# TUM Data Innovation Lab

Munich Data Science Institute (MDSI)

Technical University of Munich

&

# TUM Chair of Robotics, AI and Real-time Systems

Final report of project:

# Meta-Reinforcement Learning for Broad Non-Parametric Tasks


| | |
|---|---|
| Authors | Gentrit Fazlija, Sören Glaser-Gallion, Anton Mauz, Michel Osswald |
| Mentor(s) | Dr. rer. nat Zhenshan Bing |
| Project Lead | Dr. Ricardo Acevedo Cabra (MDSI) |
| Supervisor | Prof. Dr. Massimo Fornasier (MDSI) |


Feb 2023

# Abstract

Reinforcement Learning (RL) is a machine learning discipline that focuses on agents solving specific tasks by learning through rewards and penalties. Lately, there have been exciting improvements in this field. However, training new tasks has shown to be sample inefficient. Meta-Reinforcement Learning (Meta-RL) tries to get rid of this impairment by leveraging prior knowledge of different tasks to solve new unseen problems. This approach has shown to be more sample efficient. Although many tasks have natural symmetries underlying, there have only been a few trials of exploiting these equivariances. Latest algorithms tackle the above, however, do not achieve continual and stable performance.

In this thesis, we reproduce Symmetry-Aware Bayes-adaptive Meta-RL (STABLE), a fully equivariant, inference-based Meta-RL algorithm that works for arbitrary finite group symmetries. We enhance its performance and strengthen its convergence towards a stable return. Thus, outperforming earlier work and, subsequently, the well-known baseline PEARL. Further, we look into the impact of different hyperparameters on the overall outcome. We demonstrate the algorithm's ability to deal with changing objectives and its capability of generalizing to out-of-distribution symmetric tasks.

# Contents

# Acronyms

**RL** Reinforcement Learning

**Meta-RL** Meta-Reinforcement Learning

**Multi-RL** Multi-task Reinforcement Learning

**BPTT** Backpropagation Through Time

**PEARL** Probabilistic Embeddings for Actor-Critic RL

**MLP** Multilayer Perceptron

**RNN** Recurrent Neural Network

**CNN** Convolutional Neural Networks

**GMM** Gaussian Mixture Model

**MDP** Markov Decision Process

**AE** Autoencoder

**VAE** Variational Autoencoder

**ELBO** Evidence Lower Bound

**GRU** Gated Recurrent Unit

**SAC** Soft Actor-Critic

**POMDP** Partially Observable Markov Decision Process

**TIGR** Task-Inference-based Meta-RL algorithm using GMM and GRU

**CEMRL** Continuous Environment Meta-RL

**BAMDP** Bayes-Adaptive Markov Decision Processes

**STABLE** Symmetry-Aware Bayes-adaptive Meta-RL

# 1. Introduction

Reinforcement Learning (RL) is one aspect of Machine Learning which got a lot of attention lately. In 2016 the program AlphaGo, powered by RL, made headlines. It was able to beat the leading World Champion in the Chinese strategy game Go. The great complexity of the game made it impossible to beat an experienced player through brute force algorithms. However, through a system of rewards and punishments, the program learned to differentiate between a good and a bad move, thus mimicking the intuition of a human player [1]. More recently, the viral chatbot ChatGPT (2022) drew much attention to the discipline. Its language prediction model excited the world with its human-like interactions. During the training of ChatGPT, a team of trainers asked the language model a question with a correct output in mind. If the model answers incorrectly, the trainers tweak it iteratively to teach it the right answer [2].

A different set of examples lies in the world of robotic control. For instance, the supposedly simple task of opening a door. A human child can learn this with relatively few trials by leveraging prior knowledge of grabbing, pushing and pulling. For a robot arm, on the other hand, this is presented with a major challenge. Traditional concepts rely on learning tasks through a lot of repetition. Even if the robot can already grip, push and pull, opening a door remains a completely new obstacle to him. This requires a lot of meticulous training, which is not desirable in real life due to time constraints, wear and tear and computational complexity [3]. Following this challenge, Meta-RL methods try to infer new unseen tasks (here: opening a door) from a set of related already seen tasks (here: grip, push and pull a door) quickly to increase sample efficiency.

In traditional computer vision a way to increase sample efficiency is the creation of slightly transformed copies of already existing data. If an input image is rotated, the features extracted by the network should also be rotated by the same amount, resulting in the same prediction as before the rotation. For example, classifying an image of a cat. The model is *invariant* if the rotated version of the input image is still classified as the cat. This *equivariance* allows the model to be robust to rotations in the input data and generalize well to new unseen transformations of known data [4]. A transfer of this concept to Meta-RL would help to collect experience for agents much faster and deem hundreds of training iterations unnecessary, hence being sample efficient.

## 1.1. Problem Statement

In this paper, we focus on an Meta-RL algorithm which leverages equivariance in the environment, named Symmetry-Aware Bayes-adaptive meta-reinforcement Learning (STABLE) [5]. The algorithm is *context-based* as it only focuses on the last pieces of training, which shows to deliver better generalization. The method is *inference-based*, as it uses an encoder prior to the policy, to deduce a latent representation of the task and therefore identify the task. This is subsequently propagated to the policy in addition to the current state. The algorithm combines a Gaussian Mixture Model (GMM) and a discrete task

representation. STABLE achieves state-of-the-art results and outperforms conventional algorithms [6, 7]. However, after this peak in performance, the algorithm declines and does not converge. We evaluate the cause of this behavior by identifying a subset of highly influential parameters. We achieve our main goal, namely reproducing the state-of-the-art. Furthermore, by hypertuning these parameters we slightly surpass state-of-the-art performance. We aim to stabilize the performance and achieve approaching convergence within the algorithm.

# 2. Background

## 2.1. Reinforcement Learning

In Supervised learning, a model is trained using labeled data, where the model receives input and the corresponding output. In Unsupervised learning, a model is trained using unlabeled data, where the model finds patterns or structure in the data without explicit inputs and outputs. Reinforcement learning differs from both in that an agent interacts with an environment. He receives feedback on his actions as rewards or penalties and learns to make decisions that maximize the cumulative reward over time using the concept of Markov Decision Process (MDP) [5].

### 2.1.1. The Markov Decision Process

The main learning paradigm in Reinforcement Learning (RL) describes an interaction between the *Environment* and an *Agent* in that *Environment* at each timestep $t \in \mathbb{N} \cup \{\infty\}$. The main objective here is to maximize that reward as much as possible.


Figure 2.1.: *Agent* interaction with *Environment*.

As shown in Figure 2.1, for each $t$, the *Agent* fulfills an action with respect to a policy $\pi$. This action is fed into the *Environment*, giving us the state $s_{t+1}$ and reward $r_t$ associated to that action. This circular process is an instance of a MDP, defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, p_0, r, \gamma)$ [5, 7], where:

| | |
|---|---|
| $\mathcal{S}$ | the space of possible states |
| $\mathcal{A}$ | the space of actions the agent can take |
| $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ | a function with $p\left(s_t, a_t, s_{t+1}\right)$ being the probability density that the next state is $s_{t+1}$, given the current state $s_t$ and the agent's action $a_t$ and $p\left(s_{t+1} \mid a_t, s_t\right)$ |
| $p_0 : \mathcal{S} \to \mathbb{R}$ | a function with $p_0(s)$ denoting the probability density that the initial state is $s$ |
| $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ | a function determining the reward $r(s, a)$ the agent gets for taking action $a$ in state $s$ |
| $\gamma \in [0, 1]$ | the discount factor described later |

It is important to note that the definition of a certain variable or function may vary in different literature. For example, in some sources, the variable $p$ is replaced with a transition function, represented by $\mathcal{S} \times \mathcal{A} \to \mathcal{S}$, that maps states and actions to other states. In this thesis, unless stated otherwise, the notation $s_t$, $r_t$, and $a_t$ refer to the state, action, and reward at time step $t$, while $S_t$, $A_t$, and $R_t$ refer to random variables for the respective values. A single interaction between an agent and the environment, where the agent takes action $a_t$ in state $s_t$, receives reward $r_t$, and reaches a new state $s_{t+1}$, is referred to as a transition and represented by $\tau_t := (s_t, a_t, r_t, s_{t+1})$. For infinite sequences of interactions, we call it trajectory (or paths and denote it by $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$, while finite interactions are called episodes (or rollouts). In this paper, we assume that all trajectories are stored in a replay-buffer $D$ from which we can sample.

The key assumption of a Markov Decision Process (MDP) is the Markov property, which states that the probability distributions of the next states and rewards depend only on the current state and action and not on any previous states or actions. As already stated, the main goal is that the *agent* learns a policy distribution over states and actions respectively $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that yields actions $a_t \sim \pi\left(a_t \mid s_t\right)$ maximizing the return (or gain)

$$G := \sum_{t=0}^{\infty} \gamma^t r_t$$

We define $\pi^*$ as an optimal policy with:

$$\pi^* \in \arg\max_{\pi} \mathbb{E}_{\tau \sim p(\tau|\pi)} \left[ \sum_{t=0}^{\infty} \gamma^t R_t \right]$$

When $\gamma = 1$, the agent aims to maximize the total accumulated reward over the entire time period. When $\gamma < 1$, the agent is able to prioritize rewards that are closer in the near future over those that are further away. The lower the value of $\gamma$, the more emphasis is placed on immediate rewards as opposed to future rewards.

As part of the key elements of an RL model, we further need to define the value function

$$V^{\pi} : \mathcal{S} \to \mathbb{R} : s \mapsto \mathbb{E}_{\tau \sim p(\tau|\pi)} \left[ \sum_{t=0}^{\infty} \gamma^t R_t \mid S_0 = s \right]$$

which denotes the expected gain when starting from a state $s$ and acting according to policy $\pi$. Additionally, we define the action-value function or Q-function

$$Q^{\pi} : \mathcal{S} \times \mathcal{A} \to \mathbb{R} : (s, a) \mapsto \mathbb{E}_{\tau \sim p(\tau|\pi)} \left[ \sum_{t=0}^{\infty} \gamma^t R_t \mid S_0 = s, A_0 = a \right]$$

as the expected gain when starting with state $s$ and taking action $a$ before acting according to policy $\pi$.

**Partially Observable and Belief MDPs** The Partially Observable Markov Decision Process (POMDP) is an extension of the traditional MDP framework that accounts for the uncertainty and incompleteness of the observations made by the agent. In a POMDP, the agent is not always able to fully observe the state of the system, and must make decisions based on a belief state, which is a probability distribution over the set of possible states. The agent's goal is still to find a policy that maximizes some measure of performance, such as expected reward, over time. POMDP are particularly useful in scenarios where the agent has limited sensing capabilities or where the environment is highly dynamic.

