}|z_t)$ as:

$$\begin{aligned}\log p(\mathcal{C}|z_t) &= \log p((s_{1:N}, r_{1:N})|s_{0:N-1}, a_{0:N-1}; z_t)\\ &= \sum_{t=0}^{N-1} \log p(s_{t+1}|s_t, a_t; z_t)\\ &\quad + \sum_{t=0}^{N-1} \log p(r_t|s_t, a_t; z_t).\end{aligned} \tag{36}$$

The reconstruction loss from the decoder is then:

$$\begin{aligned}\mathcal{L}_{recon} &= \mathcal{L}_{dynamics}(z_t; \phi_{dynamics}) + \mathcal{L}_{reward}(z_t; \phi_{reward})\\ &= \frac{1}{dim(s)} \sum_{t=0}^{N-1} ||s_{t+1} - \log p_{\phi_{dynamics}}(s_{t+1}|s_t, a_t; z_t)||^2\\ &\quad + \frac{1}{dim(r)} \sum_{t=0}^{N-1} ||r_t - \log p_{\phi_{reward}}(r_t|s_t, a_t; z_t)||^2,\end{aligned} \tag{37}$$

where we normalize the loss with the dimension of the state and reward to make the contribution from the two losses relatively equally important.

VII. Experiments

We conduct two sets of experiments in this study. First, we demonstrate the application of the deep clustering framework on the MNIST dataset to highlight its capacity for simultaneous data clustering and deep latent representation learning. Second, we evaluate the performance of MELTS on robotic tasks within both parametric and nonparametric task environments. For further details please refer to our open sourced implementation.[1]

A. Simultaneous Representation Learning and Clustering on MNIST

We evaluate the performance of our DPMM-VAE and its variants in the setting of unsupervised learning. We focus on its performance in three aspects, namely, unsupervised clustering, representation learning and dynamic adaptation.

TABLE I
UNSUPERVISED CLUSTERING ACCURACY (%) ON MNIST TEST SET

Frameworks	Average unsupervised clustering accuracy
Gaussian-VAE	11.34 [±0.01]
GMM K=5	33.11 [±0.01]
GMM K=10	43.27 [±1.54]
GMVAE K=5	46.25 [±2.16]
GMVAE K=10	77.31 [±3.98]
DPMM-VAE	**91.90** [**±0.27**]

1) Experiment Setup: The MNIST dataset of hand-written digits [37] is commonly used for evaluating various types of VAEs [38] and deep clustering algorithms [14], [15]. The dataset contains 50,000 training digits and 10,000 testing digits, with each digit as a 28-by-28 pixel image that is centered and size-normalized. In our approach, we utilize a basic structure comprising 2 convolutional layers as the backbone for the VAE and select the Adam optimizer. To initialize the DPMM model, we assume only a single cluster $K = 1$ initially and set the Beta prior to $\mathcal{B}(1, 5)$ for the stick-breaking proportion.

For comparison, we include one of the state-of-the-art framework GMVAE [39] and we select GMM and VAE with isotropic Gaussian as the baselines. The GMVAE adopts a Gaussian mixture model (GMM) in its prior space and jointly learns clustering and feature representation. However, it requires the number of clusters in the prior space to be predefined before training (in our case, we select K = 5, 10). In contrast, our DPMM-VAE framework dynamically determines the cluster components based on the input data and performs inference in an online manner, eliminating the need for predefining the number of clusters.

2) Experiment Results:

Evaluation of Clustering: The quantitative evaluation of clustering performance is conducted from three perspectives. First, we employ the widely accepted unsupervised clustering accuracy as a metric, which has been used in prior studies such as [40] and [39]. Each framework is trained for 100 epochs, and each trial is repeated with at least five random seeds. The average test accuracy and corresponding 95% confidence intervals are reported in Table I. In this comparison, we assume Gauss-VAE has only one component in its prior space. As shown in Table I, our DPMM-VAE outperforms all baselines on the MNIST test set with an average accuracy exceeding 91%. Notably, it demonstrates competitive performance even when compared to supervised approaches.

Furthermore, we visualize the learned structure of the latent space to further inspect the clustering qualitatively. Fig. 8 shows the latent representations after t-SNE projection [41] for DPMM-VAE and Gaussian-VAE, respectively. Compared with Gaussian-VAE, we observe that for the DPMM-VAE the latent representations of the same class are closer to the cluster centroid, and different cluster centroids are further away from each other. This indicates that our DPMM-VAE has better intra-cluster cohesion and inter-cluster separation capabilities. There is no assumption on the number of clusters in our DPMM-VAE model ($K = 1$ at the start of the training); rather, the model

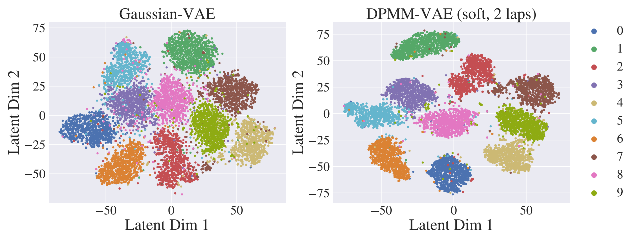


Fig. 8. The t-SNE projections of the latent representations learned by Gaussian-VAE and DPMM-VAE, colored by the input true label.

