This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3185549



# Meta-Reinforcement Learning in Non-Stationary and Dynamic Environments

Zhenshan Bing[1], David Lerch[1], Kai Huang*, and Alois Knoll, *Senior Member, IEEE*

**Abstract**—In recent years, the subject of deep reinforcement learning (DRL) has developed very rapidly, and is now applied in various fields, such as decision making and control tasks. However, artificial agents trained with RL algorithms require great amounts of training data, unlike humans that are able to learn new skills from very few examples. The concept of meta-reinforcement learning (meta-RL) has been recently proposed to enable agents to learn similar but new skills from a small amount of experience by leveraging a set of tasks with a shared structure. Due to the task representation learning strategy with few-shot adaptation, most recent work is limited to narrow task distributions and stationary environments, where tasks do not change within episodes. In this work, we address those limitations and introduce a training strategy that is applicable to non-stationary environments, as well as a task representation based on Gaussian mixture models to model clustered task distributions. We evaluate our method on several continuous robotic control benchmarks. Compared with state-of-the-art literature that is only applicable to stationary environments with few-shot adaption, our algorithm first achieves competitive asymptotic performance and superior sample efficiency in stationary environments with zero-shot adaption. Second, our algorithm learns to perform successfully in non-stationary settings as well as a continual learning setting, while learning well-structured task representations. Last, our algorithm learns basic distinct behaviors and well-structured task representations in task distributions with multiple qualitatively distinct tasks.

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

## 1 Introduction

HUMANS are exceptionally proficient in learning new skills based on very few examples and trials, since they have well-established representations of the world and great amounts of experience from previously acquired skills from which they can learn. In contrast, modern artificial agents trained with standard RL algorithms usually lack this ability, since they can only learn new skills from scratch by performing extensive training. For example, for learning the dexterity for an artificial robotic hand to solve a Rubik's Cube, OpenAI reported a cumulative experience of 13 thousands years [1]. On the contrary, humans are able to manipulate the cube nearly instantaneously, as they have learned how to manipulate objects in general beforehand.

One approach for tackling this open challenge is presented by meta-RL, which aims to learn a priori from a set of training tasks with shared structure, to enable quick adaptation to similar but new tasks, instead of learning them from scratch. While the general concept of meta-learning was proposed decades ago [2], modern meta-learning approaches can be categorized into three lines of work. A first line of work utilizes recurrent neural networks (RNNs) fed with data from previous transitions to implicitly obtain a notion of the environment and task through the hidden states of the recurrent units [3], [4]. A second category of approaches is based on model-agnostic meta-learning (MAML) [5]. With this method, agents learn a highly sensitive parameter prior, which forms the basis for quick adaptation to new tasks through gradient descent. Both concepts adopt on-policy RL algorithms, leading to sample-inefficient training. Based on this premise, Rakelly et. al. [6], in a third line of work, proposed a model-free and off-policy method, utilizing a variational auto-encoder (VAE) [7] in combination with soft actor-critic (SAC) [8]. Their algorithm probabilistic embeddings for actor-critic RL (PEARL) [6] achieves state-of-the-art results and significantly outperforms previous methods in sample efficiency and asymptotic performance.

Despite the great improvement, PEARL has several limitations and is far cry from achieving human-like performance or even applicability outside of stationary scenarios. Furthermore, all previous meta-RL approaches, including PEARL, are only developed for narrow task distributions. Using a housekeeper robot as an example, with current methods a robot would be able to quickly learn how to pick up and place objects in a new position, if it had been trained previously on a set of different positions. However, for the robot to be a helpful housekeeper, we would like to teach it how to set a table for dinner once it has already learned to unload the dishwasher, open and close cupboards and similar tasks, i.e., we expect it to learn new and qualitatively distinct tasks based on a broad set of skills it has already acquired. Supported by [9], we suspect one reason for this limitation of PEARL is the usage of a single Gaussian as latent distribution and instead propose a generative model that leads to a mixture of Gaussians as latent distribution.

Furthermore, in the literature, there is currently no algorithm that is model-free and applicable to non-stationary environments. Previous approaches are either model-free and achieve high performance, but are only applicable

- *Z. Bing, D. Lerch, and A. Knoll are with the Department of Informatics, Technical University of Munich, Germany. E-mail: bing@in.tum.de, davidn.lerch@web.de, knoll@in.tum.de*
- *K. Huang is with Sun Yat-sen University, Guangzhou and Peng cheng Laboratory, Shenzhen.(huangk36@mail.sysu.edu.cn)*
- [1] *Both authors contribute equally.*
- * *Corresponding author.*

*Manuscript received April 19, 2005; revised August 26, 2015.*

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 06,2022 at 11:01:49 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3185549



to stationary scenarios (MAML [5], PEARL [6], CASTER [10]), or applicable to non-stationary environments, but model-based, thus sample-efficient but in general achieving less asymptotic performance (GrBAL [11], MOLe [12]), or model-free and applicable to non-stationary environments but on-policy, such that they exhibit bad sample efficiency ($RL^2$ [3], LSTM A2C [4], SNAIL [13], Rubik's Cube [1]). Therefore, we build on the sample efficiency and asymptotic performance of PEARL and design a zero-shot adaptation strategy applicable to non-stationary environments.

In this work, we introduce Continuous Environment Meta-Reinforcement Learning (CEMRL), an efficient, off-policy algorithm with zero-shot adaptation, applicable to a variety of meta-RL settings such as non-stationary environments and broad task distributions, based on three main concepts. First, we design a zero-shot adaptation strategy based on recent context to transfer the sample efficiency and asymptotic performance of previous few-shot methods to non-stationary and continual learning settings. Second, compared with previous methods using only a Gaussian for task representation, we derive an encoder from a generative model based on Gaussian mixture models (GMMs), to represent complex task distributions with cluster structure. In providing discrete task indicators naturally by design, the encoder constitutes the basis for transferring successful methods from multi-task RL to the meta-RL setting in the future. It should be noted that, although Gaussian or Gaussian mixture models are widely used to approximate the dynamic models with uncertainty and stochasticity in model-based RL algorithms, the novelty of using GMM in CEMRL is to formalize the concept of the parametric and non-parametric variability in meta-RL tasks, in which the mixtures are used to model the non-parametric base tasks and each mixture is used to model the parametric sub-tasks conditioned on its category. Third, our encoder and decoder are trained in an unsupervised manner, by reconstructing the task describing the Markov decision process (MDP), which allows them to learn a well structured latent space with few experience. This strategy decouples the training of the task inference encoder from the training of the conditioned policy and avoids the problem of the complex joint training. Finally, CEMRL is demonstrated by experiments in several continuous locomotion problems. On four accepted meta-RL benchmarks, CEMRL achieves competitive performance and superior sample efficiency compared with PEARL, while it learns successful behaviors in non-stationary environments, that PEARL cannot solve. We also show that CEMRL can learn distinct behaviors and well-structured task representations in task distributions with multiple qualitatively distinct tasks. To the best of the authors' knowledge, CEMRL is the first model-free meta-RL algorithm that can solve non-stationary environments requiring zero-shot adaptation and uses GMMs as generative models to represent broad task distributions.

## 2 Background

In standard reinforcement learning, a task can be formalized as an MDP and the goal is to find a policy that allows an agent to solve this task, i.e., maximizing the expected reward in this MDP. Multi-task and meta-reinforcement learning extend this concept and aim to solve not only one specific, but multiple tasks with some common structure. This can be interpreted as an MDP where both the transition and reward function are dependent on some task indicator $\boldsymbol{z}$. Although it is possible to learn a specific policy $\pi_i$ for each task $i$ from scratch, multi-task RL aims to learn one single policy that is able to generalize and solve all tasks simultaneously and learn more efficiently, leveraging the shared structure of the tasks. Meta-RL, often framed as *learning to learn*, aims to leverage a set of tasks with shared structure in a training phase, such that it can quickly adapt to new and similar tasks that are not seen during training.

### 2.1 Meta-Reinforcement Learning

A meta-reinforcement learning problem consists of a set of training tasks $\mathcal{D}_{\mathcal{T}}^{train}$ and a set of test tasks $\mathcal{D}_{\mathcal{T}}^{test}$, both drawn from the same distribution over tasks $p(\mathcal{T})$. The agent is supposed to leverage the training tasks during so-called meta-training in such a way that it is able to perform the previously unseen test tasks during the so-called meta-testing by small adaptations only, without needing to learn them from scratch. The meta-RL objective is as

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathcal{T}\sim\mathcal{D}_{\mathcal{T}}^{test}} \left[ \mathbb{E}_{\boldsymbol{\tau}\sim p(\boldsymbol{\tau}|\pi_{\boldsymbol{\theta}})} \left[ \sum_{t\geq 0} \gamma^t r_t \right] \right], \quad (1)$$

Based on the general problem definition of meta-RL, we introduce these two perspectives of how to view and tackle the meta-RL problem, which provide the theoretical framework of our algorithms.

#### 2.1.1 Meta-RL as learning an adaptation procedure

The first approach aims to learn a procedure $u_{\boldsymbol{\phi}}$ for adapting the policy $\pi_{\boldsymbol{\theta}}$ to specific tasks. During training, the agent uses the training tasks to learn the adaptation procedure and applies it to new tasks during the test phase. This can be formally defined as:

$$\boldsymbol{\theta}^*, \boldsymbol{\phi}^* = \arg\max_{\boldsymbol{\theta},\boldsymbol{\phi}} \mathbb{E}_{\mathcal{T}\sim\mathcal{D}_{\mathcal{T}}^{test}} \left[ \mathbb{E}_{\boldsymbol{\tau}\sim p(\boldsymbol{\tau}|\pi_{\boldsymbol{\theta}'})} \left[ \sum_{t\geq 0} \gamma^t r_t \right] \right], \quad (2)$$
$$\text{where } \boldsymbol{\theta}' = u_{\boldsymbol{\phi}}\left(\mathcal{D}_{\mathcal{T}}, \boldsymbol{\theta}\right).$$

We denote the adaptation procedure (also referred to as update function) as $u_{\boldsymbol{\phi}}(\mathcal{D}_{\mathcal{T}}, \boldsymbol{\theta})$, to emphasize that it takes the initial parameters $\boldsymbol{\theta}$ and returns task-specific parameters $\boldsymbol{\theta}'$ based on the data from task $\mathcal{D}_{\mathcal{T}}$. The specification of the procedure is abstractly denoted as $\boldsymbol{\phi}$ and depends significantly on the concrete workings of the specific algorithm. During meta-training, the agent learns optimal initial parameters $\boldsymbol{\theta}^*$ and an optimal adaptation procedure $\boldsymbol{\phi}^*$, which are the basis for adaptation during meta-testing.

#### 2.1.2 Meta-RL as task inference

The second approach views meta-RL as an extension of multi-task RL. While in multi-task RL task information $\boldsymbol{z}$ is known as a prior to the algorithm for all tasks, it is unknown in meta-RL tasks. From this perspective, meta-RL can be viewed as learning a multi-task RL policy $\pi(\boldsymbol{a} \mid \boldsymbol{s}, \boldsymbol{z})$, while additionally learning how to infer the task $\boldsymbol{z}$, using the set of training tasks $\mathcal{D}_{\mathcal{T}}^{train}$. Therefore, some inference method is required to infer the task from the transition data.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 06,2022 at 11:01:49 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3185549



### 2.2 Meta-RL Environments

In this section, we introduce the characteristics of the task distribution $p(\mathcal{T})$ and how such a distribution is constituted in terms of the underlying MDPs. In this work, we use the terms task distribution and environment interchangeably. The environment is a set of tasks with a shared structure, in which each task is defined by its specific MDP.

#### 2.2.1 Parametric and Non-Parametric Variability

To allow an agent to generalize over training and test tasks, they need to share some common structure. Yu et. al. [14] categorized task distributions into parametric and non-parametric task distributions. In parametric environments, the reward and transition function of the specific MDP differ only in specific parameters. Formally, we can denote such an environment and a set of tasks $\mathcal{D}_{\mathcal{T}}$ of $N$ tasks as

$$\mathcal{D}_{\mathcal{T}} = \{\mathcal{T}_i\}_{i=1}^{N} = \{p_i(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\vartheta}_i), r_i(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{\varphi}_i)\}_{i=1}^{N}, \tag{3}$$

where $\boldsymbol{\vartheta}_i$ and $\boldsymbol{\varphi}_i$ are parameters of the transition and reward function, respectively, drawn from some distribution. Each task $\mathcal{T}_i$ is an individual MDP, but the state space, action space, and discount factor are shared among tasks. In non-parametric environments, the tasks are qualitatively distinct and cannot described by parameter variations. Both properties can lead to environments with cluster-like structure. They consist of a number of qualitatively distinct tasks (referred to as *base-tasks*), where each *base-task* itself has multiple sub-tasks that constitute a parametric distribution.