### 2.1.2. The Soft Actor-Critic algorithm

The Soft Actor-Critic (SAC) algorithm is an off-policy reinforcement learning method that prioritizes sample efficiency and high performance in continuous control tasks. A task where the agent has to control a continuous set of actions rather than discrete ones to achieve a goal. SAC utilizes a Q-function, referred to as the *soft* Q-function, which allows for the efficient optimization of a stochastic policy by making use of the entropy of the policy. This results in a more robust and stable learning process, as well as an improved exploration of the state space. Additionally, SAC utilizes a separate value function for the policy's entropy, which encourages the exploration of diverse behaviors. Overall, SAC is a sample efficient algorithm that has been shown to achieve high performance in a wide range of continuous control tasks. [8]

**The policy** One of the major challenges in reinforcement learning is balancing exploration and exploitation. Exploration involves taking actions that may not be immediately rewarding in order to discover potentially more valuable behavior in the environment, while exploitation involves taking actions that have previously led to high rewards. While traditional Q-learning methods often use an $\epsilon$-greedy policy to address this challenge, the SAC algorithm provides a more general approach. The SAC algorithm modifies the objective function of RL to optimize both the expected reward and the entropy of the policy's action distribution. Hence, we extend the optimal policy of an MDP; we want to maximize the expected reward and the target entropy $\overline{\mathcal{H}}$ of the policy's action distribution:

$$\mathbb{E}_{a \sim \pi(a|s)} \left[ \alpha_\psi \log \pi_t^* \left( a \mid s; \alpha_t \right) + \alpha_\psi \overline{\mathcal{H}} \right]$$

where:

$$\mathcal{H}(\pi(a \mid s)) = \mathbb{E}_{a \sim \pi(a|s)}[-\log \pi(a \mid s)]$$

The Soft Actor-Critic (SAC) algorithm uses a temperature parameter $\alpha$ to control the balance between exploration and exploitation by determining the relative importance of the entropy term in the objective function. When the value of $\alpha$ is high, the policy becomes

more stochastic, emphasizing exploration. On the other hand, when $\alpha$ approaches zero, the objective function returns to its original form, performing pure exploitation. However, in the original SAC algorithm, the temperature $\alpha$ is a fixed hyperparameter, but it has been found that learning is highly sensitive to it. To address this issue, the method proposed in [9] adapts the learned $\alpha_\psi$, which improves the performance of the algorithm and increases its robustness.

In conclusion, the SAC algorithm's objective function incentivizes learning diverse good strategies, rather than a single deterministic action per state, by maximizing the overall objective using random samples from the replay buffer $\mathcal{D}$. To increase stability, we train two Q-Networks and take their minimum, proposed by [9]:

$$\mathop{\mathbb{E}}_{\substack{s\sim\mathcal{D}\\ a\sim\pi(a|s)}} \left[\min\left\{Q_{\phi_1}(s,a), Q_{\phi_2}(s,a)\right\} + \alpha_\psi \log \pi(a \mid s)\right]$$

**The Q-Networks** Unlike the $Q - functioin$ for the MDP, the objective function of the SAC algorithm not only takes into account the discounted reward from the policy $\pi$ starting with action $a$ in state $s$, but also includes the entropy of the actions taken. This results in a more robust and stable learning process, as well as improved exploration of the state space; the concept that helps us realize this is done by training two Q-Networks to stabilize the training by introducing the concept of lagged versions $Q_{\phi_{1,\text{ targ}}}, Q_{\phi_{2,\text{ targ}}}$ levered during the training objective. Details are in the Appendix 2.1.2 and yield that the updated Q-Networks are trained by minimizing the mean squared error using samples from the replay buffer:

$$\mathop{\mathbb{E}}_{\substack{(s_t,a_t,r_t,s_{t+1},d_t)\sim\mathcal{D}\\ a_{t+1}\sim\pi(a|s_{t+1})}} \left[\frac{1}{2}\left(Q_{\phi_i}(s_t,a_t) - q_{\text{target}}\ (r_t,d_t,s_{t+1},a_{t+1})\right)^2\right]$$

**On- vs. off-policy** *On-policy algorithms* update the policy parameters using data from the same policy that is being evaluated. This means that the policy parameters are updated by data generated by the same policy. *Off-policy algorithms*, on the other hand, update the policy parameters using data generated by a different policy. This allows the policy parameters to be updated by data generated by a different policy, which can help the algorithm learn more efficiently hereby being more sample efficient.

### 2.1.3. Multi-Task Reinforcement Learning

Multi-task Reinforcement Learning (Multi-RL) allows an agent to improve its overall performance by learning multiple tasks simultaneously by sharing information across tasks. For instance, a self-driving car that can navigate on a highway might be able to learn city driving faster, as underlying dynamics such as recognizing traffic lights, signs, and other vehicles remain the same. Thus, Multi-RL improves performance by training a single policy for multiple tasks with a shared structure.

Following this idea, in Multi-RL, the agent's goal is no longer to maximize the gain in a specific MDP as described above, but to maximize the gain for all tasks $\mathcal{D}_{\text{tasks}}$ sampled from a distribution $p(\mathcal{T})$ of MDPs:

$$\mathbb{E}_{\mathcal{T}\sim\mathcal{D}_{\text{tasks}}}\left[\mathbb{E}_{\tau\sim p_{\mathcal{T}}(\tau|\pi)}\left[\sum_{t=0}^{\infty}\gamma^t r_t\right]\right]$$

where we denote the corresponding transition and reward function for a given task $\mathcal{T}$ with $p_{\mathcal{T}}$ and $r_{\mathcal{T}}$ respectively. Here, the policy $\pi(a, s, z_{\mathcal{T}})$ is provided with a task indicator $z_{\mathcal{T}}$ that uniquely identifies the task at hand. This task indicator plays a crucial role in Multi-RL as it allows the agent to adapt to different tasks by sharing information across tasks.

**Parametric vs. Non-parametric variation** The tasks in $p(\mathcal{T})$ can be classified into two main categories. First, *parametric tasks* are those where the number of parameters of the model are fixed and do not change as the amount of data increases. These tasks are mostly the same, with the only difference being some (real-valued) goal specifications. One example includes a robot arm that should pick up the same object at a different position in each task. Secondly, *non-parametric tasks* are those where the tasks are qualitatively distinct and vary by more than just a real-valued goal specification. An example is a robot that has to throw a ball and open a door. A task distribution is considered *broad* if it consists of multiple qualitatively distinct tasks, at least some of which contain parametric variability.

### 2.1.4. Meta-Reinforcement Learning

Meta-Reinforcement Learning (Meta-RL) builds upon the concept of Multi-RL by introducing two distinct sets of tasks, $\mathcal{D}_{\text{test}}$ and $\mathcal{D}_{\text{train}}$, both drawn from $p(\mathcal{T})$. It's important to note that a substantial number of different training tasks are necessary to achieve a level of policy generalization that is considered adequate.

In Meta-RL, the policy no longer receives a task identifier but must determine the current task from the sequence of states, actions, and rewards. This distinction allows for various forms of adaptation, with Meta-RL viewed as learning a Multi-RL policy $\pi(a|s, z)$ and also learning how to infer the task specification $z$ from the set of training tasks $\mathcal{D}_{\text{train}}$ , which requires some form of inference method. The overall objective is now to maximize:

$$\mathbb{E}_{\mathcal{T}\sim\mathcal{D}_{\text{test}}}\left[\mathbb{E}_{\tau\sim p_{\mathcal{T}}(\tau|\pi)}\left[\sum_{t=0}^{\infty}\gamma^t r_t\right]\right]$$

**Zero-, one- vs. few-shot adaptation** The term *zero-shot adaptation* refers to the agent's ability to perform a new task without any prior experience. The agent must infer the task and maximize the expected gain from the first step. In contrast, *few-shot adaptation* allows the agent to experience a limited number of rollouts in a task before being tested and having the ability to identify the task from the transitions collected. A special case of this is *one-shot adaptation*, where the agent only has one episode to identify the task before being tested.

**Stationary vs. non-stationary environments** Until now the task usually stays the same for each episode, which is known as a *stationary environment*. However, it can also happen

that the task changes during one episode, creating a *non-stationary environment*. This requires the agent to adapt on the fly, known as *online adaptation*. Examples include changes in reward or transition function, such as a scenario where the layout of a board game changes mid-episode [10].

**Bayes-Adaptive Markov Decision Processes** As seen in section 2.1.1 of POMDP being an extension of the traditional MDP framework that accounts for uncertainty and incompleteness in the observations made by the agent. A further extension is the Bayes-Adaptive Markov Decision Processes (BAMDP). The main difference between the two is that in POMDP, the agent's observations of the environment are not complete. In contrast, in BAMDP, the environment itself can change over time, leading to different transition and reward functions. In other words, in a POMDP the environment model is uncertain, while in a BAMDP, both the model and the parameters of the environment are uncertain.

Formalizing this idea, BAMDP [11] is a tuple $\left(\mathcal{S}^+, \mathcal{A}, p^+, p_0^+, r^+, \gamma, H^+\right)$, where $\mathcal{S}^+ = \mathcal{S} \times \mathcal{B}$ is referred to as the space of hyper states which combine the state $s_t \in \mathcal{S}$ of a MDP with a belief $b_t \in \mathcal{B}$ over the current underlying MDP.

The belief state, representing uncertainty in the reward and transition functions, can be represented by the distribution $b_t = p\left(p, r \mid \tau_{0:t}\right)$ over possible reward and transition functions, given the agent's past experience. The set of all possible reward functions and transition functions are represented by $\mathcal{R}$ and $\mathcal{P}$, respectively, and can be sampled from any belief $b \in \mathcal{B}$. By updating the new hyperstate $p^+\left(s_{t+1}^+ \mid s_t^+, a_t, r_t\right)$ and reward $r^+\left(s_t^+, a_t, s_{t+1}^+\right)$ with respect to the Appendix A.3 and defining $p^+(\tau \mid \pi)$ analogously to $p(\tau \mid \pi)$ with $\mathcal{S}^+, r^+, p^+$instead of $\mathcal{S}, r, p$, the goal of the policy $\pi\left(a_t \mid s_t^+\right)$is maximizing

$$\mathbb{E}_{\tau \sim p^+(\tau|\pi)}\left[\sum_{t=0}^{H^+-1} \gamma^t r^+\left(s_t^+, a_t, s_{t+1}^+\right)\right]$$

A policy that maximizes this objective is called *Bayes-optimal*. Note that the belief about the current transition and reward function $b_t = p\left(p, r \mid \tau_{0:t}\right)$ change with the agent exploring the environment.

## 2.2. Variational Autoencoder

Vivid readers wonder how task encodings can be represented in a compact and interpretative manner. Variational Autoencoder (VAE) [12] are used in RL to reduce a collection of previous trajectories into a compact task description. They are generally unsupervised machine learning methods that can handle non-linear transformations using neural networks to encode high-dimensional data into a lower-dimensional, latent space. The encoding is performed by an *encoder network*, and the decoded data is generated by a *decoder network* (Figure 2.2). VAEs can be trained off-policy, making them useful in RL to represent the task description and learn a task-conditioned policy.

VAEs assume a generative process for data points and use variational inference to produce a deeper, stochastic encoding. The VAE consists of two parts, the encoder and the decoder. First, we try to encode the data $x$ into a latent space, by $p_\phi(z|x)$. For the generating step, they work by first drawing a latent, unobserved random variable $z$ from a prior distribution $p(z)$, then sampling the observed data from a conditional

distribution $p_\theta(x|z)$ that represents the generating transformation from $z$ to the data (also called decoding). The parameters of the model and the actual values of $z$ are unknown. The goal of using a VAE for dimensionality reduction is to find the optimal compact representation $z$ based on the data, so mathematically speaking $x \sim p_\theta(x \mid z), x \in D$ is a sample of the data (e.g. a picture of a robot) and $z \sim p(z), z \in \mathbb{R}^n$ is the corresponding embedding (e.g. a vector of features like metal type etc.).