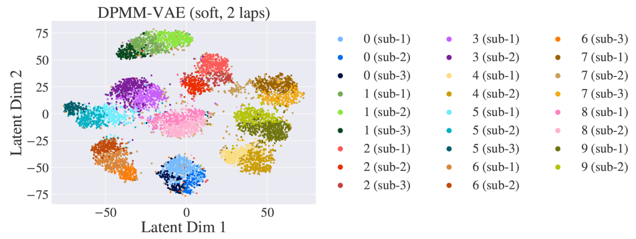


Fig. 9. Latent representations learned by the DPMM-VAE (after T-SNE projection). Data points are colored by the cluster membership assigned internally by the DPMM model. The DPMM model in our DPMM-VAE learns 2 or 3 sub-clusters for each digit. The label "0 (sub -1)" represents the first sub-cluster of the digit "0".

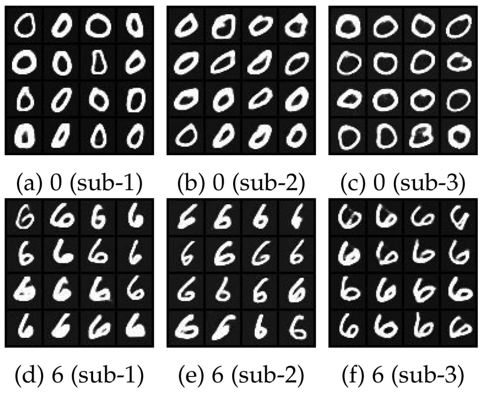


Fig. 10. Selectively generated samples from the individual clusters learned by DPMM-VAE.

learns to best adapt to the data. Hence there is not always a one-to-one mapping between the true labels and the learned DPMM clusters. Fig. 9 shows the t-SNE projection of learned latent samples from DPMM-VAE, colored by clusters.

Evaluation of Representation Learning: The reconstruction quality serves as an indicator of the usefulness of the learned latent representations. To assess this aspect, we utilize the decoder to reconstruct images and evaluate its performance in an intuitive manner. The generation results are illustrated in Fig. 10. Notably, it is evident that each component of DPMM captures one digit and effectively reconstructs clearly clustered images. Furthermore, it is interesting to observe that our DPMM-VAE results exhibit hierarchical clustering on the MNIST data. The same digit is mapped to two or three DPMM clusters, with each DPMM cluster capturing a slightly different style of writing

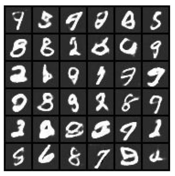


Fig. 11. Samples from the Gaussian-VAE.

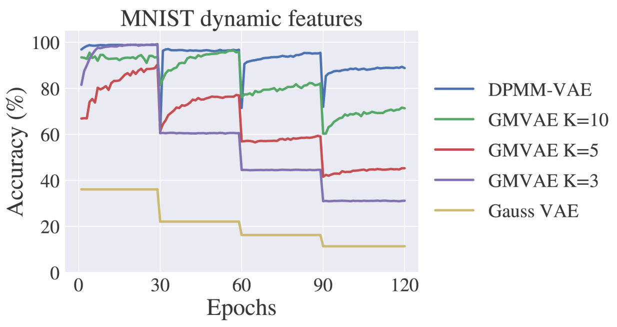


Fig. 12. Test unsupervised clustering accuracy of DPMM-VAE on MNIST with dynamic changing features. We introduce dynamic feature in training phase, where the model was initially trained on 3 digits, and the number of digits is increased to 5, 7, and 10 at epochs 30, 60, and 90.

that digit. For example, Fig. 10(a), (b), and (c) are generative samples from the three learned DPMM clusters corresponding to the digit "0". These clusters appear to capture different tilting and roundness of the digit.

A distinct benefit of the DPMM-VAE is that we can generate samples from each learned cluster. To generate an image with the Gaussian-VAE, we use samples from the Gaussian prior $\mathcal{N}(0, I)$ and feed them to the decoder. This gives us the generated images of all digits, for example, as shown in Fig. 11, whereas generating an image for a specific digit is not possible out of the box. With the DPMM-VAE, we can sample random noises from the learned distribution $\mathcal{N}(\mu_k, \Sigma_k)$ of each DPMM cluster and then use the decoder to obtain generated images for a specific digit. Fig. 10 shows that we can obtain generative samples for any single digit with the DPMM-VAE.

Evaluation of Dynamic Feature Adaptation: Moreover, to demonstrate the superiority of DPMM-VAE over normal Gauss-VAE and other state-of-the-art baselines, we conduct evaluations in a more challenging scenario. In this setting, the frameworks are tasked with classifying data inputs from a dataset with dynamically changing features, particularly with an increasing number of features.