#### 2.2.2 Stationary and non-stationary environments

Environments can also be classified as stationary and non-stationary environments. In stationary environments, the task specification changes at an episodic level, i.e., the task describing MDP is fixed during an episode and only changed between episodes. Therefore, an algorithm is only required to perform a few-shot adaptation to solve such environments. In contrast, in non-stationary environments, the task can potentially change every timestep. To solve such an environment, the algorithm must perform a zero-shot (online or continuous) adaptation. However, to be able to utilize data from prior timesteps, the environment still needs to exhibit *local consistency* [11] over a period of time, i.e., the environment stays fixed for at least some timesteps. For example, an agent running at a specified velocity throughout the episode without anything changing in the environment is a stationary task. While in a non-stationary task, the goal velocity can change suddenly or the robot can suddenly suffer from a motor malfunction, which manifests as a new transition function from an MDP point of view.

#### 2.2.3 Adaptation in Meta-RL environments

In meta-RL there are two different settings for the adaptation to new tasks, few-shot (episodic) adaptation and zero-shot (online) adaptation. In a few-shot adaptation setting, when being exposed to a new task, the agent is allowed to collect data in this new task for a few episodes and adapt its policy after each episode based on this data, to get an increasingly better notion of the environment. Due to this episode-wise adaptation, few-shot adaptation is only applicable to stationary environments. Furthermore, this method is only reasonable if the agent does not need to perform the task successfully directly after being exposed to a new task, but only after a few trials. In contrast, in a zero-shot adaptation setting, the agent is supposed to perform successfully in a new task instantaneously, without any previous data collection. This makes it necessary to adapt the behavior within the episode, at the transition level. Compared to few-shot adaptation, zero-shot adaptation is not only applicable to stationary, but also to non-stationary environments, where the agent might be exposed to a new task at every timestep and has to react immediately.

#### 2.2.4 Classic benchmark environments

One category of problems used exhaustively throughout the literature is high-dimensional locomotion tasks based on the *MuJoCo* physics engine [15], where simple simulated robot agents are required to run. Several papers in the field (e.g., [5], [16], [11], [6], [17]) used extensions of these problems as multi-task and meta-RL environments. Thereby, agents are required, for example, to run in different directions, run at different speeds (parametric reward functions), or physical properties such as mass, damping, and friction are changed (parametric transition functions). The typical robotic agents are Half-Cheetah, Ant, Walker, and Hopper [18].

### 2.3 Probabilistic Embeddings for Actor-Critic RL

A major baseline for our work is the meta-RL algorithm probabilistic embeddings for actor-critic RL (PEARL) [6]. PEARL is developed for fast few-shot meta-RL and sample-efficient training. Compared to other meta-RL algorithms, PEARL is an off-policy, model-free algorithm and achieves good asymptotic performance as well as sample efficiency.

PEARL tackles the meta-RL problem in the notion of the meta-RL as task inference. It combines a VAE for task inference with SAC [8] for policy learning, where the state is augmented by the task encoding $\boldsymbol{z}$. The encoder of the VAE uses so-called context tuples $\boldsymbol{c}_n^{\mathcal{T}} = (\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{r}_n, \boldsymbol{s}'_n)$ as inputs, incorporating MDP defining data for each transition, and encodes them as independent Gaussian factors $\Psi_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{c}_n)$. The encoder network is implemented as a multilayer perceptron (MLP). To obtain the overall posterior estimate $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{c}_{1:N})$, the Gaussian factors are multiplied

$$q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{c}_{1:N}) \propto \prod_{n=1}^{N} \Psi_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{c}_n) \tag{4}$$

Hence, the encoder is permutation-invariant and can incorporate arbitrary amounts of sampled context from a task. For reconstruction and decoding, PEARL uses the Bellman error $\mathcal{L}_Q$ of the critic from SAC as reconstruction loss. Thereby the encoder receives gradients from the Q-function. The corresponding ELBO for the VAE is

$$\mathbb{E}_{\mathcal{T}}\left[\mathbb{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{c}^{\mathcal{T}})}\left[\mathcal{L}_Q(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{s}', \boldsymbol{z})\right] + \beta\, \mathbb{KL}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{c}^{\mathcal{T}})||p(\boldsymbol{z})\right)\right] \tag{5}$$

with $p(\boldsymbol{z}) \sim \mathcal{N}(0, \boldsymbol{I})$, a unit Gaussian prior over tasks and $\beta$, a hyperparameter to weight the KL-divergence.

#### 2.3.1 Meta-Training

Each training epoch consists of a data collection and optimization phase. First, for each task, the data is collected

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 06,2022 at 11:01:49 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3185549



using the task conditioned policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{z})$, where $\boldsymbol{z}$ is either sampled from the posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{c}_{1:N})$ or the prior $p(\boldsymbol{z})$. The data is stored in an individual replay buffer $\mathcal{B}^i$ for each task. Thereby, PEARL assumes clear task boundaries and knowledge from which task data is currently collected. In the optimization phase, for each task, the losses for the VAE and SAC are computed and the gradient of the averaged losses is used to update the parameters of the policy, critic and encoder.

#### 2.3.2 Meta-Testing

During meta-testing, the policy is adapted to new tasks in the few-shot manner. For few episodes, context data is collected and used to update the posterior belief distribution for the task at hand. With newly collected data from each episode, the task hypothesis $\boldsymbol{z}$ becomes increasingly accurate and the task can be solved increasingly optimally. The average reward from the last episode is reported as the test performance.

## 3 Related Work

The general idea of meta learning aims to learn a prior from a distribution of tasks, to enable the agent to adapt to similar tasks quickly. Modern meta-RL approaches can be classified into three main lines of work, with each approaching the meta-RL problem statement in a different way.

### 3.1 Recurrence-Based Meta-RL

The first category of approaches uses recurrent models, such as recurrent neural networks (RNNs) with long short-term memory (LSTM) or gated recurrent units (GRUs), as policy networks. The basic idea behind these approaches is to leverage the ability of recurrent models to represent long-term relationships in meta-RL by feeding relevant transition data from previous experience into the network. Concretely, at each timestep, the network receives a tuple of action, state, reward, termination signal, and other task-relevant information depending on the specific setting. Through the sequence of transitions, the recurrent units are able retrieve specific task information. $RL^2$ [3] leveraged the hidden states of GRUs to encode and memorize relevant task information over time steps and episode boundaries. LSTM A2C [4] followed a similar approach but used LSTM and the advantage actor-critic algorithm [19] as the optimizer. Both $RL^2$ and LSTM A2C were validated on bandit problems and visual navigation tasks. Similarly, SNAIL [13] used a combination of temporal convolutions [20] and soft attention [21] as a more efficient alternative to recurrent units. Further, OpenAI [1] achieved impressive results in dexterous manipulation by solving a Rubik's cube with a robotic hand. Their algorithm was based on an LSTM policy network, optimized with PPO. However, the training required huge amounts of data and time.

The methods from this line of work approach the meta-RL problem in the notion of meta learning in POMDPs, as they implicitly contain a belief state of the task in the hidden state of the LSTM or GRU. In general, recurrence-based meta-RL algorithms are applicable to both stationary and non-stationary settings, as the hidden states of the recurrent units are updated at each timestep. However, due to the RL optimizers used, they are on-policy and comparably sample-inefficient.

### 3.2 Gradient-Based Meta-RL

Gradient-based methods build on the foundation of MAML [5] and leverage the perspective of meta-RL as the adaptation procedure. MAML aimed to learn a highly sensitive weight initialization from a set of given tasks, in order to adapt quickly to new tasks by taking only few gradient steps. A great benefit of MAML is the fact that it is model-agnostic, i.e., the algorithm can be used with any architecture of differentiable model, such as neural networks.

In the original paper [5], the MAML adaptation procedure is used in a model-free, episodic adaptation setting for RL. Furthermore, as the algorithm needs to collect new data for adaptation, it is strictly limited to few-shot settings. Al-Shedivat [22] proposed a method for few-shot adaptation in non-stationary environments based on MAML. They view non-stationarity as a Markov chain of tasks that are introduced on an episodic level. Nagabandi et al. [11] developed GrBAL, a model-based online adaptation algorithm based on MAML. Instead of adapting a policy network on an episodic basis, they follow a model-based approach and adapt the parameters of a dynamics function network on a per-time-step basis. Thus, the GrBAL can cope with non-stationary, locally consistent environments. Extending GrBAL to the continual learning setting, Nagabandi et al. [12] introduced Meta-Learning for Online Learning (MOLe). The method combined MAML with an Expectation-Maximation (EM) algorithm and a Chinese restaurant process as task prior. With this approach, new tasks can be learned without forgetting old tasks. Additionally, in contrast to the previously described method [11], which only takes data from some previous timesteps into account, all data can be incorporated to build up specific task knowledge.

### 3.3 Inference-Based Meta-RL

Inference-based algorithms approach the meta-RL objective in the notion of meta-RL as task inference. The basic idea is to identify the task and learn a task-conditioned policy. This requires a mechanism to learn task representations and identify single tasks with some strategy for policy learning.

Gupta et al. [23] first utilized the concept of encodings in a latent space, by introducing structured noise from a latent space into a policy network. The noise from the latent space is modeled as Gaussian distribution. With the parameters of the Gaussian being learnable per task, it can be used to encode task information. Their algorithm MAESN optimizes an adaptive policy together with the latent space using MAML. Through this, during test-time, MAESN can efficiently test different task hypotheses imposed through Gaussian parameters and adapt to collected data, similar to MAML. In contrast, Lan et al. [24] trained an RNN encoder for each task in an MAML-like manner together with a shared policy. This method is especially effective for out-of-distribution tasks during meta-testing. Hausman et al. [25] learned a skill embedding space with a combination of a variational inference formulation and entropy-regularized policy gradient formulation. Through modulations in the

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 06,2022 at 11:01:49 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3185549



skill embedding space, the agent can interpolate between different skills. Igl et al. [26] proposed a method leveraging an ELBO-based auxiliary loss and incorporating an inductive bias as a task identification approach. Humplik et al. [17] motivated their method from the POMDP view on meta-RL and use what they call privileged task information, which is simply task-specifying parameters like goal positions etc., which are known for hand-crafted environments. Utilizing this information, task inference is reduced to supervised learning of privileged information. Thus it is possible to separate task inference from task fulfillment, i.e., no information must be backpropagated from the policy to the inference module. However, this method is based on the strong assumption of access to privileged information. Most related to our work, Rakelly [6] proposed PEARL, an efficient, off-policy algorithm, that combines a VAE with SAC, as discussed in Section 2.3. Extending PEARL to broader task distributions, Ren [9] proposed a structured latent space with a combination of a Dirichlet distribution as base task distribution and Gaussian distribution as so-called style factors. They conduct validation experiments on a point-robot navigation task. During the completion of our work, Wang et al. [10] present CASTER, which adopts a latent Graph Neural Network architecture [27] as encoder and decomposes meta-RL into task exploration, task inference, and task fulfillment. They report slightly increased performance regarding efficient exploration, performance, and sample efficiency in stationary environments.

## 4 Problem Statement

In this work, we focus on meta-RL methods for continuous control in non-stationary and broad task distributions. We approach the problem in the notion of meta-RL as task inference and utilize an encoder for task inference and a task-conditioned policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \boldsymbol{s}, \boldsymbol{z})$ as the actor. In particular, we aim to achieve the following goals. Our method should be applicable to a variety of task distributions (also referred to as environments). The task distributions can either be parametric, i.e., the MDPs that describe the tasks differ only in specific parameters, or non-parametric, i.e., there is a number of qualitatively distinct base tasks, where each base task itself has multiple sub-tasks that constitute a parametric distribution. Further, the environments can either be stationary, which means the task is the same during the whole episode and its corresponding MDPs is fixed respectively, or non-stationary, which means the task can potentially change at every timestep within an episode. Beyond that, we consider continual learning (also known as lifelong learning) environments, in which the tasks are not accessible right from the start, but introduced sequentially or are dependent on each other. To cope with the task changes within episodes in non-stationary environments and continual learning settings, we aim to develop an algorithm that is able perform zero-shot adaptation.