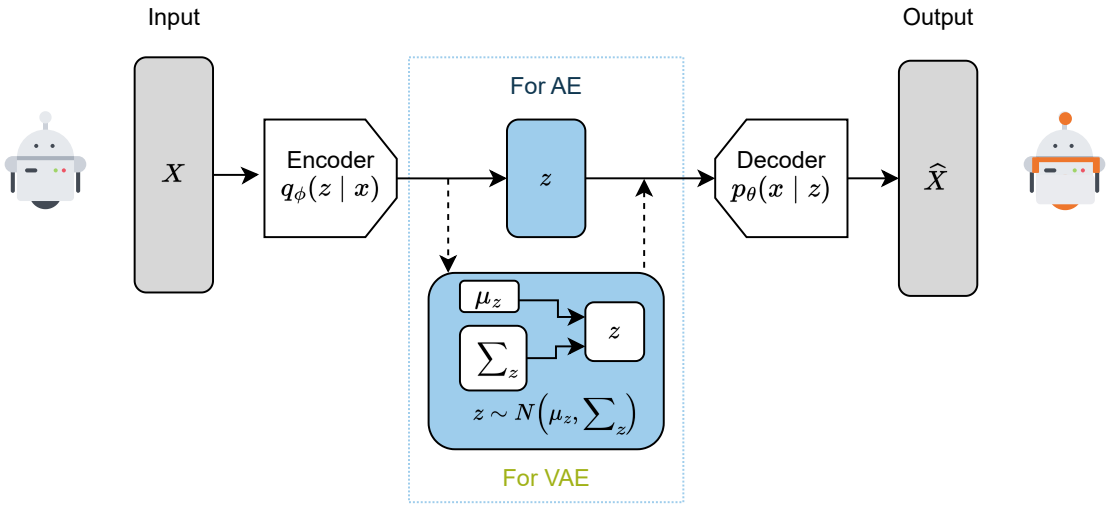



Figure 2.2.: The architecture of a Variational Autoencoder; but in the decoding-step we pick a $z \sim p(z)$ corresponding to a change of metal type

Next to the fact that a VAE is comprised of two parts, it is important to make the jump from a *traditional* Autoencoder (AE) to an *Varational Autoencoder* (see Figure 2.2). This is done by sampling to $z \sim p(z)$. This sampling step makes it impossible to do proper backpropagation for learning, so we use the parametrization trick, which introduces a noise $\epsilon$ sampled from a normal distribution, so that we define $z$ as

$$z = \mathbb{E}\left[q_\theta(z \mid x)\right] + \text{diag}^{-1}\left(\text{Var}\left[q_\theta(z \mid x)\right]\right) \odot \varepsilon \quad \text{where } \varepsilon \sim \mathcal{N}\left(\overrightarrow{0}_n, I_n\right)$$

For more details, see Appendix A.2.

## 2.3. Inference-Based Meta RL algorithm

### 2.3.1. Recurrent-, Gradient- vs. Inference-based Methods

Recurrent-based methods in Meta-RL use a Recurrent Neural Network (RNN) to maintain a *hidden state* throughout the learning process, which captures the agent's past experiences and allows it to adapt to new tasks. These methods are typically trained using a form of unrolling, where the RNNs hidden state is updated at each time step. They are often used for tasks that involve sequential decision-making or have temporal dependencies.

Gradient-based methods, on the other hand, use the gradients of the parameters of the agent's policy with respect to the task's reward. These methods are typically trained using Backpropagation Through Time (BPTT) or a related technique, where the gradients are computed over the entire task episode. They are often used for tasks that have a simple and well-defined structure.

Inference-based methods are based on the idea of learning a model of the task that can be used to infer the agent's latent state aka. task encoding. These methods typically use a VAE or a related model to learn a compact task representation. Inference-based methods have the advantage of being able to handle high-dimensional state spaces, and they are more sample efficient. Additionally, they can also handle non-stationary tasks and are more robust to changes in task distribution [5].

### 2.3.2. PEARL

Probabilistic Embeddings for Actor-Critic RL (PEARL) [6] is developed for fast few-shot meta-RL and sample-efficient training. It is an off-policy algorithm and achieves state-of-the-art asymptotic performance as well as sample efficiency. Figure 2.3 shows the training setup as provided in the paper.

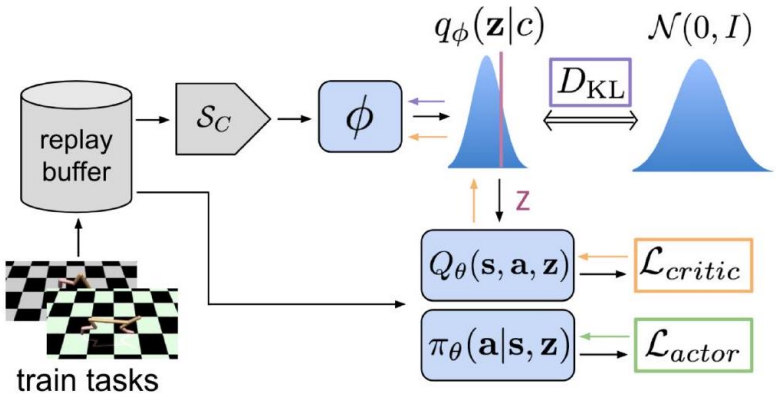



Figure 2.3.: PEARL meta-training procedure [6]

PEARL combines a VAE for task inference with Soft Actor-Critic [8] for policy learning, using task encoding $z$ as the augmented state. The VAE encoder takes *context tuples* $\boldsymbol{c}_n^{\mathcal{T}} = (\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{r}_n, \boldsymbol{s}'_n)$ as input, which includes data for each transition in an MDP, and outputs independent Gaussian factors $\Psi_\phi(\boldsymbol{z} \mid \boldsymbol{c}_n)$ for task encoding. The encoder network is implemented as vanilla Multilayer Perceptron (MLP). To obtain the overall posterior estimate $q_\phi(\boldsymbol{z} \mid \boldsymbol{c}_{1:N})$, the Gaussian factors are multiplied:

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

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 hence writes

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

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.

### 2.3.3. CEMRL

Lerch's Continuous Environment Meta-RL (CEMRL) algorithm [7] builds on PEARL and is suitable for non-stationary environments. The training process is divided into four phases:

1. Experience collection: CEMRL samples from the policy and encoder like PEARL and adds the data to the replay buffer.
2. Encoder training: The encoder is trained by sampling individual transitions $\tau_t^{\mathcal{T}}$ along with a fixed number of previous time steps $\tau_{t-T:t-1}^{\mathcal{T}}$ from the replay buffer. By only considering this recent experience, the algorithm adapts to non-stationary environments.
3. Decoding: Instead of using the critic loss $\mathcal{L}_{\mathrm{Q}}$ from the SAC, two separate decoders are used to predict the next state $s_{t+1}$ and reward $r_t$ from the current state $s_t$, action $a_t$, and encoding $z \sim q_\phi\left(z \mid \tau_{t-T:t-1}^{\mathcal{T}}\right)$. This ensures the encoding has sufficient information about the current task's transition and reward functions. The encoder and decoders are trained jointly using an Evidence Lower Bound (ELBO) similar to the VAE.
4. Network update: The encodings stored in the replay buffer are updated, and the networks of the SAC are trained with data from the buffer before the next epoch begins.

CEMRL enhances the performance on diverse task distributions by using a Gaussian Mixture Model (GMM) for task encoding, instead of a single Gaussian. To achieve this, CEMRL has multiple neural networks in its encoder. For each transition $x = (s_i, a_i, r_i, s_{i+1})$ in the task history $\tau_{t-T:t-1}^{\mathcal{T}}$, it generates a shared encoding $m$ and a categorical random variable $y$ representing the respective cluster through a neural network $y \sim q_\phi(y \mid x)$ with one output neuron per category. Each category then has its own neural network that generates the mean and variance of the Gaussian distribution for the transition $x$. The overall distribution $q\left(z \mid \tau_{t-T:t-1}\right)$ is the combination of Gaussians from each transition, similar to PEARL. Instead of sampling $y$ for each transition separately, the distributions over $y$ can be combined, resulting in one $y$ for all transitions, which is then used to generate the Gaussian distribution $q_\phi(z \mid y, x)$ for each transition $x$.

To calculate the ELBO, the prior $p(z \mid y)$ is necessary. Lerch [7] compares two options for the prior. The first option is to fix it as $p(z \mid y) = \mathcal{N}(y \cdot \overrightarrow{1}_n, \sigma z^2 \cdot \overrightarrow{1}_n)$, where $\overrightarrow{1}_n \in \mathbb{R}^n$ is a vector of ones and $\sigma_z^2$ is a fixed variance hyperparameter. This enforces that the clusters are formed around positions of the $K$ individual clusters. The alternative is to learn these positions through another neural network that takes a one-hot encoding of $y$ as input and outputs the mean and variance of $p(z \mid y)$. This network is trained through the gradients from the ELBO loss.

## 2.4. Equivariance

The real work is full of symmetries; interacting with them is multifaceted. On our robot on a 2D board game, where the optimal action for our agent is to go up, the optimal corresponding action in the same board game rotated by 90 degrees to the right will be

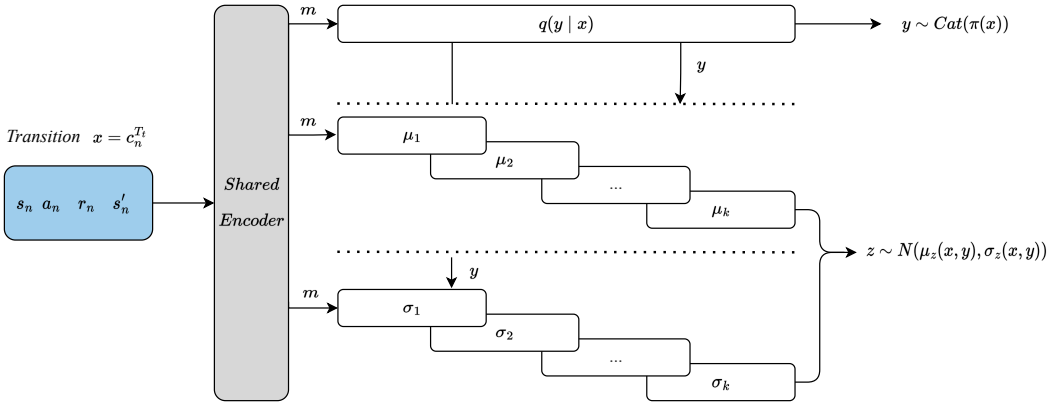



Figure 2.4.: The encoder setup of CEMRL as introduced in the thesis [7].

to go right. This means that the output, such as going to the right, should be the same whether the input is transformed first and then the action is generated, or the action is generated first and then transformed. The transformation itself can be different between the input and output. For instance, the input may be a 2D image, while the output may be one of four discrete actions (moving the image up, right, down or left). While a 90-degree rotation of the image corresponds to a permutation of the actions (up $\rightarrow$ right, right $\rightarrow$ down, down $\rightarrow$ left, left $\rightarrow$ up), they are not the same mappings.