We employed the MNIST dataset to simulate a scenario where dynamic changing features are encountered. Initially, the frameworks are trained with 3 types of digits, and after several epochs, we increase the number of accessible digits to 5, 7, and 10 at epochs 30, 60, and 90, respectively. The models run for a total of 120 epochs, and after each epoch, we evaluate and report their performance on the test dataset. We utilize the unsupervised clustering accuracy as metric for comparison, which has been widely accepted in prior studies [39], [40].

TABLE II
CONFIGURATION FOR THE EIGHT BASE TASKS IN OUR BENCHMARK ENVIRONMENT *HALF-CHEETAH 8-TASK*

Base Task	Target State	Range	Reward Function
Run forward	Horizontal velocity	$1 \le v_x^* \le 5$	$-\lvert v_x^* - v_x \rvert$
Run backward	Horizontal velocity	$-5 \le v_x^* \le -1$	$-\lvert v_x^* - v_x \rvert$
Goal in front	Horizontal position	$5 \le p_x^* \le 25$	$-\lvert p_x^* - p_x \rvert$
Goal in back	Horizontal position	$-25 \le p_x^* \le -5$	$-\lvert p_x^* - p_x \rvert$
Front stand	Angular position	$\pi/6 \le p_\theta^* \le \pi/2$	$-\lvert p_\theta^* - p_\theta \rvert$
Back stand	Angular position	$-\pi/2 \le p_\theta^* \le -\pi/6$	$-\lvert p_\theta^* - p_\theta \rvert$
Jump	Vertical velocity	$1.5 \le v_z^* \le 3$	$-\lvert v_z^* - \lvert v_z \rvert \rvert$
Front flip	Angular velocity	$2\pi \le v_\theta^* \le 4\pi$	$-\lvert v_\theta^* - v_\theta \rvert$

The target states and the reward functions for the tasks are different for the base tasks. The sub-tasks for each base task can be parameterized by the target state

The test result is shown in Fig. 12. As new features are added to the dataset, the DPMM-VAE dynamically "births" new components to accommodate them. This may cause a temporary decrease in accuracy during the early stages of training when both the VAE and DP components are not yet converged. However, once training on the new subset is converged, the accuracy returns to the best possible performance. In contrast, the baseline framework GMVAE lacks this dynamic-adaptive capability in facing non-parametric features. As the number of features exceeds the number of components, the test accuracy of GMVAE displays a stepwise decline. Furthermore, since the Gauss-VAE model can be considered to have only one component in its prior, the standard Gauss-VAE fails in this experimental setup.

B. Meta-RL for Robotics

1) Experiment Setup: The OpenAI Gym [42] toolkit provides a set of benchmarks and functions that helps RL algorithms to communicate with the environment. We use the environments Half-Cheetah-v2, Ant-v2, Hopper-v2 and Walker-v2 in OpenAI Gym to set up a distribution of high-dimensional continuous control tasks for our experiments. In these environments, the agent observes the kinematic properties of the robot in the continuous state space. The action of the agent controls the torques applied to the joints of the robot. These environments are used extensively to evaluate meta-RL and multi-task RL algorithms [3], [5], [6], [7]. We modify the environments by adjusting the reward function to create non-parametric base tasks, such as "run forward", "front flip", or "jump". Additionally, to introduce parametric variations within each base task, we randomize the target state, such as the target velocity for the "run forward" base task, ranging from $1\,m/s$ to $5\,m/s$. This ensures that the agent learns a more generalized task distribution. Fig. 13 shows the eight base tasks of Half-Cheetah environment, the specific configuration for the Half-Cheetah 8-Task environment is provided in Table II, while the details of other utilized environments can be found in our open-sourced implementation. To align with the meta-learning setting, in Half-Cheetah 8-Task we initialize in total 120 variations for these base tasks, which are then split into two groups. During the meta-training phase, the agent has access to 80 train sub-tasks, while during the meta-testing phase, it is tested on other 40 test tasks that were not seen during training.

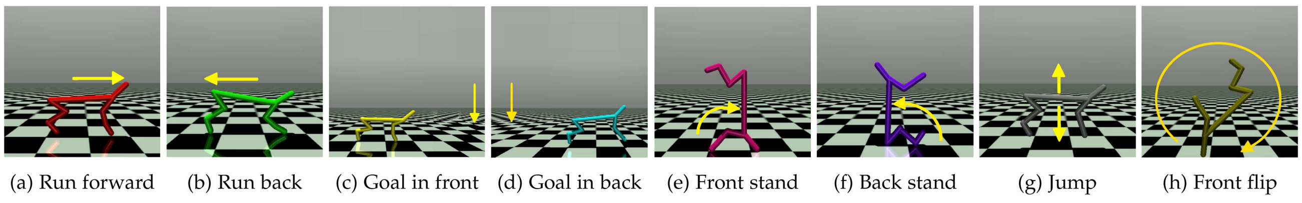


Fig. 13. Eight base tasks in the benchmark environment *Half-Cheetah 8-Task*.