As introduced and explained in prior work, the task of designing efficient model-free meta-RL algorithms that can perform robotic tasks in non-stationary environments remains unsolved. And to the best of our knowledge, there is no approach that combines all advantages from previous work, i.e., a model-free, off-policy algorithm that is applicable to non-stationary environments. Furthermore, previous inference-based meta-RL methods use simple latent spaces for task representation, e.g., a single isotropic Gaussian in PEARL. However, to represent complex task distributions, these representations are not rich enough.

## 5 Methodology

In this section, we first give an overview of our proposed algorithm CEMRL. Then we explain the strategy to make CEMRL applicable in non-stationary environments, derive the generative model and explain how we implement the resultant encoder and decoder. We finally summarize the algorithm CEMRL with its pseudocode.

### 5.1 Overview

In this work, we leverage the notion of meta-RL as task inference. Similar to PEARL, we also follow the paradigm that uses an encoder to generate task embeddings and provide this information to a goal-conditioned policy learned via SAC. Different from PEARL, we first redesign the encoder strategy and the training procedure with zero-shot adaptation for applicability to non-stationary environments. Second, we introduce a decoder, which serves as an auxiliary loss that allows the encoder to be trained in an unsupervised manner through the reconstruction of the MDP. This decouples the training of the encoder from the training of the conditioned policy via SAC and avoids the problem of joint training. The architecture of CEMRL is shown in Figure 1. The algorithm is briefly explained as follows.

- During each meta-training iteration, we first collect training data from the different training tasks. Thereby, at each timestep $t$, we feed the recent context into the encoder to identify the current task. The recent context is a set of transitions (state, action, reward, next state) of a specified number of previous timesteps. The transitions are encoded individually and the resulting encodings are fused to get an overall task encoding $\boldsymbol{z}_t$ for the timestep $t$. The task encoding is provided to the conditioned policy, which outputs an action, based on this encoding and the current state.
- During the optimization phase, we first train the encoder and decoder in an unsupervised manner to learn a task representation by reconstructing the underlying MDP. This works by encoding the context from a specific timestep in the collected data and using the decoder to predict the reward and next state based on state, action, and the computed task encoding. Thereafter, we use the trained encoder to label each transition in the replay buffer and attach the computed task encoding to the transition. By doing so, we are able to train the conditioned policy via SAC, where the state is appended with the task encoding, independently from the task representation learning.
- During meta-testing, CEMRL is able to identify and adapt to a task with zero-shot adaptation, as the encoder infers the task on a per-time-step basis, allowing the policy to react to task changes immediately.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 06,2022 at 11:01:49 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3185549



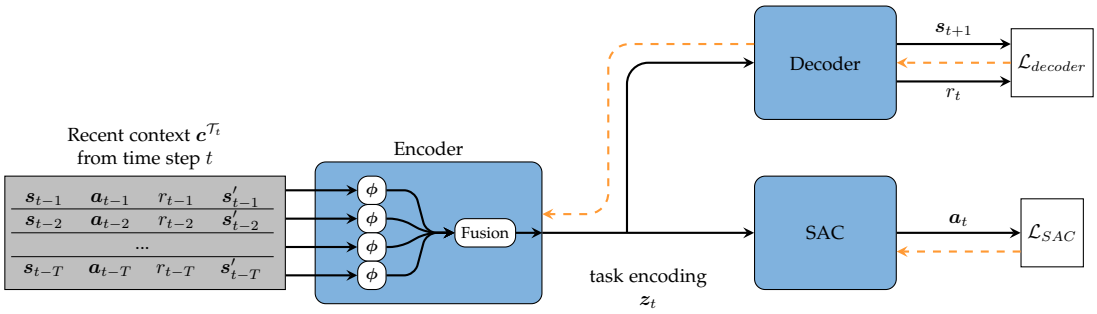



Fig. 1: Components of our algorithm: The encoder learns task encoding for recent context from gradients (orange) of the decoder and provides it as input for SAC.

### 5.2 Online Adaption to Non-Stationary Environments

PEARL is limited to few-shot meta-RL, since the algorithm by design is build on the assumptions of stationary environments: For task identification, transitions from arbitrary previous timesteps are considered, the adaptation procedure is performed on a per-episode basis, and the algorithms utilizes "warm-up" episodes for task identification and adaptation before being able to perform the requested task. Further, data is collected episode-wise from a specific task and stored in individual replay buffers. However, in non-stationary environments the task can change at every timestep and corresponding adaptation is needed at once.

As a naive solution, assuming a setting where is possible in a few-shot setting and zero-shot adaptation is needed only during meta-testing, one could simply retain the PEARL meta-training, and use only the latest transition as context for the encoder and perform task inference every timestep during meta-testing. However, we find that encodings learned in a few-shot adaptation setting during meta-training are not transferable to the zero-shot adaptation setting during meta-testing. This can be explained by a distribution shift between few-shot training and zero-shot testing. During meta-training, transitions from arbitrary situations in the task are used to determine the task, which represents a broad distribution. In contrast, during meta-testing, only few highly related transitions are used to determine the task. Therefore, during meta-testing, the encoder may produce encodings that the policy was not trained for, and thus fail to solve the task.

Based on this observation, we use the following modified training procedure. During both meta-training and meta-testing, at each timestep $t$, the encoder produces task indicators $z_t$, based on the recent context. Thus, during training, the encoder and policy are already explicitly exposed to the context that they will receive during testing. More formally, this method employs the concept of local consistency of the environment and uses only data from timesteps $t-T$ to $t-1$ as sequential context when inferring the task corresponding to timestep $t$. We denote the context corresponding to timestep $t$ as $\boldsymbol{c}^{\mathcal{T}_t} = \{(\boldsymbol{s}_n, \boldsymbol{a}_n, r_n, \boldsymbol{s}'_n)\}_{n=t-T\dots t-1}$ to emphasize that this data resembles only valid context for the current, locally consistent task $\mathcal{T}_t$. The transition of timestep $t$ itself is denoted as $\boldsymbol{\tau}_t = (\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}'_t)$. Akin to PEARL, we encode single transitions $\boldsymbol{c}_n^{\mathcal{T}_t}$ individually and use the resulting encodings as Gaussian factors, to determine the overall encoding $\boldsymbol{z}_t$ for timestep $t$. A proper choice of $T$ should consider two design factors. First, $T$ should be high enough to contain sufficient timesteps to reflect the consistency of one task. Second, $T$ should be set as small to save computation burden and ensure quick reactions to task changes, under the condition of ensuring satisfied accuracy. Empirical results indicate that the performances are robust when $T$ is set in the range of $5-20$.

Using this strategy of performing adaptation at each timestep by inferring the task based on recent context and allowing the environment to be non-stationary during training not only has consequences for the data collection phase during meta-training and the meta-testing, but for the training process as a whole. In non-stationary environments, it is no longer known which task collected data stems, meaning that the training strategy from PEARL of storing transitions in task-specific replay buffers and performing optimization on a task-wise basis cannot be used. Instead, we store the transitions with their corresponding recent context in a common replay buffer, to be able to recover the exact context the agent was exposed to during data collection in the optimization phase. During optimization, we process batches of randomly selected transitions from the replay buffer with their context and compute the task encodings and reconstruction loss for every transition individually.

### 5.3 Task Representation

Normally, broader task distributions have cluster characteristics and can be described as base tasks with specific subtasks. Standard VAEs, like the one used in PEARL, with a single Gaussian as underlying generative model are not intended to represent such clustered task distributions. We propose a more complex generative model that resembles cluster-like task distributions inherently and design an encoder and decoder on its basis.

#### 5.3.1 *Generative Model*

Inspired by Gaussian mixture models (GMMs), we introduce our generative model and derive the evidence lower bound (ELBO) by applying variational inference. Thereafter, we show how to implement the generative model in terms of an encoder and decoder.

A common generative model for distributions with cluster characteristics is GMMs. To point out the connection between standard GMMs and our application to meta-RL, we first revisit the generative process in GMMs. We consider the following sampling process for some observable

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 06,2022 at 11:01:49 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3185549



random variable $\boldsymbol{x}$. First, a latent one-hot cluster indicator $\boldsymbol{y} \sim \mathrm{Cat}(\boldsymbol{\pi})$ is drawn from a categorical distribution, with $\boldsymbol{\pi}$ being the prior probability for each cluster. Each cluster has its own multivariate Gaussian distribution as generator with cluster-specific parameters $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$. The observed data $\boldsymbol{x}$ is drawn from the Gaussian distribution corresponding to the cluster indicator $\boldsymbol{y}$, i.e., $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ if $y_k = 1$.

This concept can be transferred to the meta-RL setting in a similar way. In the meta-RL setting, the observed data are transitions of the agent in the environment $\boldsymbol{x} = (\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}')$ and we assume the following generative process. The task distribution is resembled by a GMM, where the base task is determined by the category and the sub-tasks are determined by the Gaussian distribution. In this setting, the outcome of the GMM is the unobserved task indicator $\boldsymbol{z}$. The observed data $\boldsymbol{x}$ is thought of as being produced by a generator function from the task indicator $\boldsymbol{z}$, like in a standard VAE. This generator is another Gaussian distribution that resembles the characteristics of the environment.

Formalizing this mathematically, we utilize the following generative model:

$$\begin{cases} \boldsymbol{y} \sim p(\boldsymbol{y}) = \mathrm{Cat}(\boldsymbol{\pi}) & \text{Base task distribution} \\ \boldsymbol{z} \sim p(\boldsymbol{z} \mid \boldsymbol{y}) = \mathcal{N}\left(\boldsymbol{\mu}_z(\boldsymbol{y}), \boldsymbol{\sigma}_z^2(\boldsymbol{y})\right) & \text{Sub-task distribution} \\ \boldsymbol{x} \sim p(\boldsymbol{x} \mid \boldsymbol{z}) = \mathcal{N}\left(\boldsymbol{\mu}_x(\boldsymbol{z}), \boldsymbol{\sigma}_x^2(\boldsymbol{z})\right) & \text{Environment model} \end{cases} \tag{6}$$

The joint probability of the generative model factorizes $p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = p(\boldsymbol{x} \mid \boldsymbol{z})\, p(\boldsymbol{z} \mid \boldsymbol{y})\, p(\boldsymbol{y})$. $p(\boldsymbol{y})$ is the distribution over base tasks and $p(\boldsymbol{z} \mid \boldsymbol{y})$ is the distribution over sub-tasks corresponding to base task $\boldsymbol{y}$. Together, these terms constitute the mixture of Gaussians that model a cluster-like task distribution. Finally, $p(\boldsymbol{x} \mid \boldsymbol{z})$ is the generator of observed data $\boldsymbol{x}$ from task $\boldsymbol{z}$. Note that this generative model is an extension to the generative model of VAE, where the latent space is modeled not by a mixture of Gaussians, but a single Gaussian.