### 2.4.1. Formal Definition

Formally, this can be expressed as follows: Recall, that a mathematical group is defined as a pair $(G, *)$ of a set $G$ and an operation $* : G \times G \rightarrow G$ with the following properties:

$$\begin{aligned}
\forall a, b, c \in G: \quad & (a * b) * c = a * (b * c) && \text{associativity} \\
\exists e \in G, \forall a \in G: \quad & a * e = e * a = a && \text{existence of a neutral element} \\
\forall a \in G, \exists a^{-1} \in G: \quad & a * a^{-1} = a^{-1} * a = e && \text{existence of an inverse element}
\end{aligned}$$

It is easy to observe that the rotation to the right $(\{\text{id, rot 90 , rot180, rot}\,270\}, *)$ with $*$ being the subsequent execution of rotations (e.g. $\text{rot}\,270 * \text{rot}\,180 = \text{rot}\,90$) indeed form a group. In fact, the rotation group defined here is called cyclic, since there exists one element *rot*90 that can generate all others. Given a group $(G, *)$ [13], a mapping $f : \mathcal{X} \rightarrow \mathcal{Y}$ is now called equivariant to a transformation $L_g : \mathcal{X} \rightarrow \mathcal{X}$ if there exists a transformation $K_g : \mathcal{Y} \rightarrow \mathcal{Y}$ with:

$$\forall g \in G, x \in \mathcal{X}: \quad K_g[f(x)] = f\left(L_g[x]\right)$$

In the special case where $K_g[y]$ is the identity function $\left(\forall g \in G : L_g[y] = y\right)$, $f$ is called invariant to $L_g$. Intuitively, this means that applying a transformation $L_g$ to the input does not change the output. In 2.5, the robot (our agent) wants to hover to this optimal action of the fullest battery (grey arrow). If we rotate the whole board game (aka. rotation of state), so the optimal action is rotated. Another commonly used example would be Convolutional Neural Networks (CNN) that detects whether an object is contained in the image or not. It does not matter where e.g. a cat is in the image. Thus, the output should be invariant to shifts in the image.

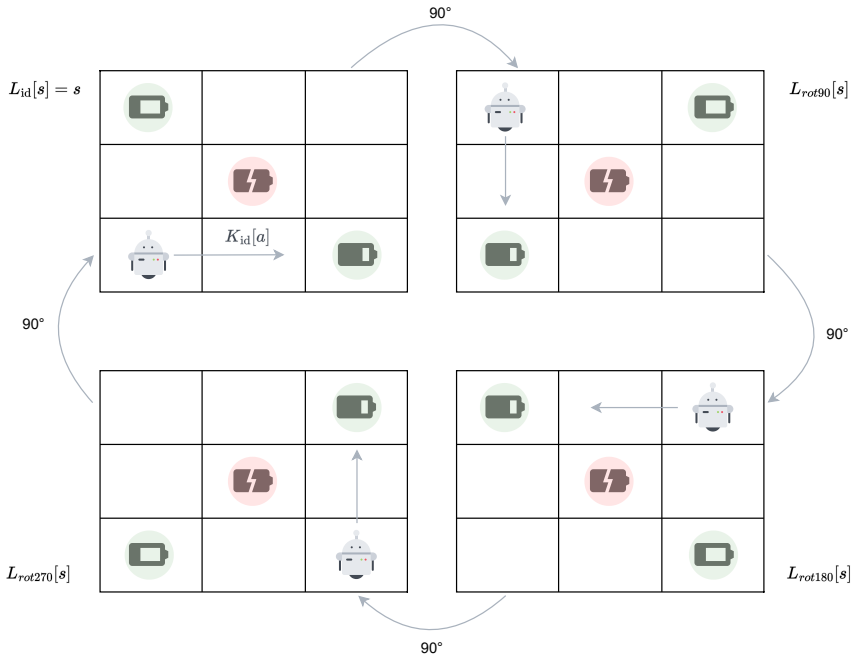



Figure 2.5.: Equivariant transformations in a board game environment.

### 2.4.2. MDP Homomorphic Networks

Pol et al. [13] propose an algorithm to exploit equivariance in reinforcement learning. They define a MDP with symmetries as a MDP $(\mathcal{S}, \mathcal{A}, p, p_0, r, \gamma, H)$ where a group $(G, *)$ exists along with transformations $L_g : \mathcal{S} \to \mathcal{S}, K_g^s : \mathcal{A} \to \mathcal{A}$ such that:

$$\begin{aligned} &\forall g \in G, s \in \mathcal{S}, a \in \mathcal{A} : && r(s,a) = r\left(L_g[s], K_g^s[a]\right) \\ &\forall g \in G, s, s' \in \mathcal{S}, a \in \mathcal{A} : && p\left(s' \mid s, a\right) = p\left(L_g\left[s'\right] \mid L_g[s], K_g^s[a]\right) \end{aligned}$$

Assuming a finite group $G$ and the same action transformation $K_g$ for all states, they then design an equivariant layer: $\mathbb{R}^{D_{\text{in}}} \to \mathbb{R}^{D_{\text{out}}} : y \mapsto Wy + b$ with $W \in \mathbb{R}^{D_{\text{out}} \times D_{\text{in}}}, b \in \mathbb{R}^{D_{\text{out}}}$. For convenience, the bias is moved into the weights by adding it as a column vector of $W$ and appending a 1 to $y$, which yields:

$$\mathbb{R}^{D_{\text{in}}+1} \to \mathbb{R}^{D_{\text{out}}} : y \mapsto Wy \qquad \text{where } W \in \mathbb{R}^{D_{\text{out}} \times D_{\text{in}}+1}$$

Assuming the transformations $L_g, K_g$ to be linear $(\forall g \in G)$, we get

$$\forall g \in G, y \in \mathbb{R}^{D_{\text{in}}+1} : \quad K_g W y \stackrel{!}{=} W L_g y$$

Since $y$ is independent the space of all *equivariance-preserving* weight matrices depics:

$$\mathcal{W} := \left\{ W \in \mathcal{W}_{\text{total}} \mid \forall g \in G : K_g W = W L_g \right\}$$

where $\mathcal{W}_{\text{total}} \subseteq \mathbb{R}^{D_{\text{out}} \times D_{\text{in}}+1}$ denotes the space of all possible weights. This might be $\mathbb{R}^{D_{\text{out}} \times D_{\text{in}}+1}$ to obtain a fully connected layer but also a subset like the weight space of a CNN, thus allowing to build further equivariances on top of the network structure like CNNs.

# 3. Methodology

The chapter overviews the proposed methods, which extend Symmetry-Aware Bayes-adaptive Meta-RL STABLE [5]. First, we show how the algorithm achieves equivariance in the generative model, encoder, prior and decoder. Then we illustrate the actions with respect to the policy taken. Lastly, we dive into the Hyperparameter adjustments conducted.

## 3.1. Algorithm Overview

The algorithm displayed in Figure 3.1 involves an encoder neural network, which takes the recent trajectory as input and generates a distribution over encodings. Two separate networks that make up the decoder, utilize these encodings to predict the next state and reward given the encoding, current state, and action.

The Gated Recurrent Unit (GRU) encoder utilized in the algorithm takes the entire collected experience in the current episode, enabling it to learn Bayesian optimal behavior in arbitrary tasks. To enhance stability in stationary environments, the algorithm reconstructs the next states and rewards using the latent variable from one time step for each of the $H^+$ time steps in the current task [14]. The components are made equivariant with automatically derived, equivariant layers as proposed by [13], allowing the algorithm to be robust and stable even in non-stationary environments.

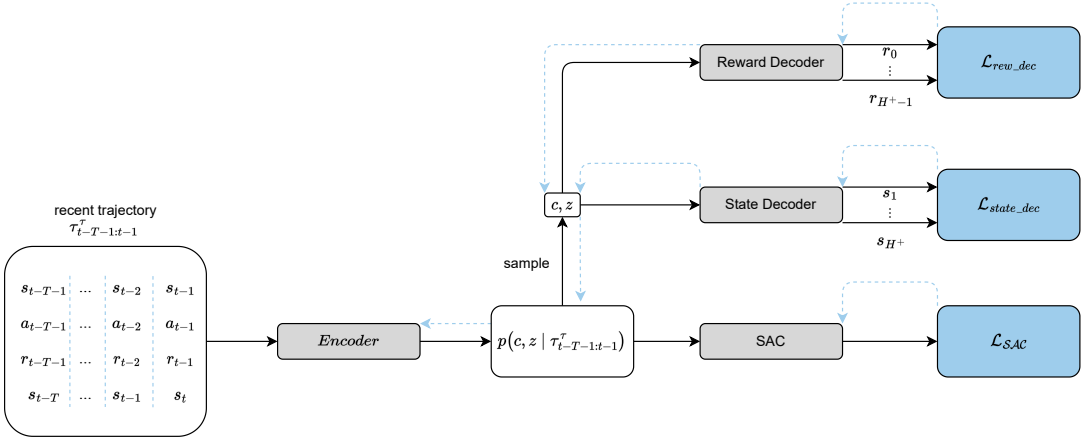



Figure 3.1.: Components of the algorithm. Depicted is the flow of a context window to the Decoders 3.5) and SAC 3.6. Dashed arrows denote gradients.

The training procedure of the algorithm involves four repeating steps. First, the current policy and encoder are utilized to gather data in randomly sampled tasks, which are then added to the replay buffer. Second, the encoder and decoder are jointly trained with data from the replay buffer through a VAE framework. Third, the encodings in the replay buffer are updated using the current encoder, by this process enhancing the encoding

and decoding process. Finally, the SAC networks are updated with data from the replay buffer. The algorithm uses a fixed time frame of experience, $\tau^{\mathcal{T}}_{t-T-1:t-1}$, to learn optimal behavior in environments with dense reward, handling non-stationarity.

## 3.2. Generative Model

The algorithm models the distribution $p\left(z \mid c, \tau^{\mathcal{T}}_{t-T-1:t-1}\right)$ as a GMM. To identify qualitatively distinct tasks, the algorithm introduces the additional categorical latent variable $c$. This incorporates the key assumption that reward and transition functions are equivariant to $z$. Crucially, $c$ is observed by the decoders and thus is part of the latent representation, whereas $k$ solely serves as an auxiliary variable to model $z$ as a mixture of Gaussians. In this case, using a GMM for $z$ improves the capability to represent uncertainty over different specifications of the same qualitatively distinct task.

For instance, an environment with sparse rewards where the agent should navigate to a goal that is either on the right or the left-hand side from the starting position, but never close to the starting position itself. A single, normally distributed random variable cannot model the two zones, whereas a mixture of two Gaussians could lead to a better approximation. In general, the algorithm relies on the fact that the mixture of a sufficient amount of (multivariate) normal distributions with a diagonal covariance matrix can approximate any distribution [15, 16]. We assume the following generative model:

$$
\begin{aligned}
c &\sim p(c) = \text{Unif}(\{0, \dots, C-1\}) && \text{qualitative task} \\
k &\sim p(k \mid c) = \text{Unif}(\{0, \dots, K-1\}) && \text{cluster} \\
z &\sim p(z \mid c, k) = \mathcal{N}(\mathbb{E}[p(z \mid c, k)], \text{Var}[p(z \mid c, k)]) && \text{task parameter} \\
r, s' &\sim p\left(r, s' \mid c, z, s, a\right) && \text{environment interaction}
\end{aligned}
$$

where $p$ denotes the underlying probability density and is unrelated to the transition function of a MDP.