We use three common measures to compare our method with previous meta-RL algorithms.

- *Asymptotic performance:* We measure the asymptotic performance by the average return on the test tasks $\mathcal{D}_{\mathcal{T}test}$.
- *Sample efficiency:* An algorithm is more sample-efficient if it can make better use of the collected experiences and achieve a better result on the learning curve with fewer training samples.
- *Latent representation:* Another important goal of our experiments is to show the capability of task self-discovery by inspecting the structure of the task latent space.

We compare our algorithms with all three types of meta-RL algorithms as outlined in Section II-A: We select RL^2 [3] to represent recurrence-base methods, ProMP [43] to represent gradient-based methods, and PEARL [5] to represent context-based methods. Our framework is developed with PyTorch. We conduct each experiment using at least five random seeds and report their mean value and corresponding 95% confidence interval for comparison, where the random seeds restrict the selection of goal space for each base task and framework initial parameters. Common hyperparameters are used as they had already been well fine-tuned (refer to Appendix Table III, available online). Additionally, we fine-tune the hyperparameters of DPMM on MNIST, and further details are provided in the Appendix Tables IV and V, available online.

2) Experiment Results:

Half-Cheetah 8-Task: We compare the MELTS (DPMM) and MELTS (Gauss) with three baseline frameworks PEARL [5], RL^2 [3], and ProMP [43]. For PEARL, MELTS with DPMM-VAE, and MELTS with Gaussian-VAE, we use the same latent dimension $D = 8$ for a fair comparison. We conducted training for our framework with 2000 epochs. Within each epoch, we initially sample sub-tasks from the task distribution $D_{T_{train}}$ and collect corresponding transitions. Subsequently, we sample batches from the collected replay buffer and perform network updates. Following the completion of each training epoch, we evaluate the performance of our framework on the meta-test task distribution $D_{T_{test}}$. We report the overall average return achieved through zero-shot adaptation for our framework, while for other baselines, we provide performance results for few-shot adaptation. For a detailed account of the training process, we refer readers to Algorithm 2 and our open source code.[1] Fig. 14 shows the average return on the test tasks versus the number of transitions collected during meta-training for all experiments. The shaded regions represent the confidence intervals corresponding to the average test rewards. We observe that all of our MELTS experiments significantly outperform all baselines in terms of sampling efficiency and asymptotic performance. MELTS with DPMM-VAE achieves better asymptotic performance than MELTS with Gaussian-VAE. Videos show the behavior of the agent when solving an individual test task during zero-shot adaptation.[2]

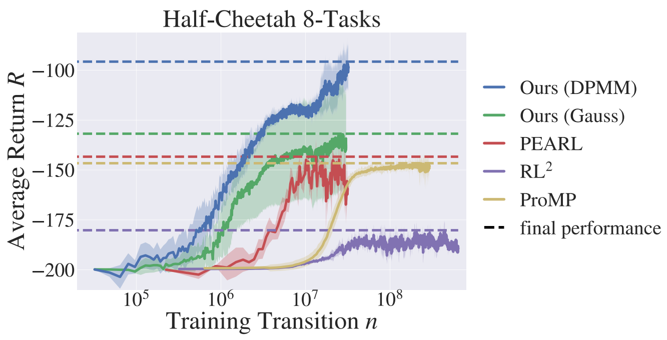


Fig. 14. Meta-testing performance versus accumulated samples collected during meta-training for the *Half-Cheetah 8-Task*, x-axis is shown in *log-scale*. All versions of our algorithm MELTS outperform the baselines in terms of sampling efficiency and asymptotic performance.

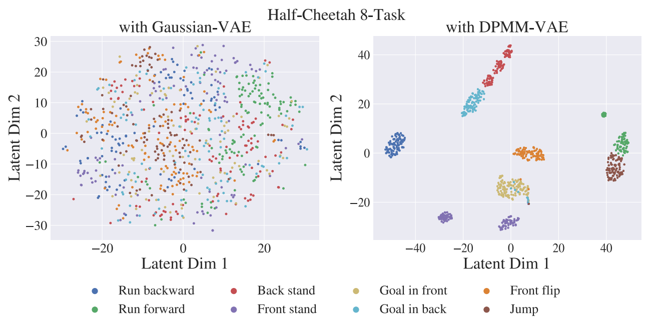
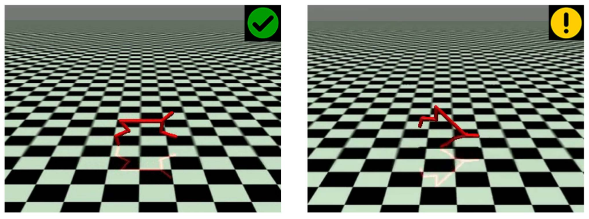


Fig. 15. Visualization of the latent task representations learned by MELTS (Gauss) and MELTS (DPMM). The task latent space learned with DPMM-VAE exhibits clustering based on the base tasks, while the space learned with Gaussian-VAE does not.