Similar to VAEs, we aim to infer the posterior $p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})$, i.e., gain information about the task from the observed data. However, here we not only aim to infer the overall task $\boldsymbol{z}$ but also its corresponding base task $\boldsymbol{y}$. We employ the approximate variational posterior $q(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})$ to approximate the intractable true posterior:

$$q(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}) = q(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y})\, q(\boldsymbol{y} \mid \boldsymbol{x}) \tag{7}$$

Note that, in accordance with VAEs, this variational posterior will be implemented as the encoder, while the decoder will represent the generating function $p(\boldsymbol{x} \mid \boldsymbol{z})$. We follow the variational inference approach employed by VAEs, which formulates the KL-divergence between the true posterior and the posterior approximation to find the ELBO:

$$\begin{aligned} &\mathbb{KL}\left(q(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}) \parallel p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})\right) = \mathbb{E}_{q(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})}\left[\log \frac{q(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})}{p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})}\right] \\ &= \mathbb{E}_{q(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})}\Big[\log q(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}) - \log p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})\Big] \\ &= \mathbb{E}_{q(\boldsymbol{y} \mid \boldsymbol{x})}\big[\mathbb{E}_{q(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y})}\big[-\log p(\boldsymbol{x} \mid \boldsymbol{z})\big] \\ &\qquad + \mathbb{KL}\left(q(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y}) \parallel p(\boldsymbol{z} \mid \boldsymbol{y})\right)\big] \\ &\qquad + \mathbb{KL}\left(q(\boldsymbol{y} \mid \boldsymbol{x}) \parallel p(\boldsymbol{y})\right) + \log p(\boldsymbol{x}) \end{aligned} \tag{8}$$

Solving for $\log p(\boldsymbol{x})$ gives the evidence lower bound $\mathcal{L}_{\text{ELBO}}$:

$$\begin{aligned} \log p(\boldsymbol{x}) &= \mathbb{KL}\left(q(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}) \parallel p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})\right) + \mathcal{L}_{\text{ELBO}} \\ &\geq \mathcal{L}_{\text{ELBO}} = \mathbb{E}_{q(\boldsymbol{y} \mid \boldsymbol{x})}\Big[\mathbb{E}_{q(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y})}\big[\log p(\boldsymbol{x} \mid \boldsymbol{z})\big] \\ &\qquad - \mathbb{KL}(q(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y}) \parallel p(\boldsymbol{z} \mid \boldsymbol{y}))\Big] - \mathbb{KL}(q(\boldsymbol{y} \mid \boldsymbol{x}) \parallel p(\boldsymbol{y})) \end{aligned} \tag{9}$$

The expectation $\mathbb{E}_{q(\boldsymbol{y} \mid \boldsymbol{x})}$ can be computed exactly by marginalizing over $K$ categorical options. The expectation $\mathbb{E}_{q(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y})}$ is intractable and approximated with Monte Carlo sampling $\boldsymbol{z}^{(k)} \sim q(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y} = k)$ from the approximate posterior using the reparametrization trick [28]. In the resulting approximation of the ELBO, we find terms similar to the single-component VAE.

$$\begin{aligned} \mathcal{L}_{\text{ELBO}} \approx \sum_{k=1}^{K} \overbrace{q(\boldsymbol{y} = k \mid \boldsymbol{x})}^{\text{Component posterior}} \Bigg[ & \overbrace{\log p(\boldsymbol{x} \mid \boldsymbol{z}^{(k)})}^{\text{Component-wise reconstruction loss}} \\ & - \alpha_{\text{KL}} \overbrace{\mathbb{KL}\left(q(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y} = k) \parallel p(\boldsymbol{z} \mid \boldsymbol{y} = k)\right)}^{\text{Component-wise regularizer}} \Bigg] \\ & - \beta_{\text{KL}} \underbrace{\mathbb{KL}\big(q(\boldsymbol{y} \mid \boldsymbol{x}) \parallel p(\boldsymbol{y})\big)}_{\text{Categorical regularizer}} \end{aligned} \tag{10}$$

Intuitively, with this loss, the model can either have high entropy over $q(\boldsymbol{y} \mid \boldsymbol{x})$, such that all reconstruction losses and regularizers must be low, or assign high probability to a single base task $k$ and use this specific one to model a datum well. Note that we introduce hyperparameters $\alpha_{\text{KL}}$ and $\beta_{\text{KL}}$ to weight the Gaussian and categorical regularization terms. In short, this generative model has the expressiveness to represent clustered task distributions made up of base tasks and sub-tasks, as the model itself is derived from a cluster distribution.

### 5.4 Encoder

After deriving the generative model and the ELBO, we describe how the encoder is implemented as a neural network, which works as the approximate variational posterior. The encoder part of the generative model comprises the terms $q(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y})$ and $q(\boldsymbol{y} \mid \boldsymbol{x})$ of the approximate variational posterior $q(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})$. Both terms are implemented as individual layers of an overall encoder neural network. In our setting, the network encodes single transitions $\boldsymbol{c}_n^{\mathcal{T}_t}$ from the context $\boldsymbol{c}^{\mathcal{T}_t}$, that resemble the input $\boldsymbol{x}$, and outputs a categorical distribution over base tasks $\boldsymbol{y}$ and a Gaussian distribution over sub-task encodings $\boldsymbol{z}$. Afterwards, the encoded distributions from the single transitions are fused by a multiplication of Gaussian factors to get the information for the overall context. We show the network with its layers and their interplay in Figure 2.

The encoding process for a single transition $\boldsymbol{c}_n^{\mathcal{T}_t}$ works as follows. First, the input $\boldsymbol{x}$ is transformed to a shared representation $\boldsymbol{m}$. For simplicity, this shared layer and the following layers are implemented as MLPs. The base task probabilities $q(\boldsymbol{y} \mid \boldsymbol{x})$ are determined by an MLP with softmax activation, like in standard classification networks. The task-specific Gaussian parameters $\boldsymbol{\mu}_k, \boldsymbol{\sigma}_k$ of $q(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y})$

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 06,2022 at 11:01:49 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3185549



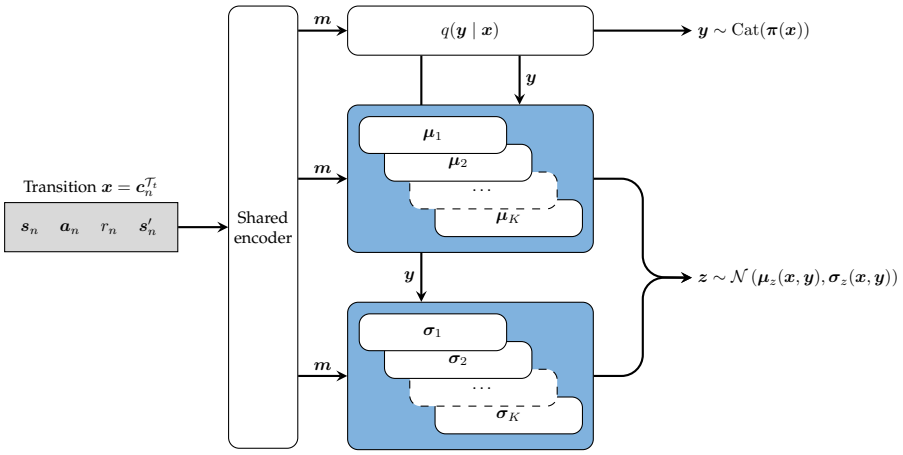



Fig. 2: Encoder: Based on the context of a transition $\boldsymbol{c}_n^{\mathcal{T}_t}$ the class encoder $q(\boldsymbol{y} \mid \boldsymbol{x})$ computes a probabilistic class encoding $y$, component-specific Gaussian encoders $q(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y} = k)$ generate a probabilistic encoding $\boldsymbol{z}$.

are modeled with individual network layers for each base task $k$, conditioned on the shared representation $\boldsymbol{m}$, hence implicitly conditioned on $\boldsymbol{x}$. With our encoder, providing base task and sub-task distributions for each of the base tasks, we gain a new degree of freedom in designing the fusion of information from individual transitions.

In our fusion strategy, we first fuse the base task probabilities of the $N$ timesteps by averaging $q(\boldsymbol{y} \mid \boldsymbol{c}^{\mathcal{T}_t}) = \frac{1}{N} \sum_n q\left(\boldsymbol{y} \mid \boldsymbol{c}_n^{\mathcal{T}_t}\right)$, which again gives a valid probability function, as all $q\left(\boldsymbol{y}_n \mid \boldsymbol{c}_n^{\mathcal{T}_t}\right)$ are already valid probability functions. We then perform $k^* = \arg\max_{\boldsymbol{y}} q(\boldsymbol{y} \mid \boldsymbol{c}^{\mathcal{T}_t})$ on the fused distribution and retrieve the Gaussian factors from one base task for all transitions. With this strategy, the base task is "agreed on" by all transitions, such that all parametric encoding hypotheses are drawn for this single base task. Thus, the focus of the subsequent multiplication is to determine precise parameteric encoding $\boldsymbol{z}$ for this one base task.

Further, we use different sampling strategies from PEARL during optimization and rollouts. During rollouts, we utilize the fusion strategy to get the Gaussian approximate posterior $q(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y})$. From this posterior, instead of sampling, we take the mean of the distribution as task encoding $\boldsymbol{z}$ to provide a more stable task indicator to the SAC. We use the same method for the labeling of the data in the replay buffer. During optimization of the encoder, for $q(\boldsymbol{y} \mid \boldsymbol{x})$, no sampling is needed, as the ELBO demands that a $\boldsymbol{z}^{(k)}$ be sampled for each task $k$ from its task-specific Gaussian $\boldsymbol{z}^{(k)} \sim \mathcal{N}\left(\boldsymbol{\mu}_z(\boldsymbol{y} = k), \boldsymbol{\sigma}_z^2(\boldsymbol{y} = k)\right)$ respectively. The sampling from the Gaussian is performed using the reparametrization trick.

**Priors** We choose a uniform distribution for the class prior $p(\boldsymbol{z})$. To determine the parameters of the component-specific priors $p(\boldsymbol{z} \mid \boldsymbol{y} = k) = \mathcal{N}\left(\boldsymbol{\mu}_z(\boldsymbol{y} = k), \boldsymbol{\sigma}_z^2(\boldsymbol{y} = k)\right)$, we use a linear layer conditioned on a one-hot representation of $\boldsymbol{y}$. This layer receives gradients from the ELBO loss, so that its parameters are optimized jointly with the parameters of the encoder and decoder. Thus, the means and variances of the clusters are parametrized in such a way that only the cluster-like structure is enforced in the prior, but not the actual values for the component specific Gaussian.

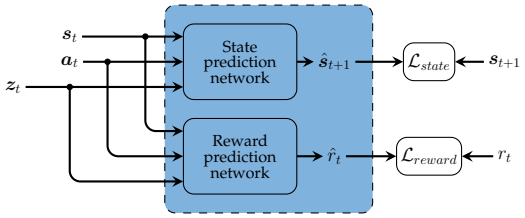



Fig. 3: Decoder: Based on the state $\boldsymbol{s}_t$, the action $\boldsymbol{a}_t$, and the task encoding $\boldsymbol{z}_t$ produced by the encoder from the recent context, the decoder predicts the state transition and rewards for the timestep $t$.

### 5.5 Decoder

In principle, the generative model does not specify how the decoder $p(\hat{\boldsymbol{x}} \mid \boldsymbol{z})$, mapping from the latent space to some reconstruction $\hat{\boldsymbol{x}}$, should be implemented, especially considering that the concrete data and loss function is arbitrary. PEARL uses the Bellman error from SAC as the reconstruction loss. The disadvantage of this is that the Q-function itself converges in the course of the SAC training procedure and thus can only serve as an approximation in the beginning of the training. Further, through this decoding strategy the training of the encoder is inherently coupled with the policy optimization in SAC. This makes it impossible to recover task indicators before SAC training. In this work, we propose a dedicated decoder network that models the transition and reward function conditioned on the task indicator, and thereby implicitly reconstructs the MDP of the task, to learn a task representation in unsupervised manner from experience data decoupled from the policy training.

In standard model-based RL, neural networks are used to learn the transition function $p(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{a}_t)$. In non-stationary meta-RL environments, the transition function $p(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{z}_t)$ is also dependent on the task $\boldsymbol{z}_t$ (the index emphasizes non-stationarity on transition level). Accordingly, we can formulate a reward prediction function, modeled with $p(r_t \mid \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{z}_t)$. By modeling both the transition and the reward function, we reconstruct the complete Markov decision process describing the task at hand from the latent encoding $\boldsymbol{z}_t$. It is only possible to achieve accurate

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 06,2022 at 11:01:49 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3185549



predictions with a proper task indicator. Therefore, the quality of the state and reward predictions are a measure of how well our encoder describes the task using the indicator $\boldsymbol{z}_t$. In the end, we are not interested in the actual predictions, but the emerging encoding $\boldsymbol{z}_t$ is used as the input for SAC.

The encoder predicts the task indicator $\boldsymbol{z}_t$ using context data from timesteps $t-T : t-1$. This task indicator $\boldsymbol{z}_t$ is used to predict state transitions and rewards from timestep $t$, assuming the environment is a locally consistent environment from $t-T$ to $t$. In practice, we use a two-head network with parameter $\boldsymbol{\psi}$ as shown in Figure 3, with one head for state and one head for reward prediction. While the state prediction network $p_{\boldsymbol{\psi}}(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{z}_t)$ and the reward prediction network $p_{\boldsymbol{\psi}}(r_t \mid \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{z}_t)$ are independent MLP networks, they both propagate gradients back into $\boldsymbol{z}_t$.