Analogously to section 2.2, the algorithm uses a decoder neural network $p_\theta(x \mid c, z)$ along with an encoder neural network $q_\phi(c, z \mid x)$. Given a single time step $x := (s, a, r, s')$, the ELBO derives as follows (Full derivation in Appendix A.4)

$$
\begin{aligned}
& \mathbb{KL}(q(c, k, z \mid x) \| p(c, k, z \mid x)) \\
\Leftrightarrow \log p(x) =& \mathbb{KL}(q(c, k, z \mid x) \| p(c, k, z \mid x)) + \mathbb{E}_{c,k \sim q(c,k \mid x)} \Big[ \mathbb{E}_{z \sim q(z \mid x,c,k)} [\log p(x \mid c, z)] \\
& - \mathbb{KL}(q(z \mid x, c, k) \| p(z \mid c, k))] - \mathbb{KL}(q(c, k \mid x) \| p(c, k))
\end{aligned}
$$

As $q(c)$ and $q(k \mid c)$ are discrete probability density functions, so is $q(c, k \mid x) = q(c \mid x) q(k \mid x, c)$. Consequently, the algorithm computes the outer expectation explicitly, whereas the inner is approximated expectation using a single Monte Carlo sample $z^{(c,k)} \sim q(z \mid x, c = c, k = k)$. Adding hyperparameters $\alpha_{\text{KL}}, \beta_{\text{KL}}$ for scaling like [17], the algorithm arrives at the ELBO loss:

$$\begin{aligned}
&\mathbb{E}_{c,k\sim q(c,k|x)}\left[\mathbb{E}_{z\sim q(z|x,c,k)}[\log p(x \mid c,z)] - \mathbb{KL}(q(z \mid x,c,k)\|p(z \mid c,k))\right] \\
&- \mathbb{KL}(q(c,k \mid x)\|p(c,k)) \\
&\approx \sum_{c=0}^{C-1}\sum_{k=0}^{K-1} q(c=c,k=k \mid x)\overbrace{[\log p\left(x \mid c,z^{(c,k)}\right)}^{\text{I}} \\
&- \alpha_{\text{KL}}\overbrace{\mathbb{KL}(q(z \mid x,c=c,k=k\|p(z \mid c=c,k=k))]}^{\text{II}} - \beta_{\text{KL}}\underbrace{\mathbb{KL}(q(c,k \mid x)\|p(c,k))}_{\text{III}}
\end{aligned}$$

The goal of maximizing (I) is to improve the accuracy of a model in correctly classifying inputs into their respective categories. Minimizing (II) ensures that training points are not arbitrarily assigned to values in $z \in \mathbb{R}^n$ (see section 2.2). By minimizing (III), the model is prevented from relying solely on one cluster $k$ or channel $c$. The utilization of $c$ and $k$ facilitates information transfer. To distinguish tasks with identical $k$ and $c$, it is necessary for the encoding $p(z \mid x,c,k)$ to deviate from the prior $p(z \mid c,k)$, which will result in an increase in loss (II). Tasks with different z can still be differentiated even when the priors are the same, as long as the decoders take $c$ into consideration.

## 3.3. The Encoder

The algorithm models $q_\phi(z,c \mid x) = q_\phi(z \mid x,c,k)q_\phi(c,k \mid x)$ as a neural network with parameters $\phi$ as depicted in Figure 3.2. To reconstruct a goal position from a dense reward, at least two state-reward pairs are necessary for a 1D setting and three for 2D. In line with the methodology employed in Task-Inference-based Meta-RL algorithm using GMM and GRU (TIGR) [18, 19], the algorithm encodes the recent trajectory $\tau^{\mathcal{T}}_{t-T-1:t-1}$ as a whole using a GRU. To make this setup equivariant, the input transformation is used:

$$I_g : (s,a,r,s') \mapsto \left(L_g[s], K_g[a], r, L_g\left[s'\right]\right)$$

for each time step of the GRU. The shared encoding thus is permuted as described in subsection 2.4.2.

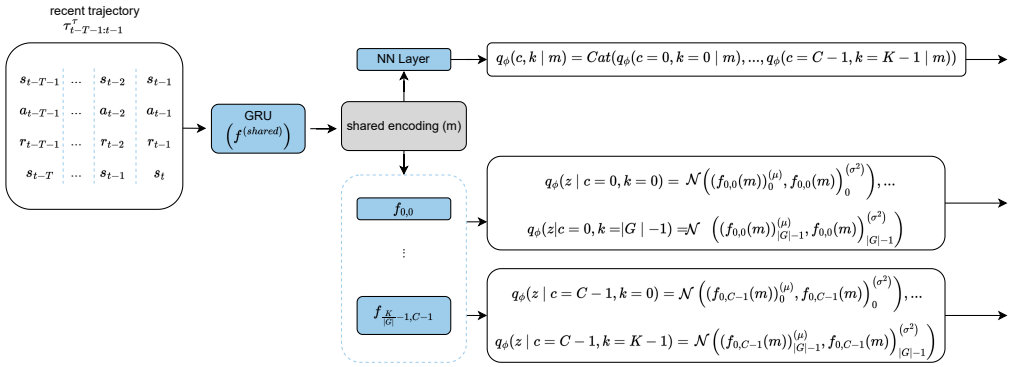



Figure 3.2.: Architecture of our encoder.

To ensure that the predicted distribution over qualitatively distinct tasks remains unchanged in the transformed setting, the number of clusters (denoted by $K$) must be divisible by the group size (denoted by $|G|$). The clusters are then grouped in a specific manner, such that clusters 0 to $|G|-1$ form one group, $|G|$ to $2|G|-1$ form another group, and so on. The distribution over the parametric specifications $z \in \mathbb{R}^{n_t}$ needs to be transformed according to the function $J_g$.

In the experiments, we focus on cyclic groups (see subsection 2.4.1) where $n(g)$ defines as how often a fixed generating element needs to be repeated to get to $g$ (with 0 for the neutral element). An example would be $n(\text{ id }) = 0, n(\text{rot}\,90) = 1, n(\text{rot}\,180) = 2, n(\text{rot}\,270) = 3$ (see section 2.4). Thus, the equation for the output transformation is:

$$\begin{aligned} & q_\phi \left( c = i, k = j \mid \tau^{\mathcal{T}}_{t-T-1:t-1} \right) \\ & \quad = q_\phi \left( c = i, k = \left\lfloor \frac{j}{|G|} \right\rfloor |G| + ((j + n(g)) \mod |G|) \mid I_g^{(\text{traj})} \left[ \tau^{\mathcal{T}}_{t-T-1:t-1} \right] \right) \end{aligned}$$

For instance, in Figure 3.3 the cluster k = 0 is in the upper half. By the equation above, it is ensured that after rotating right by 90°, the cluster $k = 1$ has the probability that $k = 0$ had before [5]. The probabilities of the classes 0 to $|G| - 1 = 3$ rotate as well. To make sure that the distribution over $z$ rotates accordingly the neural network $f_{j,i}$ is used for each channel $i \in \{0, \ldots, C-1\}$ and each set of related classes $j \in \left\{0, \ldots, \frac{K}{|G|}\right\}$. The network outputs the means $\left(f_{j,i}(\cdot)\right)_l^{(\mu)}$ and variances $\left(f_{j,i}(\cdot)\right)_l^{(\sigma^2)}$ of each class $j|G| + l$ for $l \in \{0, \ldots, |G|-1\}$ and ensures that the distribution over $z$ remains consistent after a rotation.

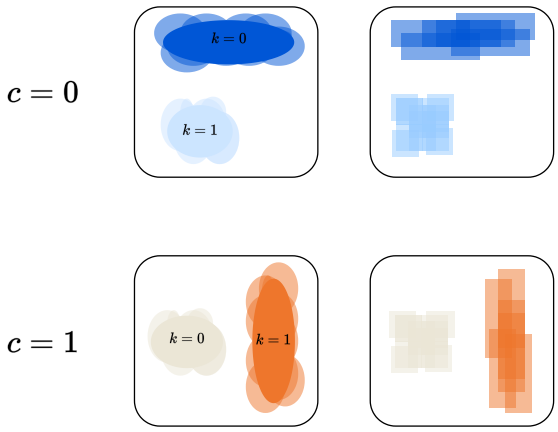



Figure 3.3.: An example of discretizing a 2D latent space.

The goal is to maintain the integrity of the distribution over $z$ following a rotation, such as a 90° rotation in this case. To accomplish this, we use a neural network $f_{j,i}$ for each channel $i \in \{0, \ldots, C-1\}$ and set of related classes $j \in \left\{0, \ldots, \frac{K}{|G|}\right\}$ [5]. The network calculates the means $\left(f_{j,i}(\cdot)\right)_l^{(\mu)}$ and variances $\left(f_{j,i}(\cdot)\right)_l^{(\sigma^2)}$ of each class $j|G| + l$ for $l \in \{0, \ldots, |G|-1\}$ and ensures that the distribution over $z$ remains consistent after a rotation. All of these neural networks utilize the equivariance:

$$f_{j,i}\left(f^{(\text{shared})}\left(I_g\left[\tau^{\mathcal{T}}_{t-T-1:t-1}\right]\right)\right)_l^{(\sigma^2)} = \left| J_g\left[ f_{j,i}\left(f^{(\text{shared})}\left(\tau^{\mathcal{T}}_{t-T-1:t-1}\right)\right)^{(\sigma^2)}_{((l-n(g)) \bmod |G|)} \right] \right|$$

where $f^{(\text{shared})}$ denotes the shared encoder and $|\cdot|$ denotes the element-wise absolute value.

## 3.4. The Prior

To achieve equivariance in $p(z \mid c, k)$ earlier literature ensures [5]:

$$p(z \mid c, k = j) = J_{n^{-1}(j \bmod |G|)} \left[ p\left( z \mid c, k = \left\lfloor \frac{j}{|G|} \right\rfloor |G| \right) \right]$$

As we assumed a normal distribution for $p(z \mid c, k)$ this is accomplished by only choosing $p(z \mid c, k)$ for $k \in \{0, |G|, 2|G|, \ldots, |K|\}$ and defining:

$$\begin{aligned} \forall i \in \{1, \ldots |G| - 1\}, j \in \left\{0, \ldots, \frac{K}{|G|} - 1\right\} : \\ \mathbb{E}[p(z|c, k = j|G \mid +i)] := J_{n^{-1}(i)}[\mathbb{E}[p(z \mid c, k = j)]] \\ \text{Var}[p(z|c, k = j|G \mid +i)] := \left| J_{n^{-1}(i)}[\text{Var}[p(z \mid c, k = j)]] \right| \end{aligned}$$

where $J_g$ is applied along the diagonal in the last Equation (recall that all other entries of the covariance matrix are zero). $|\cdot|$ is used to denote the element-wise absolute value. Means and variances of the remaining priors $p(z \mid c, k)$ for $k \in \{0, |G|, 2|G|, \ldots, |K|\}$ can now be chosen arbitrarily, providing another way to incorporate domain knowledge. In the experiments we use a fixed variance (e.g. $\Sigma := I_{n_e}$ ) and distribute the means over some grid in a minimal subset of the latent space that allows covering the desired area using $J_g$. For instance, for a rotation-equivariant 2D latent space, one could evenly distribute a quarter of the means in some area of one quadrant (e.g. in $[0, 5] \times [0, 5]$). By definition of the expectation, the area $[-5, 5] \times [-5, 5]$ would thus be covered in total.