To inspect the structure of the task latent space, we sample contexts from the replay buffer that belongs to roll-outs of different tasks at the end of the training. We then use the context and task encoder to obtain the latent task variables for the sampled contexts. We visualize the latent task encodings with t-SNE projection, as shown in Fig. 15. All the task encodings in the latent space visualization for MELTS (Gauss) are centered

[2][Online]. Available: https://videoviewsite.wixsite.com/melts



(a) Run forward MELTS (DPMM) (b) Run forward MELTS (Gauss)

Fig. 16. Snapshot comparison of "Run forward" between (a) MELTS (DPMM) and (b) MELTS (Gauss). Note that even MELTS (Gauss) is able to reaches close to the individual task target, the behavior of Half-Cheetah robot is inaccurate, such actions are unacceptable in the real physical world[2].

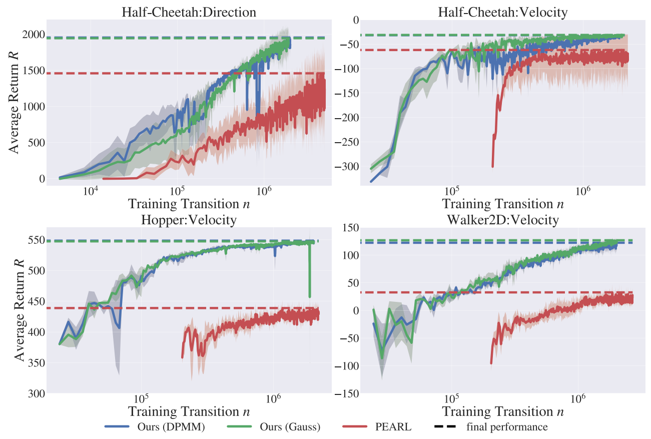


Fig. 17. Meta-testing performance versus accumulated transitions collected during meta-training for four parametric tasks. Our algorithm MELTS is more sample-efficient than PEARL in all four base tasks and outperforms PEARL in asymptotic performance in *Half-Cheetah: Direction*, *Hopper:Velocity* and *Walker2D:Velocity*.

around zero, which is the effect of using a single Gaussian prior. We observe that in some local regions, task encodings from the same base tasks are close to each other. However, overall the task encodings are scattered, and there is no obvious clustering phenomenon. In comparison, we observe in the latent space of MELTS with DPMM-VAE that the latent task encodings are clearly clustered according to the base tasks. In fact, the internal DPMM model identifies $K = 8$ clusters after the training. For tasks *run backward*, *run forward*, *front flip*, *back stand*, *goal in front*, *goal in back*, and *jump*, we can find a one-to-one mapping between the learned DPMM cluster and the base task. For the *forward stand* task, the DPMM learns two separate clusters that correspond to this task. This is not problematic as long as task encodings for different base tasks occupy different regions in the latent space.

Importantly, our MELTS framework, incorporating DPMM-VAE, demonstrates superior performance compared to Gauss-VAE in terms of overall return value and sample efficiency. Additionally, the contribution of DPMM is also reflected in the downstream action patterns captured by the agent. Improperly inferred task conditioning may interfere with or damage downstream policy learning, leading to incorrect action patterns executed by the agent in corresponding tasks. Such differences may not be intuitively observed on the learning curve. A clear example can be observed in both our released videos[2] and Fig. 16. For instance, in the task "Run forward", MELTS (DPMM) learns a normal action behavior for forward running motion, while MELTS (Gauss) produces inaccurate action patterns due to its entangled and unclear clustering inference.

3) Ablation Study: In this section, we thoroughly assess the individual contributions of each component in our framework, considering both parametric and non-parametric task settings. Specifically, we compare two variations of our MELTS framework: MELTS with DPMM-VAE and MELTS with Gauss-VAE, alongside the original PEARL framework, which serves as a baseline for comparison. This evaluation allows us to understand the impact of reconstruction training module (illustrated in Fig. 4), and further assess the specific contribution of DPMM in enhancing the overall performance of our framework when compared to MELTS with Gauss-VAE.

Parametric Task Distribution: To assess the scalability and efficiency of the proposed MELTS algorithm, we conduct experiments in four independent parametric base tasks: (1) The *HalfCheetah: Direction* task, involving stationary target directions. (2) The *HalfCheetah: Velocity* task, which focuses on stationary target velocities. (3) The *Hopper: Velocity* task, which targets stationary velocities of the Hopper robot. (4) The *Walker2D: Velocity* task, involving stationary target 2-dimensional velocities. For each base task, we prepare 100 task variations for training and evaluate the overall performance on another 30 variations. These task sets were also employed to evaluate the performance of the PEARL framework [5], allowing us to directly compare our results with PEARL. For conducting the comparison, we utilize the original code and parameters of PEARL [5]. It is important to note that our approach always reports the performance with zero-shot adaptation, meaning the reward is evaluated within the first episode of a test task. In contrast, PEARL adopts a few-shot adaptation approach, where the agent collects data on a new task for several episodes before evaluating its performance. This distinction ensures a clear assessment of our algorithm's effectiveness in comparison with PEARL. Additionally, it is worth mentioning that we do not evaluate RL^2 [3] and ProMP [43] in this study, as they have been outperformed by the PEARL framework in the specific evaluation scenarios.