Then, the log-likelihood term of the decoder from the generative model factorizes into the two networks:

$$\begin{aligned}\log p(\hat{\boldsymbol{x}} \mid \boldsymbol{z}) &= \log p_{\boldsymbol{\psi}}(\boldsymbol{s}_{t+1}, r_t \mid \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{z}_t) \\ &= \log p_{\boldsymbol{\psi}}(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{z}_t) + \log p_{\boldsymbol{\psi}}(r_t \mid \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{z}_t)\end{aligned} \tag{11}$$

Both networks are regression networks, where the data is modeled as a normal distribution $\mathcal{N}\left(f(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{z}_t) \mid \boldsymbol{\sigma}^2\right)$. Thus the loss function is defined as the negative log-likelihood as

$$\mathcal{L}_{decoder} = -\log p(\hat{\boldsymbol{x}} \mid \boldsymbol{z}) = \left[\frac{1}{2}\left\|\hat{\boldsymbol{s}}_{t+1} - \boldsymbol{s}_{t+1}\right\|_2^2 + \frac{1}{2}\left(\hat{r}_t - r_t\right)^2\right] \tag{12}$$

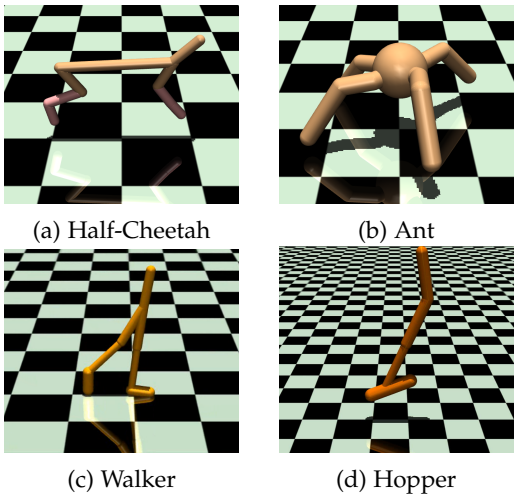
where $\hat{\boldsymbol{s}}_{t+1}$ and $\hat{r}_t$ are network predictions and $\boldsymbol{s}_{t+1}$ and $r_t$ are the true target values from the replay buffer.

### 5.6 Algorithm Overview

The algorithms for the meta-training and meta-testing are summarized in pseudo-code (See Algorithm 1 and See Algorithm 2). The code of CEMRL is available at here[1]. In every training epoch, we collect the data from the training tasks, optimize the task representation in unsupervised manner by encoding recent context $\boldsymbol{c}^{\mathcal{T}_t}$ into a latent encoding $\boldsymbol{z}_t$ and reconstructing the MDP from this encoding and the latest transition, as imposed by the ELBO of our generative model. We label all data in the replay buffer with the optimized encoder to allow us to train the policy with this labeled data from the replay buffer independently with SAC thereafter. Note that instead of using the state as sole input to the policy and Q-function, we use a concatenation of state $\boldsymbol{s}$ and latent encoding $\boldsymbol{z}$ to condition the policy on the task.

## 6 Experiments

As described in previous sections, our proposed method consists of three novel concepts compared to state-of-the-art meta-RL methods: an expressive mixture model encoder, a decoder leveraging MDP reconstruction, and redesigning the encoding and training strategy for application in non-stationary environments. We validate those concepts on different experiments that are designed on the basis of the well-known benchmark [18], taking the Half-Cheetah, Ant, Walker, and Hopper as the agents (See Figure 4). Due to the page limit, we show comprehensive experiment results with the Half-Cheetah and Ant, while the Walker and Hopper are only used to show the consistency of our method on the velocity task with a non-stationary setting. The results on the Walker and Hopper are shown in the ablation study 7.1. First, we apply our method to parametric, stationary environments to compare the performance directly with PEARL. Second, we investigate parametric, non-stationary environments and thereby the applicability of CEMRL to non-stationary environments. In the process, we also test our algorithm in a continual learning setting. Last, we conduct experiments in a non-parametric, stationary environment with three distinctive base tasks to investigate the capabilities of the mixture model encoder.



(a) Half-Cheetah (b) Ant

(c) Walker (d) Hopper

Fig. 4: MuJoCo agents from Gym [18] used in experiments.

To evaluate the efficiency of our algorithm, we use the following measures. First, we plot the average reward during meta testing over the number of collected transitions during training. In doing so, we can evaluate the asymptotic performance and the sample efficiency. Second, we illustrate the connection between the true task specification, e.g., the true goal velocity, and the encoding for $\boldsymbol{y}$ and $\boldsymbol{z}$ produced by the encoder. From this, we compute the mean and standard deviation of the encodings $\boldsymbol{z}$ from all transitions in the replay buffer that were collected from one task during training. Besides, we provide an additional plot for the encoded base task. For each task, we mark the most probable base task (i.e., $\arg\max_k q(\boldsymbol{y} = k \mid \boldsymbol{x})$) over all transitions in the replay buffer corresponding to this task. The encodings are computed each epoch after the task encoding training using the labeler. Third, for the locomotion tasks, we report time responses under test conditions with the trained encoder and policy, to evaluate, if an agent is able to perform its task successfully. Last, videos visualizing the performance of our algorithm can be found at the code's link.

### 6.1 Stationary and Parametric Environments

To compare the performance with PEARL, we apply CEMRL to two stationary and parametric environments from PEARL, namely, *cheetah-stationary-dir*: Half-Cheetah with stationary goal direction and *cheetah-stationary-vel*:

1. https://sites.google.com/view/cemrl

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 06,2022 at 11:01:49 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3185549

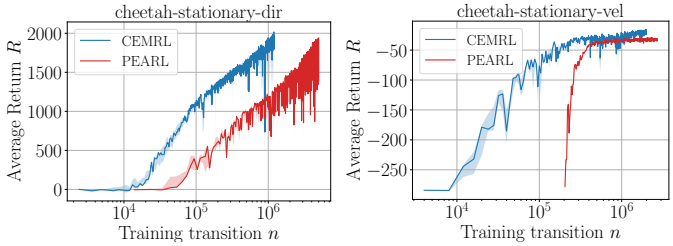


**Algorithm 1** CEMRL: meta-training

**Require:** Batch of training tasks $\{\mathcal{T}_i\}_{i=1\ldots T}$ from $p(\mathcal{T})$
**Require:** Encoder $q_{\boldsymbol{\phi}}(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})$, decoder $p_{\boldsymbol{\psi}}(\hat{\boldsymbol{x}} \mid \boldsymbol{z})$
**Require:** Policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \boldsymbol{s}, \boldsymbol{z})$, critic $Q_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{z})$
Initialize replay buffer $\mathcal{D}$
**while** not done **do**
  **for all** $\mathcal{T}_i$ **do** ▷ Collect data with $\pi_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \boldsymbol{s}, \boldsymbol{z})$ and add to $\mathcal{D}$
    Initialize context $\boldsymbol{c}^{\mathcal{T}} \leftarrow \emptyset$
    Initialize environment $\boldsymbol{s} \sim p(\boldsymbol{s})$
    **for** episode length **do**
      $\boldsymbol{y} \sim q_{\boldsymbol{\phi}}(\boldsymbol{y} \mid \boldsymbol{c}^{\mathcal{T}})$ , $\boldsymbol{z} \sim q_{\phi}(\boldsymbol{z} \mid \boldsymbol{c}^{\mathcal{T}}, \boldsymbol{y})$ , $\boldsymbol{a} \sim \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \boldsymbol{s}, \boldsymbol{z})$ (stochastic, training)
      Apply $\boldsymbol{a}$ to environment, Receive $\boldsymbol{s}' \sim p(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a})$ and $r \sim r(r \mid \boldsymbol{s}, \boldsymbol{a})$
      Update context $\boldsymbol{c}^{\mathcal{T}}$ with transition $\boldsymbol{\tau} = (\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}')$
      Add transition to $\mathcal{D}_{episode}$
    $\mathcal{D} = \mathcal{D} \cup \mathcal{D}_{episode}$
  **return** $\mathcal{D}$
  **Task representation learning** ▷ in practice performed batch-wise
  **for** step in task encoding training steps **do**
    Sample transition $\boldsymbol{\tau}_t = (\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}'_t) \sim \mathcal{D}$ and $\boldsymbol{c}^{\mathcal{T}_t} \sim \mathcal{D}$
    **for** $k = 1, \ldots, K$ **do** ▷ For each encoder base class
      $\boldsymbol{z}_t^{(k)} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{c}^{\mathcal{T}_t}, \boldsymbol{y} = k)$ ▷ Sample $\boldsymbol{z}_t^{(k)}$ from class specific Gaussian
      $p_{\boldsymbol{\psi}}(\boldsymbol{s}_{t+1}, r_t \mid \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{z}_t^{(k)})$ ▷ Decoder forward pass
      $\mathcal{L}_{decoder}^{(k)} = \frac{1}{2} \left\| \hat{\boldsymbol{s}}_{t+1} - \boldsymbol{s}_{t+1} \right\|_2^2 + \frac{1}{2} \left( \hat{r}_t - r_t \right)^2$ ▷ Reconstruction loss
      $\mathcal{L}_{\mathrm{KL}_z}^{(k)} = \mathbb{KL}\left( q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid, \boldsymbol{y} = k) \parallel p(\boldsymbol{z} \mid \boldsymbol{y} = k) \right)$ ▷ Component-wise regularizer
    $\mathcal{L}_{\mathrm{KL}_y} = \mathbb{KL}\left( q_{\boldsymbol{\phi}}(\boldsymbol{y} \mid \boldsymbol{x}) \parallel p(\boldsymbol{y}) \right)$ ▷ Categorical regularizer
    $\mathcal{L}_{\mathrm{ELBO}} = \sum_{k=1}^{K} q_{\boldsymbol{\phi}}(\boldsymbol{y} = k \mid \boldsymbol{c}^{\mathcal{T}_t}) \left[ -\mathcal{L}_{decoder}^{(k)} - \alpha_{\mathrm{KL}} \mathcal{L}_{\mathrm{KL}_z}^{(k)} \right] - \beta_{\mathrm{KL}} \mathcal{L}_{\mathrm{KL}_y}$ ▷ ELBO
    $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \lambda \nabla_{\boldsymbol{\phi}} \mathcal{L}_{\mathrm{ELBO}}(\boldsymbol{\phi})$ ▷ Update encoder parameters
    $\boldsymbol{\psi} \leftarrow \boldsymbol{\psi} - \lambda \nabla_{\boldsymbol{\psi}} \mathcal{L}_{\mathrm{ELBO}}(\boldsymbol{\psi})$ ▷ Update decoder parameters
  **Label data** ▷ using optimized encoder parameters $\boldsymbol{\phi}$
  **for** $t = 1, \ldots, T_{\mathcal{D},\max}$ **do**
    $\boldsymbol{y}_t \sim q_{\boldsymbol{\phi}}(\boldsymbol{y} \mid \boldsymbol{c}^{\mathcal{T}_t})$
    $\boldsymbol{z}_t \sim q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{c}^{\mathcal{T}_t}, \boldsymbol{y}_t)$
    Add $(\boldsymbol{y}_t, \boldsymbol{z}_t)$ to transition data in $\mathcal{D}$: $\boldsymbol{\tau}_t = (\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}'_t, \boldsymbol{y}_t, \boldsymbol{z}_t)$
  **Soft-Actor Critic training independently with labeled data** $\mathcal{D}$
  $\boldsymbol{\theta}_Q \leftarrow \boldsymbol{\theta}_Q - \lambda_Q \nabla_{\boldsymbol{\theta}_Q} J_Q(\boldsymbol{\theta}_Q)$ ▷ Q-function update
  $\boldsymbol{\theta}_\pi \leftarrow \boldsymbol{\theta}_\pi - \lambda_\pi \nabla_{\boldsymbol{\theta}_\pi} J_\pi(\boldsymbol{\theta}_\pi)$ ▷ Policy update
  $\alpha \leftarrow \alpha - \lambda \nabla_\alpha J_\pi(\alpha)$ ▷ Temperature update

**Algorithm 2** CEMRL: meta-testing

**Require:** Task $\mathcal{T}_i$, encoder $q_{\boldsymbol{\phi}}(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})$, policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \boldsymbol{s}, \boldsymbol{z})$
**Require:** context buffer $\boldsymbol{c}^{\mathcal{T}}$, episode buffer $\mathcal{D}_{episode}$
Initialize context $\boldsymbol{c}^{\mathcal{T}} \leftarrow \emptyset$
Initialize environment $\boldsymbol{s} \sim p(\boldsymbol{s})$
**for** episode length **do**
  $\boldsymbol{y} \sim q_{\boldsymbol{\phi}}(\boldsymbol{y} \mid \boldsymbol{c}^{\mathcal{T}})$
  $\boldsymbol{z} \sim q_{\phi}(\boldsymbol{z} \mid \boldsymbol{c}^{\mathcal{T}}, \boldsymbol{y})$
  $\boldsymbol{a} = \pi_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{z})$ (deterministic, testing)
  Apply $\boldsymbol{a}$ to environment, Receive $\boldsymbol{s}' \sim p(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a})$ and $r \sim r(r \mid \boldsymbol{s}, \boldsymbol{a})$
  Update context $\boldsymbol{c}^{\mathcal{T}}$ with transition $\boldsymbol{\tau} = (\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}')$
  Add transition to $\mathcal{D}_{episode}$




(a) Half-Cheetah with stationary goal direction. (b) Half-Cheetah with stationary goal velocity

Fig. 5: Stationary, parametric environments: meta-test performance over collected data during meta-training: CEMRL is more sample-efficient and stable than PEARL on *cheetah-stationary-dir* (5a) and outperforms PEARL in sample efficiency and asymptotic performance on *cheetah-stationary-vel* (5b). Note that the $x$-axis is in log scale.