## 3.5. The Decoders

The algorithm uses two separate decoder neural networks to reconstruct $s'$ and $r$. The probability of observing $x$ given $z$ and $c$ factorizes as:

$$\begin{aligned} \log p_\theta(x \mid c, z) &= \log p_\theta\left(s', r \mid s, a, c, z\right) \\ &= \log p_\theta\left(s' \mid s, a, c, z\right) + \log p_\theta(r \mid s, a, c, z) \end{aligned}$$

The output of the decoder networks can be interpreted as being distributed according to $r \sim \mathcal{N}\left(f_{\text{rew},\theta}(s, a, c, z), \Sigma\right)$ and $s' \sim \mathcal{N}\left(f_{\text{state},\theta}(s, a, c, z), \Sigma\right)$ where $f_{\text{rew},\theta}, f_{\text{state},\theta}$ are the mappings induced by the decoder networks and $\Sigma$ is some fixed, non-negative diagonal matrix. The log-likelihood of the normal distribution thus yields:

$$\mathcal{L}_{\text{decoder}} := -\log p_\theta(x \mid c, z) = \underbrace{\frac{1}{2}\|f_{\text{state},\theta}(s, a, c, z) - s'\|_2^2}_{=: \mathcal{L}_{\text{state dec}}} + \underbrace{\frac{1}{2}\|f_{\text{rew},\theta}(s, a, c, z) - r\|^2}_{=: \mathcal{L}_{\text{rew dec}}}$$

Equivariance can be attained by choosing:

$$(s, a, c, z) \mapsto \left(L_g[s], K_g[a], c, J_g[z]\right)$$

as the input transformation along with $K_g$ and the identity function as output transformation for state and reward decoder, respectively.

## 3.6. Soft-Actor-Critic

To make the actions taken by the SAC (see subsection 2.1.2) equivariant, the algorithm chooses the identity as the output transformation of the $Q$-networks and

$$(\mu, \Sigma) \mapsto \left(K_g[\mu], \left|K_g[\Sigma]\right|\right)$$

for the policy (where $|\cdot|$ is applied element-wise and $K_g$ along the diagonal again). The input transformation $L_g$ for the state (for all networks) and $K_g$ for the action (only for the $Q$-networks). Independent of the encoder and decoder, there are now multiple choices for the representation of the task encoding that is passed to policy and $Q$-networks.

**Sampled latent variables** The simplest option is just sampling $c, z \sim q_\phi\left(c, z \mid \tau^{\mathcal{T}}_{t-T-1:t-1}\right)$ from the distribution generated by the encoder. Equivariance can then be achieved by using the same input transformation for the decoders as in subsection 3.5.

**Distribution parameters** We employ another version of informing the agent of the passing the parameters of the distribution $q_\phi\left(c, z \mid \tau^{\mathcal{T}}_{t-T-1:t-1}\right)$ similar to [20, 21]. In our case, those would be the probabilities, along with the means and variances for all $i \in \{0, \dots, C-1\}, j \in \{0, \dots, K-1\}$. To make this setup equivariant, we arrange those values into vectors $q \in \mathbb{R}^{C \cdot K}, \mu, \sigma^2 \in \mathbb{R}^{C \cdot K \cdot n}$. We define the input transformation as follows:

$$\begin{aligned}
&\forall l \in \{0, \dots, CK-1\}: \\
&\qquad q_l \mapsto q\left|\frac{l}{|G|}\right| |G| + n\left(n^{-1} * g^{-1}(l \bmod |G|)\right) \\
&\qquad \mu_l \mapsto J_g\left[\mu_{\left|\frac{l}{|G|}\right||G|+n(n^{-1}*g^{-1}(l \bmod |G|))}\right] \\
&\qquad \sigma^2_l \mapsto \left|J_g\left[\sigma^2_{\frac{l}{|G|}||G|+n(n^{-1}*g^{-1}(l \bmod |G|))}\right]\right|
\end{aligned}$$

where $|\cdot|$ denotes the element-wise absolute value again.

## 3.7. Hyperparameters

The insufficient performance of previous experiments could be caused by missing hyperparameter tuning [5]. Due to computational complexity, earlier experiments could not conduct this training. They applied conventional hyperparameters from other methods. However, these methods ran in different environments and did not account for symmetry, which poses a substantial difference. Finetuning hyperparameters is a crucial step in Meta-RL since many variables influence each other. In this thesis, we employ two strategies of hyperparameter optimization. First, a *random search* and then a *grid search*; we decided on these variants since they are relatively inexpensive computational-wise [22].

Grid search specifies a grid of hyperparameters to search over and exhaustively try out every possible combination. This simple and straightforward method can be computationally expensive and may not always find the optimal solution, especially for large search spaces. It runs at complexity $\mathcal{O}(n^2)$.

The random search involves randomly sampling hyperparameters from a specified distribution for each iteration. This method can be more efficient than grid search and has been shown to perform well in practice, but it still has the risk of getting stuck in sub-optimal regions of the search space.

As a consequence of high computational workload and the lack of resources of them, the project had to be focused upon a subset of all possible parameters. We have identified the following parameters, which by small changes showed high activity in performance:

**SAC Layer Size** The SAC Layer Size determines the capacity of the model to learn from the data and represent the underlying distribution of the task. A larger layer size allows the model to fit complex meta-tasks, but simultaneously increases the risk of overfitting. On the other hand, a smaller layer size restricts the model's capacity, limiting its ability to learn from the data and generalize to newly inferred tasks [23].

**Learning Rate Encoder** The encoder learning rate determines by how much its weights are updated in response to changes in the incoming data [24].

**Target Entropy Factor** The entropy regularization term helps to encourage exploration in the policy by promoting entropy in the action distribution. By setting a target entropy value, the algorithm aims to maintain a certain level of entropy in the policy over the course of training. This helps to ensure that the policy explores a diverse range of states and actions, leading to a more robust and generalizable policy[25].

**Evaluation Interval** The evaluation interval determines how often the algorithm receives feedback on its performance and can adjust its parameters accordingly. A larger evaluation interval can lead to slower convergence, as the meta-policy has less opportunity to receive feedback and improve [26].

**Prior Sigma** The prior sigma is important because it determines the degree of regularization applied to the encoder, which produces a compact representation of the inputs to the agent. It controls the trade-off between preserving the information in the inputs and producing a compact representation that is robust to variations in the them [26].

It is worth noting that the influence of these parameters can not be exactly pinned down. They impact each other in multiple ways, which vary from situation to situation.

# 4. Experiments

## 4.1. The one-sided Toy1D environment

To reproduce the state-of-the-art and to evaluate the performance of our contribution, we use a 1D-toy environment in which the agent chooses its actions. This environment was also used in the previous papers that built the foundation of our work. The 1D-toy environment as seen in Figure 4.1, consists of an agent (represented by the robot) that can choose whether to move 0.2 to the left or 0.2 to the right. The agent receives the state as its current position on the x-axis. Here the objective is to reach the goal position $x_{goal} \in [-25, 25]$. The negative distance to the goal gives the dense reward. For training, the agent will receive all goal positions only on one site.

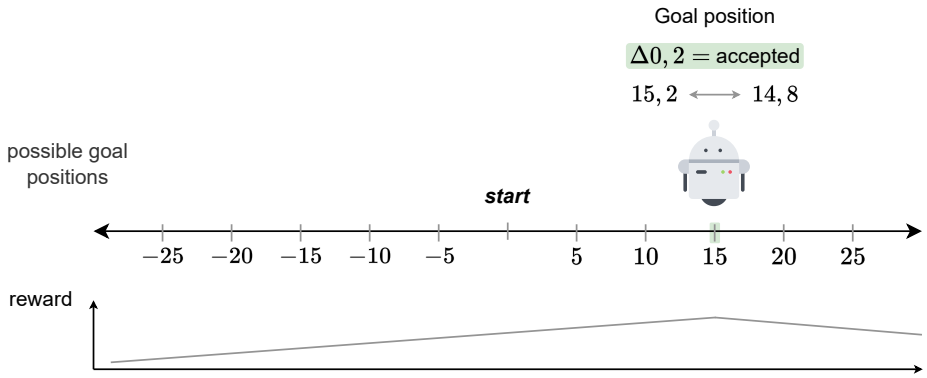



Figure 4.1.: Illustration of one-sided Toy1D environment.

To reproduce the state-of-the-art we set up an environment and trained the networks of the algorithms' different components over various amounts of epochs. We used the trained model to evaluate the performance without making any modifications. The orange graph in Figure 4.2 shows that our reproduced results are similar to the results gathered in [5]. The algorithm does not converge and it decreases strongly after it achieves decent results within the first few epochs.

As stated in section 3.7, we focused on tuning the hyperparameters with a strategy that classifies the importance of parameters and subsequently iterates through different combinations of the important parameters with a *grid search*. Because of the number of hyperparameters influencing the model, a clear separation between more- and less-important hyperparameters is hard to achieve. However, finetuning on a small selection of hyperparameters leads to performance increases and better convergence of the algorithm. The blue graph in Figure 4.2 shows the performance of the algorithm after we improved the selected hyperparameters.

A possible cause for the strong decrease in the early epochs could be a bad encoding propagated to the agent. Since the latent space is interpretable in this setting, comparing

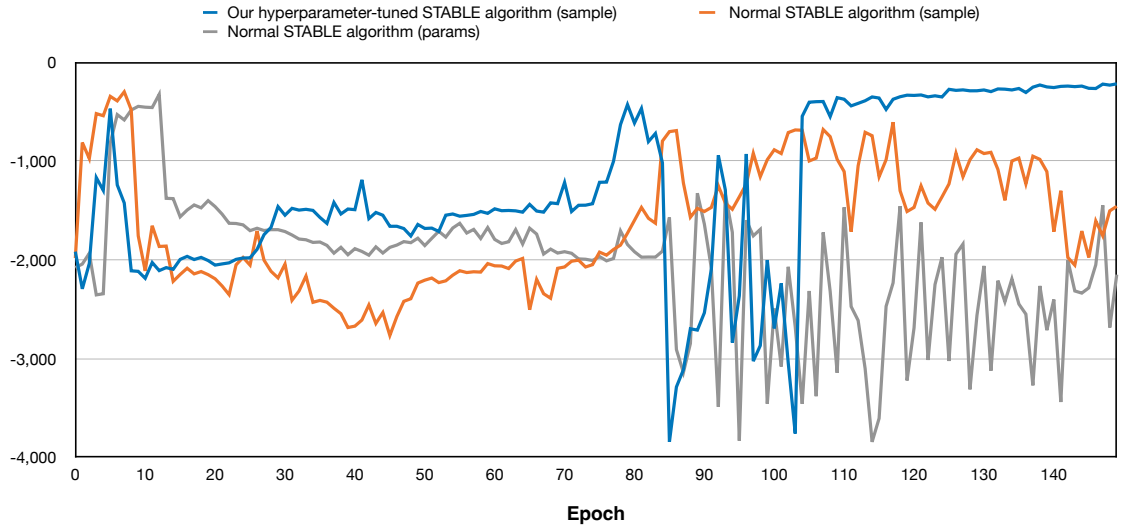



Figure 4.2.: Average deterministic test reward achieved by hyperparameter-tuned STABLE vs. normal STABLE run

the encoding and goal distributions immediately before and after the decrease makes sense. Figure 4.4 displays our agent's behavior before- and Figure 4.5 displays the behavior after the decrease in performance. If we compare the encoding charts, Figure 4.4 shows an almost perfect encoding except for the goal position at 0. All task encodings approach a unique value after the first few time steps (Figure 4.4b). This leads to a great performance as the agent nearly finds all goal positions, displayed in 4.4a. In subsequent epochs, the encoding quality drastically decreases and alternates between positive and negative values of the same magnitude, as shown in 4.5b. Ultimately, this leads to an agent, that can not find the desired goal positions (4.5a) since the received encoding gives