We show the average rewards on the test tasks versus the accumulated number of transitions collected during meta-training on the four tasks in Fig. 17. Our algorithms achieve better sampling efficiency and asymptotic performance than PEARL in *Half-Cheetah:Direction*, *Hopper:Velocity* and *Walker2d:Velocity* and are more sample efficient than PEARL in all four base environments. We tested two compositions for task self-discovery: Gaussian-VAE and DPMM-VAE. We observe almost no differences in performance between the two compositions, which is expected because both tasks have only one base task and, therefore, no real clustering for the base tasks. Our DPMM model stays with $K = 1$ cluster during the meta-training because the parametric variation of the two tasks is adequately captured by a single Gaussian distribution.

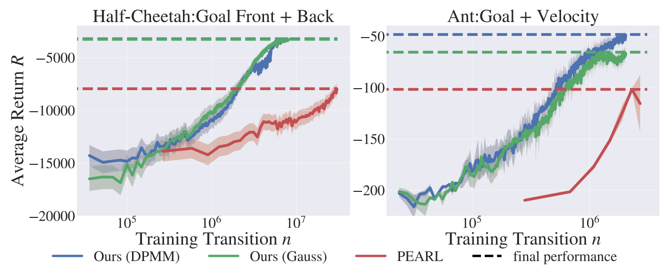


Fig. 18. Meta-testing performance versus accumulated transitions collected during meta-training for three nonparametric tasks. MELTS outperforms PEARL on all tasks.

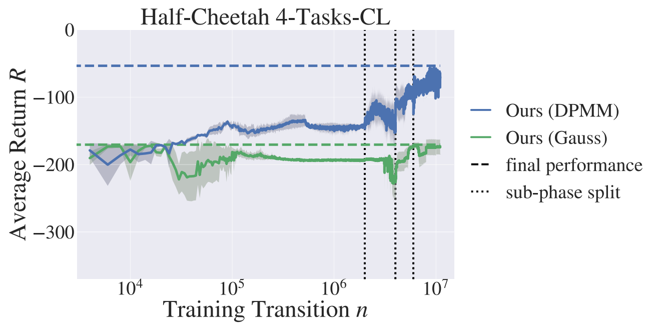


Fig. 19. Average test performance on all sampled tasks versus accumulated transitions collected during training for four non-parametric Half-Cheetah tasks. The training of Half-Cheetah 4-Tasks is conducted sequentially, with each sub-phase delimited by black vertical dotted lines. During each sub-phase, the agent has access to only one base task for training. The training sequence follows the order: "Run forward" → "Run back" → "Front stand" → "Goal in front".

Non Parametric Task Distribution: We proceed to evaluate our algorithm in simple nonparametric task environments with two base tasks. We construct two task environments: (1) Half-Cheetah with tasks of reaching different goal positions in front of it and reaching different goal positions behind it. (2) Ant with tasks of reaching different goal positions and moving at different velocities. The state space is 20-dimensional for Half-Cheetah, and 29-dimensional for Ant.

From Fig. 18, we observe that MELTS outperforms PEARL in terms of both sampling efficiency and asymptotic performance in both tasks. On Half-Cheetah tasks, the MELTS with DPMM-VAE and MELTS with Gaussian-VAE compositions show almost no differences in performance. On Ant tasks, they are comparable in sampling efficiency but MELTS with DPMM-VAE achieves better asymptotic performance. This is expected as both versions of MELTS can successfully represent different tasks as there are only two nonparametric tasks in this setting.

Contribution of Task Self-Discovery: To assess the impact of task self-discovery on the performance of MELTS, we compare two variations of our algorithms: MELTS (DPMM) and MELTS (Gauss), to investigate the contribution of the DPMM+moVB in upstream task inference and its influence on downstream policy learning. We conduct the experiment using a stream of four non-parametric Half-Cheetah tasks, including the sequences "Run forward", "Run back", "Front stand", and "Goal in front". The agent has access to only one base task at a certain phase interval (2 M steps for each sub-phase). For each base task, we randomly sample 10 variations for training, and for testing, we use all sampled task variations for evaluation. Each trial runs three times with different seeds. The results are shown in Fig. 19, with the dotted vertical lines indicating the split sub-phases. It is evident that, starting from the second stage when the agent switches to a new base task, the reward value of MELTS (DPMM) rapidly converges to a higher average reward level after the short decline. This means the agent with DPMM+moVB can quickly identify the current tasks and adaptation new action patterns to fit to the new tasks while preserving the knowledge it has acquired. Consequently, its performance in the overall task substantially improves. In contrast, MELTS (Gauss) fails to distinguish between tasks due to the assumption of a single Gaussian in its prior space, leading to ill-inferred task conditioning that may interfere with downstream policy learning, resulting in inaccurate action patterns or even task failures. Furthermore, to further support our claim, we provide an intuitive visualization example for comparison in their final performance.[2]