Half-Cheetah with stationary goal velocity. We run PEARL with the original code and parameters provided from [6]. It should be noted that PEARL achieves reported average rewards in the few-shot manner. During test time, PEARL collects data in the unseen environment, infers the task, and performs the valid and reported test run afterwards. In contrast, our algorithm achieves the reported rewards at first sight in the zero-shot manner.

**Meta-training performance and efficiency:** As shown in Figure 5, our algorithm CEMRL is more sample-efficient and stable than PEARL on *cheetah-stationary-dir* (Figure 5a) and outperforms PEARL in sample efficiency and asymptotic performance on *cheetah-stationary-vel* (Figure 5b). Further, compared to PEARL, we can train our encoder strictly off-

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 06,2022 at 11:01:49 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3185549



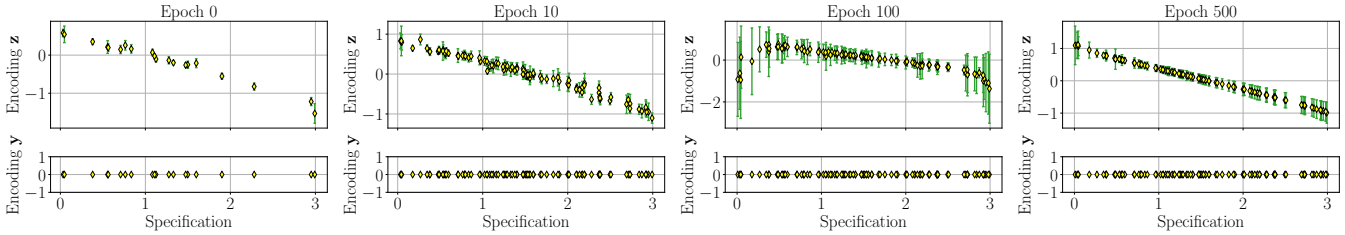



Fig. 6: Task encodings for different training epochs on *cheetah-stationary-vel*.

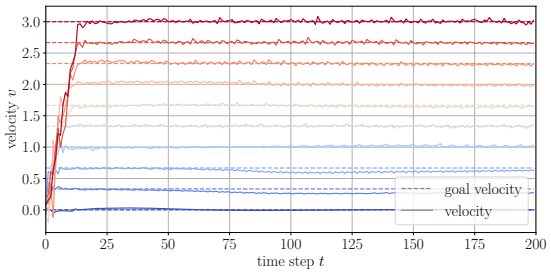



Fig. 7: Half-Cheetah with random, stationary goal velocity: Time responses for selected goal velocities from the task distribution with a well-trained policy.

policy without observing any issues regarding the task distribution shift between meta-training and testing, which enables more sample-efficient learning. We suspect the slightly higher asymptotic performance is provoked by the distinct task representation, which is stable throughout the training process and well structured and precise after convergence.

**Latent Encodings:** We show the learned task representation in different epochs in Figure 6. First we show the learned task representation in different epochs. For this specific experiment, the base task is trivial, as we configure the encoder with one base task only. As early as after the task encoding training phase of the first epoch, a proper representation in form of a linear correlation with low standard deviation between goal velocity and task indicator $\boldsymbol{z}$ is achieved, which indicates high sample efficiency for the encoder. Over time, while more data is collected, the uncertainty for the specific velocities increases slightly, however the linear correlation of the mean values becomes increasingly stable. Overall, in the last training epoch (epoch 500), the encoding is relatively certain and well correlated for all velocities. Only for very low velocities is the encoding uncertain, as motion performance at almost a standstill is hard to execute and infer.

**Time response:** To visualize how the encoding translates to perform the running task at different goal velocities, we plot the time responses of multiple goal velocities from the task distribution and the corresponding velocities achieved in Figure 7. The agent is able to achieve the desired velocity with high precision at nearly all velocities.

## 6.2 Non-Stationary and Parametric Environments

In non-stationary environments, the environment describing MDP can change on a timestep basis. We implement a set of such environments by redesigning common benchmark environments from OpenAI Gym [18]. Parameter changes are induced on certain trigger events, otherwise the MDP is fixed, allowing an agent has the chance to observe and adapt to those changes. We propose the following two strategies for generating trigger events. In the first strategy, referred to in the following as *trigger-by-time*, the environment has an internal counter of transitions the agent performed since the last event. After a specific number of transitions, the MDP is changed. In the second strategy, referred to in the following as *trigger-by-location*, the environment monitors the location of the agent at each transition and triggers parameter changes at certain landmarks, e.g., 10 m apart from the last landmark. In both strategies, the triggers (time or landmark) are randomized for each episode, to prevent the agent from memorizing them.

### 6.2.1 Specific environments

**Non-Stationary Goal Velocity.** These environments are non-stationary variants of the goal velocity environments from MAML [5] and PEARL [6]. The MDP is variable in terms of the reward function. The tasks use *trigger-by-location* as event mechanism during meta-training. Since the goal velocity is a continuous scalar random variable, we configure the latent space as one dimension.

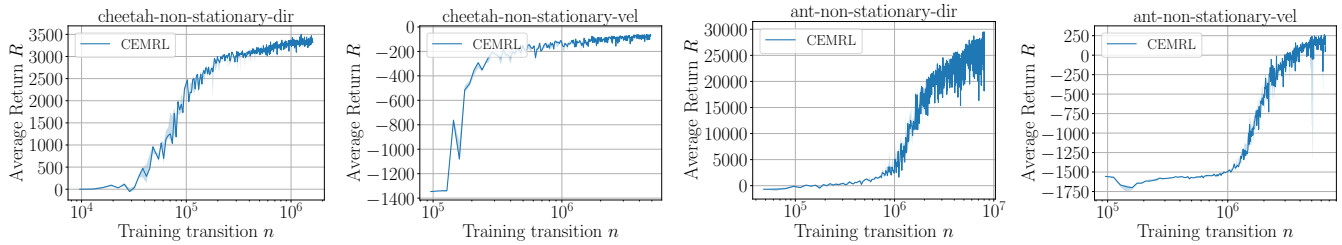
**Non-Stationary Goal Direction.** For further validation on environments with variable rewards, we apply the algorithm to non-stationary variants of the random goal direction environments with Half-Cheetah and Ant (*cheetah-non-stationary-dir* and *ant-non-stationary-dir*). The two goal directions can be interpreted as discrete, qualitatively different tasks. Therefore, we configure the encoder with two base tasks and one latent dimension in this case. We also use *trigger-by-location* as the event mechanism during meta-training.

**Variable Transition Function.** We initially planned to validate our method in non-stationary variants of environments with a variable transition function, like the Walker2D and Hopper with randomized physical parameters, as proposed in PEARL [6]. However, we identify characteristics of such environments, making the adequacy of them as meta-RL benchmark environments questionable. When validating the original benchmark from PEARL, we find that even standard RL algorithms like SAC are able to solve those environments. Similarly, intentionally compromising the encoder learned by PEARL to output noise instead of encoding led to only minor performance loss, and the task was still solved confidently. We hypothesize this is due to the fact that the induced parameter changes, namely variation of mass, friction, and damping of the joints, are reflected at a physical level in the relationship between torque and consequential angular velocity of the joint. As

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 06,2022 at 11:01:49 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3185549






(a) Half-Cheetah with non-stationary direction (b) Half-Cheetah with non-stationary velocity (c) Ant non-stationary direction (d) Ant non-stationary velocity

Fig. 8: Non-stationary, parametric environments: meta-test performance over collected data during meta-training

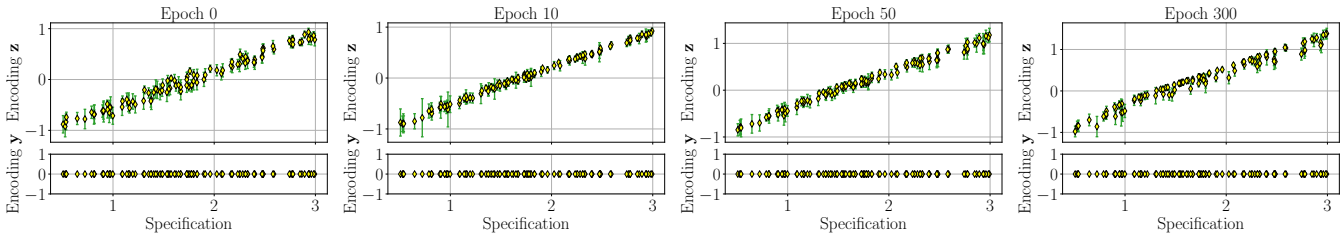
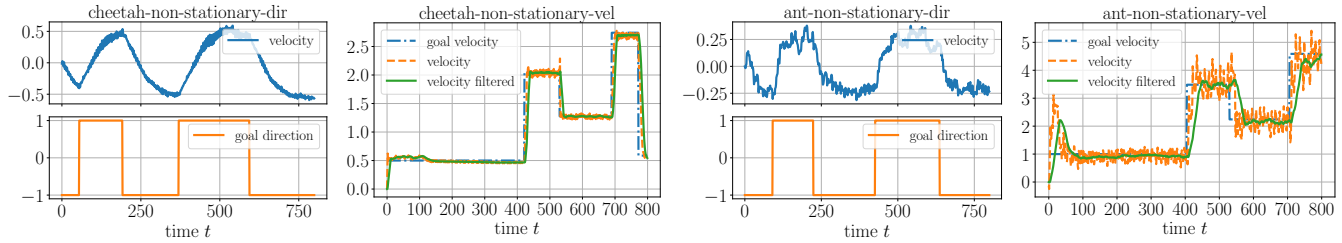



Fig. 9: Environment: *cheetah-non-stationary-vel*, encodings for different training epochs.




(a) Half-Cheetah with non-stationary goal direction (b) Half-Cheetah with non-stationary goal velocity (c) Ant with non-stationary goal direction (d) Ant with non-stationary goal velocity

Fig. 10: Non-stationary, parametric environments: Responses of the agent to MDP changes

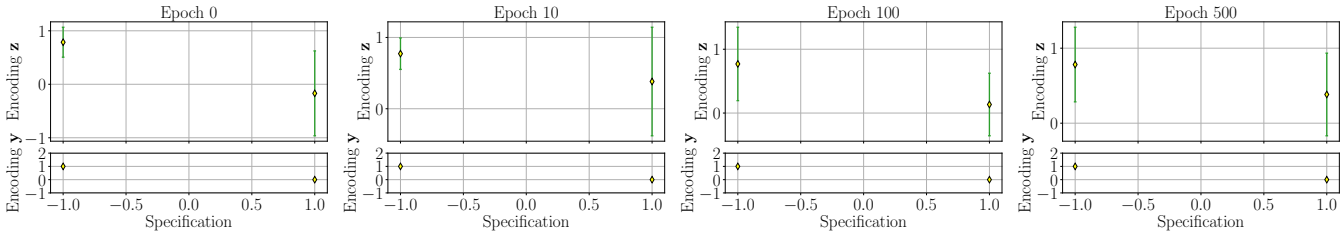



Fig. 11: Environment: *cheetah-non-stationary-dir*, encodings for different training epochs.