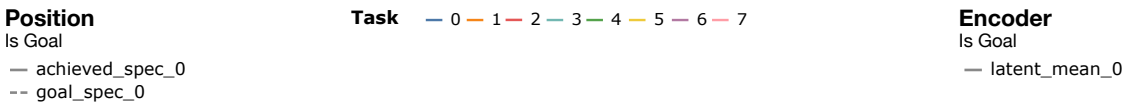
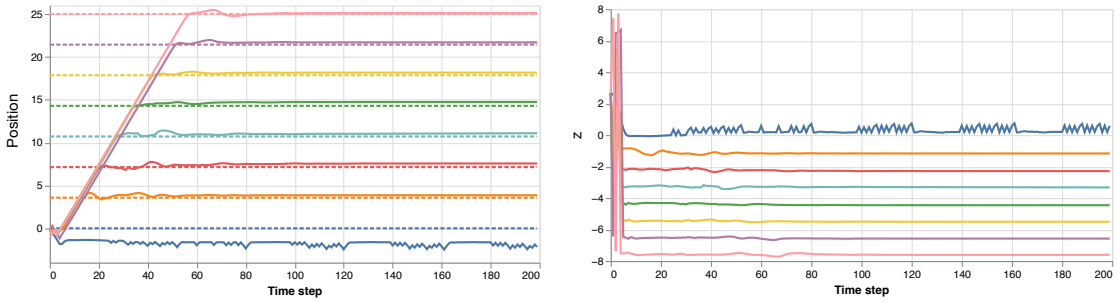


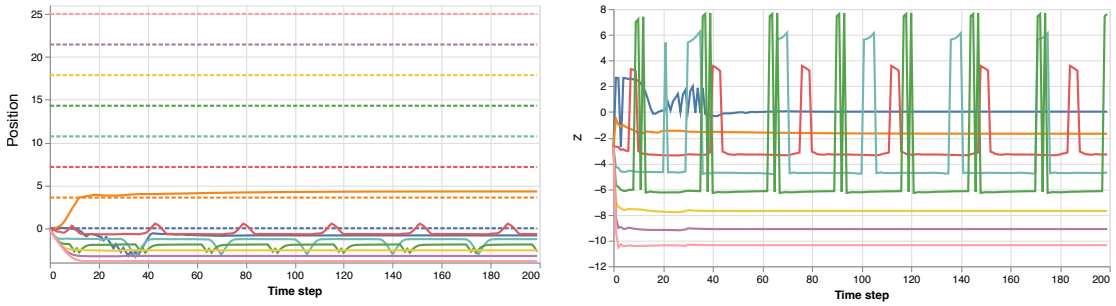

Figure 4.3.: Legend to colors and line style in subsequent figures




(a) The agents x-position over the course of 200 transitions in regard to the optimal goal position

(b) The encoders task encoding over the course of 200 transitions in regard to each goal position

Figure 4.4.: Positive result of the task encoding and achieved positions




(a) The agents x-position over the course of 200 transitions with the optimal goal position

(b) The encoders task encoding over the course of 200 transitions with each goal position

Figure 4.5.: Encoding and respective x-position of the agent that led to decline in performance

unclear instructions about the task at hand. Even though not all encodings are bad, the agent cannot reach the desired goal positions with decent performance. For instance, shown by the pink task in Figure 4.5.

### 4.1.1. Hyperparameter

As stated in section 3.7, it is impossible to perfectly distinguish between different hyperparameters and their direct impact on the overall performance. This led us to experiment with multiple hyperparameters and variations of their values in one run. Figure 4.6 shows an example of such a grid search, of which we always picked upon the most stable and best-performing combinations.

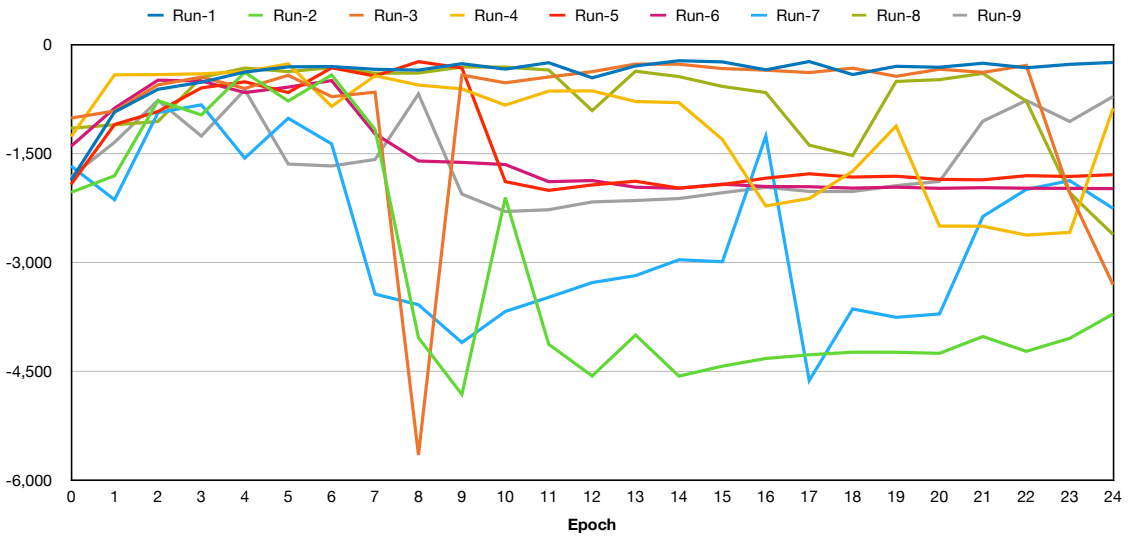


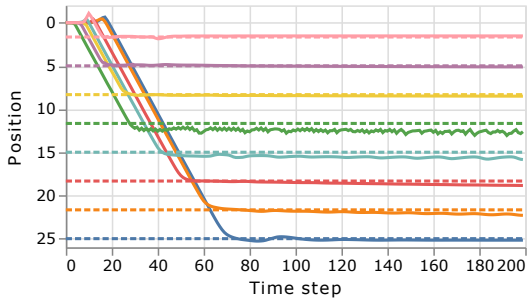

Figure 4.6.: Results of one hyperparameter tuning run

### 4.1.2. Generalization to out-of-distribution tasks

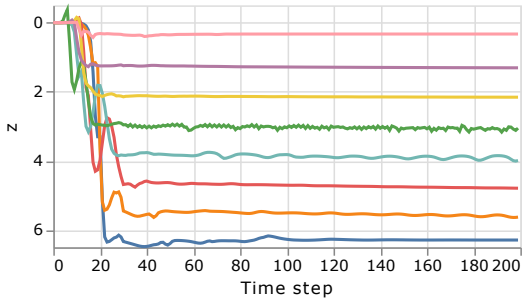
The algorithm introduced by [5] distinguishes itself from previous work, by using networks that leverage equivariance. This is an advantage to previous implementations

because the algorithm can generalize well to out-of-distribution tasks that fall into the category of the in section 2.4 specified equivariant transformations. Figure 4.7 displays the generalizations of the equivariance-based algorithm to goal positions on the opposite side of the trained tasks. In contrast, Figure 4.8 displays the generalization of non-equivariance-based algorithms in the specific setting. The equivariance-based algorithm clearly shows better performance and generalization ability by leveraging equivariant transformations. Since our results show better performance, as stated in subsection 4.1, we suggest that the generalization of our hyperparamter tuned version achieves similar to slightly better results than the algorithm in Figure 4.7.

Note that all additional experiments conducted in previous works can be reproduced with our trained policy but would be out of the scope for this project since time and computing resources were a great limitation.

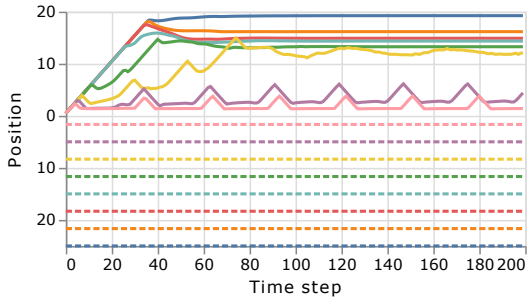



(a) The agents x-position over the course of 200 transitions in regard to the optimal goal position

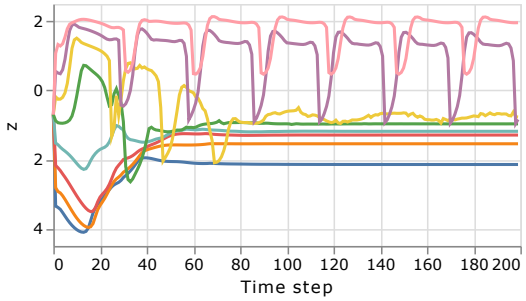



(b) The encoders task encoding over the course of 200 transitions in regard to each goal position

Figure 4.7.: Positive example for generalization of unseen tasks [5]




(a) The agents x-position over the course of 200 transitions in regard to the optimal goal position




(b) The encoders task encoding over the course of 200 transitions in regard to each goal position

Figure 4.8.: Negative example for generalization of unseen tasks [5]

# 5. Discussion

**Stabilization** We encountered many runs where the agent fell into a state of producing alternating patterns in subsequent epochs or great decreases in performance. We were not able to fully eradicate this behavior. By using different combinations of adjustments, we were able to stabilize the algorithm in between the great increase leaps and decrease drops. As an experimental measurement taken to eradicate the drops and leaps, we drastically decreased the encoder's learning rate and decreased the evaluation window. The measure taken positively impacted the stability and convergence of the algorithm but could not yield average returns as high as in runs with a higher learning rate.

**Hyperparameters** We chose the specific run in Figure 4.2 as an example because it approaches convergence and is mostly stable throughout the chosen epoch window. The reduction of the target entropy factor was one of the hyperparameters that influenced this performance. A possible explanation for this is the trade-off between exploration and exploitation that it affects. We figured that a lower target entropy factor and the associated increase in exploitation leads to a more stable performance since the agent chooses the actions that it knows lead to a higher reward, instead of exploring new trajectories. Important to note is that the parameters we used for the best-performing run were only a fraction of the hyperparameter runs we conducted. We had multiple other runs that performed well, even more, stable and without any signs of a decrease in performance. Those, however, were part of a broad hyperparameter-tuning and thus were conducted with fewer epochs. Our infrastructure posed constraints on the duration of hyperparameter optimization and the computational resources that could be utilized.