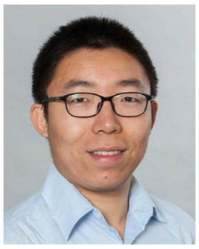
VIII. Conclusion

We presented MELTS, a context-based off-policy algorithm for meta-reinforcement learning in broad and non-parametric task distributions. We include Bayesian nonparametric models for task self-discovery, which enables our model to self-adaptively recognize the qualitatively different base tasks in the task distribution as distinct clusters in the task latent space. It also learns the parametrization for each cluster distribution in order to capture the parametric variations in the subtasks of a base task. We further introduced a zero-shot adaptation mechanism on the basis of our recurrence-based encoder-decoder strategy. On both parametric and nonparametric meta-RL task benchmarks, MELTS outperforms previous meta-RL algorithms in terms of sampling efficiency and asymptotic performance.

References

[1] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1126–1135.
[2] A. Nagabandi et al., "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," 2018, *arXiv:1803.11347*.
[3] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "RL^2: Fast reinforcement learning via slow reinforcement learning," 2016, *arXiv:1611.02779*.
[4] J. X. Wang et al., "Learning to reinforcement learn," 2016, *arXiv:1611.05763*.
[5] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 5331–5340.
[6] L. Zintgraf et al., "VariBAD: A very good method for Bayes-adaptive deep RL via meta-learning," 2019, *arXiv: 1910.08348*.
[7] Z. Bing, D. Lerch, K. Huang, and A. Knoll, "Meta-reinforcement learning in non-stationary and dynamic environments," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 3, pp. 3476–3491, Mar. 2023.
[8] T. Yu et al., "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Proc. Conf. Robot Learn.*, 2020, pp. 1094–1100.
[9] W. Zhuge, C. Hou, X. Liu, H. Tao, and D. Yi, "Simultaneous representation learning and clustering for incomplete multi-view data," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, pp. 4482–4488.