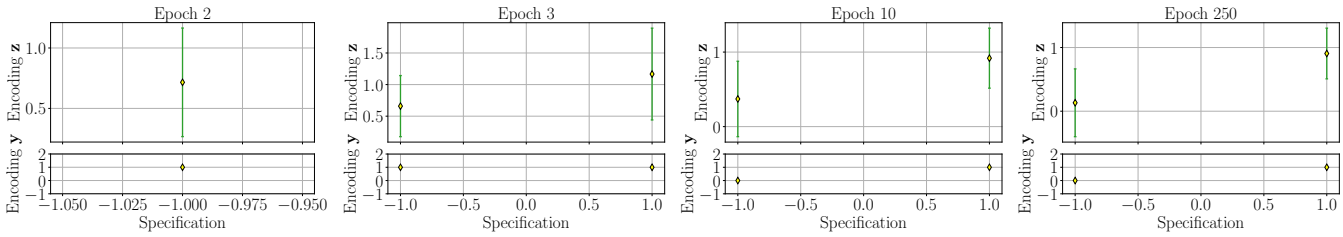



Fig. 12: Encodings Half-cheetah continuous learning

the torque is reflected in the action of the agent and the angular velocity is observed through the state, the agent can learn meaningful actions solely based on the state. In this case, the environment is purely Markovian and reduced to

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 06,2022 at 11:01:49 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3185549



a standard RL problem, with no meta-mechanism required.

#### 6.2.2 Results

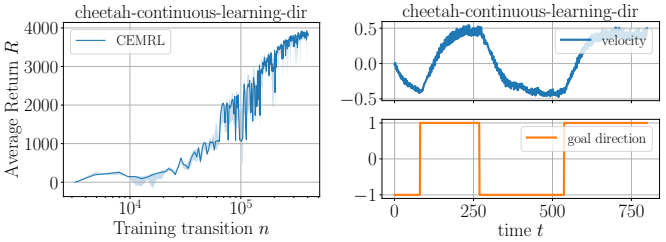
The learning curves for the different experiments in non-stationary environments are shown in Figure 8. These tasks cannot be solved by PEARL due to the non-stationary property, therefore we do not report it as a baseline for these experiments. Of the state-of-the-art algorithms introduced in Section 3, recurrence-based algorithms like $RL^2$ may be able to solve these environments, however they are clearly outperformed by PEARL in asymptotic performance and sample efficiency, as reported in [6]. They are on-policy and require massively more data even in stationary environments.

CEMRL achieves good asymptotic performance with similar sample efficiency to the stationary cases, as shown in Figure 8. For further evaluation of the performance, we analyze *cheetah-non-stationary-vel* in more depth. A visualization of the task representations over different epochs is provided in Figure 9. Similar to the stationary case, after the task encoding training phase of the first epoch, a well-structured representation is learned. During task changes, the encoding uncertainty is increased, as the agent has conflicting data from two tasks in the recent context. Accordingly, the standard deviation of the encodings from all samples in the replay buffer increases. However, the individual tasks can be inferred robustly after some timesteps. This is proven by Figure 10b, where we show a run of the ant and plot the changing goal and actual velocity to illustrate the agent's tracking behavior. The agent infers the goal velocity within few transitions and adapts to it with small steady-state error. This brief delay, paired with some over- and undershooting in the first transitions of a new task are the only imperfections in an otherwise flawless performance. Figure 10d for *ant-non-stationary-vel* shows similar, but less accurate, tracking performance. The agent is still able to identify and adapt to the velocity at hand, yet due to the more complex gait of the ant compared to the cheetah, walking at a precise velocity is harder to realize. Additionally, in Figure 10a we show a similar plot for the *cheetah-non-stationary-dir*. The agent recognizes the change in goal direction nearly instantaneously and adapts its motion. We can see that the *cheetah-non-stationary-dir* is successfully modeled as two base tasks (indicated by the encoding $\boldsymbol{y}$) as shown in Figure 11.

### 6.3 Continuous Learning Environments

We further validate our algorithm in a simple continuous learning environment. To achieve such a setting, we use the Half-Cheetah environment with non-stationary direction, but only provide one task (one direction) at the beginning of the training for data collection. The other task (the other direction) is initially inaccessible. After the agent has learned to run in one direction, it is able to reach a landmark, at which the *trigger-by-location* mechanism changes the goal direction and the agent gains access to the new task.

Figure 13a shows that our algorithm can be trained successfully in such an environment. Figure 13b shows that a good gait with quick adaptation ability is learned for both directions after convergence. In this environment we are especially interested in whether the algorithm exhibits




(a) Meta-test performance over collected data during meta-training

(b) Meta-testing after convergence: Response to direction changes

Fig. 13: Half-Cheetah performing continuous learning in random direction environment

common continuous learning problems, such as overfitting to tasks that are already known, or catastrophic forgetting. This is evaluated in Figure 12, which shows encodings for selected epochs. As the plots indicate, the agent reaches the landmark in the third epoch for the first time and is able to discriminate the new data as a new task immediately. As a result, the encoding of the old task is not impaired, meaning that no catastrophic forgetting occurs. Over the remaining epochs, the two tasks are recognized and modeled as different base tasks (indicated by $\boldsymbol{y}$), and are clearly distinguishable until the end of training.

### 6.4 Stationary, Non-Parametric Environments

Previous experiments evaluated the algorithm's capabilities to efficiently learn and solve parametric environments. With the following experiments, we validate the capability of the mixture model encoder to learn behaviors and task representations in environments with qualitatively distinct base tasks and respective parametric sub-tasks. In these experiments, we are interested in the class encodings $\boldsymbol{y}$ and parametric encodings $\boldsymbol{z}$ produced by the algorithm, as they indicate whether and in what manner our method is able to represent intra-cluster and inter-cluster relationships.

Ideally, a learned encoding should exhibit the following characteristics. First, transitions from the same base tasks are assigned to the same class $\boldsymbol{y}$. Second, the parametric encoding $\boldsymbol{z}$ resembles the intra-cluster relationships between the sub-tasks. Third, inter-cluster boundaries are emphasized by both the class encodings $\boldsymbol{y}$ and parametric encodings $\boldsymbol{z}$. Achieving clear separations between base tasks in terms of the class encoding is especially desirable when this encoder should generate discrete class labels.

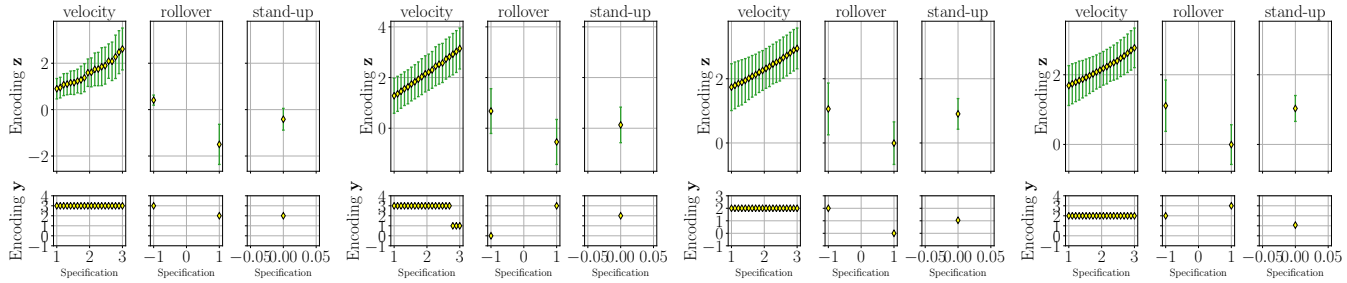
To validate our approach, we propose a novel environment (**cheetah-mixed-tasks**) with qualitatively distinct tasks by aggregating originally standalone tasks for a Half-Cheetah as base tasks. However, designing appropriate task distributions, especially non-parametric ones, comes with challenges, as indicated in [14], for example. It is important for the tasks to share a common structure to allow fast adaptation to new tasks. Conversely, they also require sufficient uniqueness to be distinguishable.

We define three base tasks for the Half-Cheetah: running with a goal velocity, rollovers in forward in backward direction, and standing up (reaching maximal height with the torso), in which the goal velocity and rollover direction

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 06,2022 at 11:01:49 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3185549






(a) Epoch 10 (b) Epoch 150 (c) Epoch 350 (d) Epoch 500

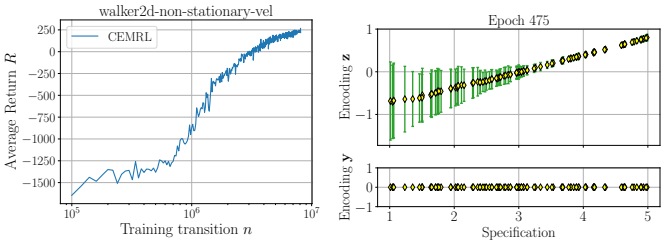
Fig. 14: Encodings for different training epochs on *cheetah-mixed-task*

base tasks have parametric sub-tasks. The reward function for the goal velocity task is the negative absolute value of the difference between the velocity and the goal velocity. The reward for the rollovers is proportional to the angular velocity of the torso of the cheetah. As test tasks we use new goal velocities, but also the same rollover and stand-up tasks as in the training tasks. Therefore, this environment is a slight simplification of the original meta-RL setting, but a challenging environment to evaluate if an algorithm is able to learn a structured task representation and qualitatively different skills.

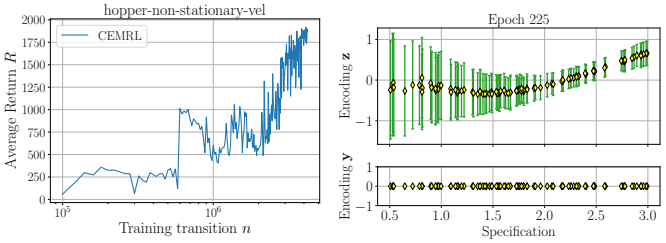
The performance of CEMRL in this environment is visualized in Figure 14. After a few training epochs the algorithm already achieves basic distinct behaviors for the different base tasks. In the videos provided, we show the learned behaviors. Furthermore, in Figure 14 we evaluate the task representation learned by the encoder. Regarding the parametric encodings $\boldsymbol{z}$, the encoder finds an unambiguous representation that, on the one hand, separates base-tasks in the latent space, and on the other hand, structures the sub-tasks of a base task, as shown by the linear correlation between the goal velocity and the encoding. Provided with these encodings, the policy can learn distinct behaviors. Regarding the class encodings $\boldsymbol{y}$, the encoder represents the similarity between the closely related goal velocity sub-tasks in assigning the same class encoding. With the training going on, we can also see the stand-up task is clearly distinguished as one bask task due to its unique dynamics. The forward rollover task (Specification 1) is assigned to one base task, while the backward rollover task (Specification $-1$) is assigned to the velocity task (See Figure 14c and 14d). While those two tasks are disaggregated in epoch 10 in Figure 14b, they are assumed to be one task in other epochs. As shown in the video, we suspect that this is due to their similarity, since both goal velocity and the backward rollover tasks require some forward motion of the torso to fulfill the task, while the forward rollover task requires a backward motion of the torso.

## 7 Ablation Study

In the ablation study, we first show two more experiments from two new agents to show the consistency of CEMRL. We second show the effect of inaccurate estimation of the task embedding.




(a) Reward of the walker-non-stationary-vel task. (b) Task embedding of walker-non-stationary-vel task.




(c) Reward of the hopper-non-stationary-vel task. (d) Task embedding of hopper-non-stationary-vel task.

Fig. 15: MuJoCo agents from Gym [18] used in experiments.

### 7.1 Walker and Hopper

To show the consistency of CEMRL, we also test it with the Walker and Humanoid agent. Figure 15 shows the reward curve and the task embedding from the walker-non-stationary-vel and humanoid-non-stationary-vel environments, which are in line with our main results.