[10] M. Sayed-Mouchaweh and E. Lughofer, *Learning in Non-Stationary Environments: Methods and Applications*. Berlin, Germany: Springer, 2012.
[11] M. Botvinick, S. Ritter, J. X. Wang, Z. Kurth-Nelson, C. Blundell, and D. Hassabis, "Reinforcement learning, fast and slow," *Trends Cogn. Sci.*, vol. 23, no. 5, pp. 408–422, 2019.
[12] L. P. Kaelbling, "The foundation of efficient robot learning," *Science*, vol. 369, no. 6506, pp. 915–916, 2020.
[13] Y. M. Asano, C. Rupprecht, and A. Vedaldi, "Self-labelling via simultaneous clustering and representation learning," 2019, *arXiv: 1911.05371*.
[14] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 478–487.
[15] K. Tian, S. Zhou, and J. Guan, "DeepCluster: A general clustering framework based on deep learning," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discov. Databases*, Springer, 2017, pp. 809–825.
[16] X. Ji, J. F. Henriques, and A. Vedaldi, "Invariant information clustering for unsupervised image classification and segmentation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 9865–9874.
[17] H. Zhong, C. Chen, Z. Jin, and X.-S. Hua, "Deep robust clustering by contrastive learning," 2020, *arXiv: 2008.03030*.
[18] E. Nalisnick and P. Smyth, "Stick-breaking variational autoencoders," 2016, *arXiv:1605.06197*.
[19] M. C. Hughes and E. Sudderth, "Memoized online variational inference for Dirichlet process mixture models," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 1133–1141.
[20] Z. Wang, Y. Ni, B. Jing, D. Wang, H. Zhang, and E. Xing, "DNB: A joint learning framework for deep Bayesian nonparametric clustering," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 12, pp. 7610–7620, Dec. 2022.
[21] M. Al-Shedivat, T. Bansal, Y. Burda, I. Sutskever, I. Mordatch, and P. Abbeel, "Continuous adaptation via meta-learning in nonstationary and competitive environments," 2017, *arXiv: 1710.03641*.
[22] J. Zhang et al., "MetaCURE: Meta reinforcement learning with empowerment-driven exploration," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 12 600–12 610.
[23] H. Wang, J. Zhou, and X. He, "Learning context-aware task reasoning for efficient meta-reinforcement learning," 2020, *arXiv: 2003.01373*.
[24] H. Ren, A. Garg, and A. Anandkumar, "Context-based meta-reinforcement learning with structured latent space," in *Proc. Skills Workshop Neural Inf. Process. Syst.*, 2019, p. 5.
[25] A. Nagabandi, C. Finn, and S. Levine, "Deep online learning via meta-learning: Continual adaptation for model-based RL," 2018, *arXiv:1812.07671*.
[26] G. Jerfel, E. Grant, T. Griffiths, and K. A. Heller, "Reconciling meta-learning and continual learning with online mixtures of tasks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 9119–9130.
[27] Z. Wang, C. Chen, and D. Dong, "Lifelong incremental reinforcement learning with online Bayesian inference," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 8, pp. 4003–4016, Aug. 2022.
[28] Y. W. Teh, "Dirichlet process," *Encyclopedia Mach. Learn.*, vol. 1063, pp. 280–287, 2010.
[29] J. Sethuraman, "A constructive definition of Dirichlet priors," *Statistica Sinica*, vol. 4, pp. 639–650, 1994.
[30] S. J. Gershman and D. M. Blei, "A tutorial on Bayesian nonparametric models," *J. Math. Psychol.*, vol. 56, no. 1, pp. 1–12, 2012.
[31] T. S. Ferguson, "A Bayesian analysis of some nonparametric problems," *Ann. Statist.*, vol. 1, pp. 209–230, 1973.
[32] S. Jain and R. M. Neal, "A split-merge Markov chain Monte Carlo procedure for the Dirichlet process mixture model," *J. Comput. Graphical Statist.*, vol. 13, no. 1, pp. 158–182, 2004.
[33] Y. Ulker, B. Günsel, and T. Cemgil, "Sequential Monte Carlo samplers for Dirichlet process mixtures," in *Proc. 13th Int. Conf. Artif. Intell. Statist.*, 2010, pp. 876–883.
[34] D. M. Blei and M. I. Jordan, "Variational inference for Dirichlet process mixtures," *Bayesian Anal.*, vol. 1, no. 1, pp. 121–143, 2006.
[35] J. Rocca and B. Rocca, "Understanding variational autoencoders," 2019. [Online]. Available: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73
[36] T. Haarnoja et al., "Soft actor-critic algorithms and applications," 2018, *arXiv: 1812.05905*.
[37] L. Deng, "The MNIST database of handwritten digit images for machine learning research," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, Nov. 2012.
[38] E. Nalisnick and P. Smyth, "Stick-breaking variational autoencoders," 2017, *arXiv:1605.06197*.
[39] N. Dilokthanakul et al., "Deep unsupervised clustering with Gaussian mixture variational autoencoders," 2017, *arXiv:1611.02648*.
[40] Z. Jiang, Y. Zheng, H. Tan, B. Tang, and H. Zhou, "Variational deep embedding: An unsupervised and generative approach to clustering," 2017, *arXiv:1611.05148*.
[41] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. 11, pp. 2579–2605, 2008.
[42] G. Brockman et al., "OpenAI gym," 2016, *arXiv:1606.01540*.
[43] J. Rothfuss, D. Lee, I. Clavera, T. Asfour, and P. Abbeel, "ProMP: Proximal meta-policy search," 2018, *arXiv: 1810.06784*.



Zhenshan Bing received the BS degree in mechanical design manufacturing and automation and the MEng degree in mechanical engineering from the Harbin Institute of Technology, China, in 2013 and 2015, and the doctorate degree in computer science from the Technical University of Munich, Germany, in 2019. He is currently a postdoctoral researcher with Informatics 6, Technical University of Munich, Munich, Germany.

Yuqi Yun received the BS degree in physics and mathematics from Duke University, Durham, NC, USA, in 2018., and the master's degree in data engineering and analytics from the Technical University of Munich, Germany, in 2022. Her research interest include reinforcement learning.

Kai Huang received the BSc degree from Fudan University, China, in 1999, the MSc degree from the University of Leiden, The Netherlands, in 2005, and the PhD degree from ETH Zurich, Switzerland, in 2010. He joined Sun Yat-Sen University as a professor in 2015. He was appointed as the director with the Institute of Unmanned Systems, School of Data and Computer Science in 2016.

Alois Knoll (Fellow, IEEE) received the diploma (MSc) degree in electrical/communications engineering from the University of Stuttgart, Germany, in 1985 and the PhD (*summa cum laude*) degree in computer science from the Technical University of Berlin, Germany, in 1988. He served on the faculty of the Computer Science Department, TU Berlin until 1993. He joined the University of Bielefeld, Germany as a full professor and served as the director of the Technical Informatics Research Group until 2001. Since 2001, he has been a professor with the Department of Informatics, Technical University of Munich (TUM), Germany. He was also on the board of directors of the Central Institute of Medical Technology, TUM (IMETUM). From 2004 to 2006, he was executive director of the Institute of Computer Science, TUM. Between 2007 and 2009, he was a member of the EU's highest advisory board on information technology, ISTAG, the Information Society Technology Advisory Group, and a member of its subgroup on Future and Emerging Technologies (FET). In this capacity, he was actively involved in developing the concept of the EU's FET Flagship projects. His research interests include cognitive, medical and sensor-based robotics, multi-agent systems, data fusion, adaptive systems, multimedia information retrieval, model-driven development of embedded systems with applications to automotive software and electric transportation, as well as simulation systems for robotics and traffic.