### 7.2 Ablation Study on Noisy Task Inference

To show the effect of inaccurate estimation of $z$, we run ablation study experiments by comparing the following settings in the cheetah-non-stationary-vel environment:

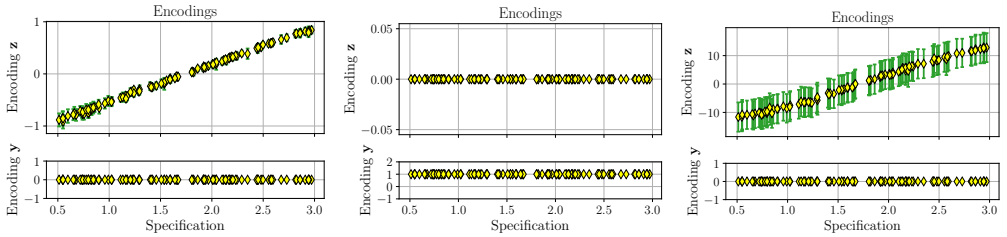
- base: the unmodified CEMRL algorithm
- noise: an additional noise $n \sim N(0, 5^2)$ is added to the encoder's output
- constant: the encoder's output is ignored and replaced by a constant value of $z = 0$

The results are shown in Figure 16. As expected, we find that the predictions of the standard CEMRL yield the most accurate estimation of the task and the goal velocity tracking. The noisy encoding yields decreased performance and has difficulty in adapting to new goal velocities. With constant

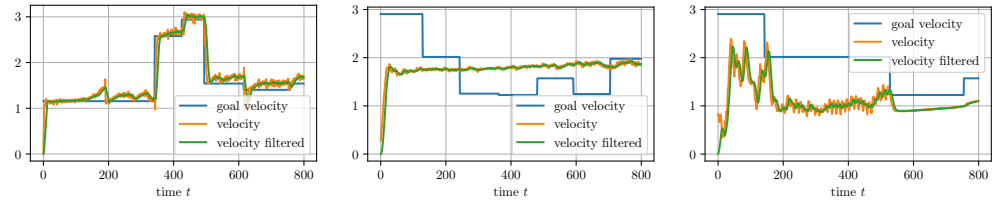
© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 06,2022 at 11:01:49 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3185549






(a) Standard task embedding. (b) Without task embedding. (c) With noisy task embedding.




(d) Standard task embedding. (e) Without task embedding. (f) With noisy task embedding.

Fig. 16: Ablation study of the latent task embedding in cheetah-non-vel task. Figures in the first row shows the task embedding and figures in the second row shows the velocity of the agent.

encoding, the decoder is unable to predict the correct reward whereas the policy cannot distinguish between different goal velocities.

## 8 Discussions

**Applicability to non-stationary environments.** Our experiments show that our algorithm is applicable to non-stationary meta-RL environments. The algorithm learns adequate encodings and good gaits, allowing the agents to perform under non-stationary MDP with zero-shot adaptation, while being stable to be trained. In continual learning problems, where the agent has to learn a set of tasks sequentially, the agent also learns good gaits and encodings without exhibiting common continual learning problems like overfitting to tasks that are already known, or catastrophic forgetting. Furthermore, our encoder establishes a good task representation as early as after the first episode in most scenarios, proving the sample efficiency of the unsupervised task representation learning strategy. With these characteristics, our method fills a gap in the landscape of algorithms that previously were either applicable to non-stationary environments but inefficient, or sample-efficient but only applicable to stationary environments.

**Superior efficiency and competitive performance in stationary environments.** In addition, as shown by the experiments in stationary environments, CEMRL is even more sample-efficient than the state-of-the-art algorithm PEARL and achieves at least competitive asymptotic performance. It is important to note that, compared to PEARL, which needs data collection during testing to perform few-shot adaptation, our performance is achieved in the first trial with zero-shot adaptation. To achieve zero-short adaptation, CEMRL infers tasks at every time step to react at the first sight of a new task. However, from our human's common sense, the task may not change so frequently at every time step, but should be consistent with certain time horizon. Therefore, inferring the task at so fine-grained timestep may lead to unstable task inference due to the potential noise of the context data. In this work, CEMRL adopts a fusion strategy to fuse the base task probabilities of the $N$ timesteps by averaging $q(\boldsymbol{y} \mid \boldsymbol{c}^{\mathcal{T}_t}) = \frac{1}{N}\sum_n q\left(\boldsymbol{y} \mid \boldsymbol{c}_n^{\mathcal{T}_t}\right)$, which can eliminate the effect of noisy data and assure the accuracy of the task inference. Another alternative to achieve online task inference can be recurrent neural networks, which encode the task information from all past states.

**Task representation and behavior learning in non-parametric task distributions.** Gaussian models and Gaussian mixture models have been popular approaches of model approximation in model-based RL algorithms, which aim at providing stochasticity and good uncertainty estimates from limited observed data. However, the Gaussian mixture model in this work is inspired by the parametric and non-parametric variability of meta-RL tasks. As explained in Section 2.2.1, a number of qualitatively distinct tasks can be modeled as $\mathcal{K}$ Gaussian mixtures and each mixture is used to model the parametric distribution of its base task. Therefore, unlike previous methods, the choice of the generative model and the design of the encoder allows our algorithm to learn a clustered task representation in environments that comprise different qualitatively distinct tasks. The latent representation provides a distinction between base tasks, while also resembling parametric relationships between sub-tasks.

## 9 Conclusion

In this work, we have presented CEMRL, an algorithm for efficient meta-reinforcement learning in non-stationary environments and broad task distributions. We showed that with our task representation learning and adaptation strategy, our algorithm is able to learn behaviors in non-stationary environments that need zero-shot adaptation. Our encoder, based on Gaussian mixture models and trained by unsupervised MDP reconstruction, makes it possible to represent complex task distributions. On two locomotion meta-RL benchmarks, our approach achieves superior sample-efficiency and performance compared to

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 06,2022 at 11:01:49 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3185549



the state-of-the-art algorithm PEARL, although the adaptation of CEMRL is performed in zero-shot manner.

## ACKNOWLEDGMENT

This project/research has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreement No. 945539 (Human Brain Project SGA3). This project is also funded by Guangdong International cooperation program with No. 2021A0505030024 and Key-Area Research and Development of Guangdong Province No. 2020B0101650001.

## REFERENCES

[1] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, "Solving Rubik's Cube with a Robot Hand," 2019. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2019arXiv191007113O

[2] J. Schmidhuber, "Evolutionary principles in self-referential learning. (On learning how to learn: The meta hook.)," Ph.D. dissertation, Technical University Munich, 1987. [Online]. Available: http://people.idsia.ch/$\sim$juergen/diploma.html

[3] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "$RL^2$: Fast Reinforcement Learning via Slow Reinforcement Learning," *arXiv e-prints*, 2016. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2016arXiv161102779D

[4] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," *arXiv e-prints*, 2016. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2016arXiv161105763W

[5] C. Finn, P. Abbeel, and S. Levine, "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks," *eprint arXiv:1703.03400*, p. arXiv:1703.03400, 2017. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2017arXiv170303400F

[6] K. Rakelly, A. Zhou, D. Quillen, C. Finn, and S. Levine, "Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables," *arXiv e-prints*, 2019. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2019arXiv190308254R

[7] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *arXiv e-prints*, p. arXiv:1312.6114, 2013. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2013arXiv1312.6114K

[8] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *35th International Conference on Machine Learning, ICML 2018*, vol. 5. International Machine Learning Society (IMLS), jan 2018, pp. 2976–2989. [Online]. Available: http://arxiv.org/abs/1801.01290

[9] H. Ren, A. Garg, and A. Anandkumar, "Context-Based Meta-Reinforcement Learning with Structured Latent Space," p. 5, 2019.

[10] H. Wang, J. Zhou, and X. He, "Learning Context-aware Task Reasoning for Efficient Meta-reinforcement Learning," mar 2020. [Online]. Available: http://arxiv.org/abs/2003.01373

[11] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to Adapt in Dynamic, Real-World Environments Through Meta-Reinforcement Learning," *arXiv e-prints*, p. arXiv:1803.11347, 2018. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2018arXiv180311347N

[12] A. Nagabandi, C. Finn, and S. Levine, "Deep Online Learning via Meta-Learning: Continual Adaptation for Model-Based RL," *arXiv e-prints*, p. arXiv:1812.07671, 2018. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2018arXiv181207671N

[13] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, "A Simple Neural Attentive Meta-Learner," *arXiv e-prints*, 2017. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2017arXiv170703141M

[14] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, "Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning," *arXiv e-prints*, 2019. [Online]. Available: http://arxiv.org/abs/1910.10897

[15] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[16] J. Rothfuss, D. Lee, I. Clavera, T. Asfour, and P. Abbeel, "ProMP: Proximal Meta-Policy Search," 2018. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2018arXiv181006784R

[17] J. Humplik, A. Galashov, L. Hasenclever, P. A. Ortega, Y. Whye Teh, and N. Heess, "Meta reinforcement learning as task inference," 2019. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2019arXiv190506424H

[18] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," jun 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[19] V. Mnih, A. P. Badia, L. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *33rd International Conference on Machine Learning, ICML 2016*, vol. 4. International Machine Learning Society (IMLS), feb 2016, pp. 2850–2869. [Online]. Available: http://arxiv.org/abs/1602.01783

[20] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A Generative Model for Raw Audio," *arXiv e-prints*, 2016. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2016arXiv160903499V

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," *arXiv e-prints*, 2017. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2017arXiv170603762V

[22] 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. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2017arXiv171003641A

[23] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, "Meta-Reinforcement Learning of Structured Exploration Strategies," 2018. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2018arXiv180207245G

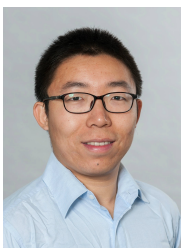
[24] L. Lan, Z. Li, X. Guan, and P. Wang, "Meta reinforcement learning with task embedding and shared policy," in *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2019-Augus. International Joint Conferences on Artificial Intelligence, may 2019, pp. 2794–2800. [Online]. Available: http://arxiv.org/abs/1905.06527

[25] K. Hausman, J. T. Springenberg, Z. Wang, N. Heess, and M. Riedmiller, "Learning an embedding space for transferable robot skills," in *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, feb 2018. [Online]. Available: https://goo.gl/FbvPGB.

[26] M. Igl, L. Zintgraf, T. A. Le, F. Wood, and S. Whiteson, "Deep Variational Reinforcement Learning for POMDPs," 2018. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2018arXiv180602426I

[27] S. Zhang, S. Yan, and X. He, "LatentGNN: Learning efficient non-local relations for visual recognition," in *36th International Conference on Machine Learning, ICML 2019*, vol. 2019-June. International Machine Learning Society (IMLS), may 2019, pp. 12 767–12 776. [Online]. Available: http://arxiv.org/abs/1905.11634

[28] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," *arXiv preprint arXiv:1506.02557*, 2015.



**Zhenshan Bing** received his doctorate degree in Computer Science from the Technical University of Munich, Germany, in 2019. He received his B.S degree in Mechanical Design Manufacturing and Automation from Harbin Institute of Technology, China, in 2013, and his M.Eng degree in Mechanical Engineering in 2015, at the same university. Dr. Bing is currently a post-doctroal researcher with Informatics 6, Technical University of Munich, Munich, Germany. His research investigates the snake-like robot which is controlled by artificial neural networks and its related applications.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 06,2022 at 11:01:49 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Transactions on Pattern Analysis and Machine Intelligence. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2022.3185549



**David Lerch** received this B. Eng degree in Mechatronics Engineering from the Baden-Wuerttemberg Cooperative State University in Stuttgart, Germany, in 2017, and his M.S degree in Robotics, Cognition, Intelligence from the Technical University Munich, Germany in 2021. His research interest is reinforcement learning and agricultural robotics.

**Kai Huang** Kai Huang joined Sun Yat-Sen University as a Professor in 2015. He was appointed as the director of the Institute of Unmanned Systems of School of Data and Computer Science in 2016. He was a senior researcher in the Computer Science Department, the Technical University of Munich, Germany from 2012 to 2015 and a research group leader at fortiss GmbH in Munich, Germany, in 2011. He earned his Ph.D. degree at ETH Zurich, Switzerland, in 2010, his MSc from University of Leiden, the Netherlands, in 2005, and his BSc from Fudan University, China, in 1999. His research interests include techniques for the analysis, design, and optimization of embedded systems, particularly in the automotive and robotic domains. He was awarded the Program of Chinese Global Youth Experts 2014 and was granted the Chinese Government Award for Outstanding Self-Financed Students Abroad 2010. He was the recipient of Best Paper Awards ESTC 2017, ESTIMedia 2013, SAMOS 2009, Best Paper Candidate ROBIO 2017, ESTMedia 2009, and General Chairs' Recognition Award for Interactive Papers in CDC 2009. He has served as a member of the technical committee on Cybernetics for Cyber-Physical Systems of IEEE SMC Society since 2015.

**Alois Knoll** (Senior Member) received his diploma (M.Sc.) degree in Electrical/Communications Engineering from the University of Stuttgart, Germany, in 1985 and his Ph.D. (*summa cum laude*) in Computer Science from Technical University of Berlin, Germany, in 1988. He served on the faculty of the Computer Science department at 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 at 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 at TUM (IMETUM). From 2004 to 2006, he was Executive Director of the Institute of Computer Science at 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.

© 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Technische Universitaet Muenchen. Downloaded on July 06,2022 at 11:01:49 UTC from IEEE Xplore. Restrictions apply.