**Generalization to out-of-distribution tasks** Even if we were able to show a decent performance increase through hyperparameter tuning, the question remains whether these results would still be reproducible to tasks that do not fall into the in section 2.4 described tasks. This is in line with concerns previous researchers had about generalization qualities. In this case, the forced equivariance means we rely on the channel to distinguish between qualitatively distinct tasks. If an unseen task is correctly classified into a channel that has not been used during training, there is no way for the policy to know how to behave. Nonetheless, the linear relationship between the magnitude of the encoding and the magnitude of the associated goal position preserves. This relationship enables the agent to react to similar unseen tasks that fall in between known tasks. Recall section 2.2, the new unseen task would receive an encoding in between the two known tasks in the latent space.

# 6. Conclusion

## 6.1. Conclusion

Our primary goal in this thesis was to reproduce the state-of-the-art performance. We reached this goal and were even able to enhance the performance in some cases. We examined which hyperparameters we have to tweak to achieve a better performance than previous experiments. We focused on the version of the algorithm which samples from the latent task representations. Our enhanced version outperforms other state-of-the-art Meta-RL algorithms in this environment. We compared it to the best-performing version of STABLE so far, an approach known to outperform the prominent baseline PEARL or CEMRL architecture. However, these findings do not generalize well to versions where the latent representation is passed as a parameter or as the latent grid. Thus, it does not improve the method which achieves Bayes-Optimal Behaviour under uncertainty.

## 6.2. Future Work

There are several possible ideas future work may improve upon. First, we had to restrict the hyperparameter search to only five hyperparameters for this report. These have already reached the limits of the computational possibilities we had. Also, it would be possible to try different parameters or different searches, for example, the Bayes search.

Second, the search also needs to be extended to the other versions, which differ in passing on the latent encodings. This could be done with a relatively small effort. Since little differences can already have a huge impact here, this is also one of the most efficient improvements we recommend.

Prior work has already shown that the algorithm can generalize to non-stationary environments in a suboptimal manner. If the encoder only receives the last T time steps, even when trained in a stationary setting, it could be directly trained with changing tasks. Future work could pick this up and show experiments with a stable performance.

Finally, as proposed in [13], the networks used in this thesis are convenient due to their ease of use and versatility. However, they are not the only way to design equivariant networks. According to Pol et al. [13], even without using equivariance, these networks are not equivalent to dense neural network layers, which can negatively impact training due to unwanted inductive biases. Our approach focuses on the presence of the desired equivariance in the layer, regardless of how it is achieved. It can be replaced with any of the numerous equivariant networks designed for specific tasks. If optimized on this, one could maybe even try the algorithm in the 2D environment.

# A. General Addenda

## A.1. Detail of lagged Q-Networks

Unlike the $Q - functioin$ for the MDP, the objective function of the SAC algorithm not only takes into account the discounted reward from the policy $\pi$ starting with action $a$ in state $s$, but also includes the entropy of the actions taken. This results in a more robust and stable learning process, as well as improved exploration of the state space; the concept that helps us realize this is done by training two Q-Networks to stabilize the training by introducing the concept of lagged versions $Q_{\phi_{1,\text{ targ}}}, Q_{\phi_{2,\text{ targ}}}$ lavered during the training objective. Details in [5].

Besides training two Q-Networks to stabilize the training, we also introduce the concept of lagged versions $Q_{\phi_{1,\text{ targ}}}, Q_{\phi_{2,\text{ targ}}}$ lavered during the training objective. Here are the weights updated as $\phi_{i,\text{ targ}} := \tau \cdot \phi_i + (1-\tau)\phi_{i,\text{ targ}}$ for $i \in [2]$ and some hyperparameter $\tau \in (0,1]$. Unlike the $Q - functioin$ for the MDP, the objective function of the Soft Actor-Critic (SAC) algorithm not only takes into account the discounted reward from following policy $\pi$ starting with action $a$ in state $s$, but also includes the entropy of the actions taken. This results in a more robust and stable learning process, as well as improved exploration of the state space and we get the target value of

$$q_{\text{target}}\ (r_t, d_t, s_{t+1}, a_{t+1}) := r_t + (1-d_t)\cdot\gamma\cdot\min_{i\in[2]}\left\{Q_{\phi_{i,\text{ targ}}}\ (s_{t+1}, a_{t+1})\right\} - \alpha_\psi \cdot \log\pi\left(a_{t+1} \mid s_{t+1}\right)$$

where $d_t$ is 1 if $s_t$ was a terminal state (e.g. because $t = H$ ) and 0 otherwise. To converge to this value, $Q_{\phi_1}, Q_{\phi_2}$ are trained by minimizing the mean squared error using samples from the replay buffer:

$$\mathop{\mathbb{E}}_{\substack{(s_t,a_t,r_t,s_{t+1},d_t)\sim\mathcal{D}\\ a_{t+1}\sim\pi(a|s_{t+1})}}\left[\frac{1}{2}\left(Q_{\phi_i}\left(s_t,a_t\right) - q_{\text{target}}\ (r_t,d_t,s_{t+1},a_{t+1})\right)^2\right]$$

## A.2. Varational Autoencoder

Varational Autoencoder motivated in [7, 5]. For:

$$\begin{aligned} z &\sim p(z) = \mathcal{N}\left(\overrightarrow{0}_{n_e}, I_{n_e}\right) \\ x &\sim p_\theta(x \mid z) = \mathcal{N}\left(\mathbb{E}\left[p_\theta(x \mid z)\right], \text{Var}\left[p_\theta(x \mid z)\right]\right) \end{aligned}$$

where $I_n \in \mathbb{R}^{n\times n}$ denotes the $n \times n$ identity matrix and $\overrightarrow{0}_n \in \mathbb{R}^n$ is a vector of zeros. To make it possible for $p(x \mid z)$ to be learned by a neural network, we approximate it by a multivariate normal distribution (with diagonal covariance matrix). Thus it can be learned by a neural network $p_\theta$ with parameters $\theta$ by learning mean and variance for each dimension.

However, to enable learning given only data from $D$, the posterior distribution $p(z \mid x)$ is required, too. In fact, this is the distribution we mainly care about if the goal is to extract structure from the given data. Using Bayes' theorem, we obtain:

$$p(z \mid x) \overset{\text{Bayes}}{=} \frac{p(x \mid z)p(z)}{p(x)}$$

Nevertheless, calculating $p_\theta(x) = \int_{\mathbb{R}^{n_e}} p(x \mid z)p(z)dz$ is intractable in general. Therefore, the assumptions about the (multivariate normal) distribution of $p(x \mid z)$ are also made for the posterior and it is learned by a separate neural network $q_\phi$ with parameters $\phi$. The goal is to find optimal parameters $\boldsymbol{\theta}^*$, such that the distance between the true posterior $q_\phi(z \mid x)$ and its approximate posterior $p_\theta(z \mid x)$ is minimized in terms of the Kullback-Leibler divergence (KL-divergence).

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathbb{KL}\left(q_\theta(\boldsymbol{z} \mid \boldsymbol{x}) \| p_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})\right)$$

Inserting the posterior and doing some algebra gives:

$$\begin{aligned}
\mathbb{KL}\left(q_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x}) \| p_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})\right) &= \mathbb{E}_{q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{q_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x})}{p_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})}\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log q_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x}) - \log p_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log q_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x}) - \log p_{\boldsymbol{\phi}}(\boldsymbol{x} \mid \boldsymbol{z}) - \log p_{\boldsymbol{\phi}}(\boldsymbol{z}) + \log p_{\boldsymbol{\phi}}(\boldsymbol{x})\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\left[-\log p_{\boldsymbol{\phi}}(\boldsymbol{x} \mid \boldsymbol{z})\right] + \mathbb{KL}\left(q_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x}) \| p_{\boldsymbol{\phi}}(\boldsymbol{z})\right) + \log p_{\boldsymbol{\phi}}(\boldsymbol{x})
\end{aligned}$$

As usual in variational inference, this is rewritten to find an evidence lower bound (ELBO) $\mathcal{L}_{\text{ELBO}}$:

$$\begin{aligned}
\log p_{\boldsymbol{\phi}}(\boldsymbol{x}) &= \mathbb{KL}\left(q_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x}) \| p_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})\right) + \mathcal{L}_{\text{ELBO}} \\
&\geq \mathcal{L}_{\text{ELBO}} = \mathbb{E}_{q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\phi}}(\boldsymbol{x} \mid \boldsymbol{z})\right] - \mathbb{KL}\left(q_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x}) \| p_{\boldsymbol{\phi}}(\boldsymbol{z})\right)
\end{aligned}$$

Revisting this Equation, to get the optimal parameters $\boldsymbol{\theta}^*$ that minimize the distance between the true and approximate posterior, the ELBO needs to be maximized. Further, for optimal reconstruction the log-likelihood of the data $\log p_{\boldsymbol{\phi}}(\boldsymbol{X})$ is maximized. Combined, this leads to the following objective, optimizing parameters for the encoder and decoder jointly:

$$\boldsymbol{\theta}^*, \boldsymbol{\phi}^* = \arg\max_{\boldsymbol{\theta}, \boldsymbol{\phi}} \mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\phi})$$

$$\mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_n^N (\underbrace{\mathbb{E}_{z \sim q_{\boldsymbol{\theta}}(z|x_n)}\left[\log p_{\boldsymbol{\phi}}\left(\boldsymbol{x}_n \mid \boldsymbol{z}\right)\right]}_{\text{reconstruction error}} - \underbrace{\mathbb{KL}\left(q_{\boldsymbol{\theta}}\left(\boldsymbol{z} \mid \boldsymbol{x}_n\right) || p_{\boldsymbol{\phi}}(\boldsymbol{z})\right)}_{\text{regulariser}})$$

where the first part of the equation can be interpreted as reconstruction error, calculated as the log-likelihood of the decoded data estimated via Monte Carlo sampling. A separate latent encoding is sampled for each data point from the approximate posterior. The regularization term helps keep the approximate posterior close to the prior, encoding only essential information for reconstruction in the latent variable $z$. This regularization results in a natural form of regularization and is known as an "information bottleneck".

The so-called reparametrization trick of rewriting $z \sim \mathcal{N}\left(\mathbb{E}\left[q_\theta(z \mid x)\right], \text{Var}\left[q_\theta(z \mid x)\right]\right)$ as

$$z = \mathbb{E}\left[q_\theta(z \mid x)\right] + \text{diag}^{-1}\left(\text{Var}\left[q_\theta(z \mid x)\right]\right) \odot \varepsilon \quad \text{where } \varepsilon \sim \mathcal{N}\left(\overrightarrow{0}_{n_e}, I_{n_e}\right)$$

enables backpropagation through $q_\theta$ and makes this goal feasible. Here, $\text{diag}^{-1}$ denotes the function that maps a diagonal matrix $A \in \mathbb{R}^{n \times n}$ to the vector $\text{diag}^{-1}(A) \in \mathbb{R}^n$ of its diagonal entries.

## A.3. BAMDP

Hyperstate and reward then change according to:

$$\begin{aligned} p^+\left(s_{t+1}^+ \mid s_t^+, a_t, r_t\right) &= p^+\left(s_{t+1} \mid s_t, b_t, a_t\right) p^+\left(b_{t+1} \mid s_t, b_t, a_t, r_t, s_{t+1}\right) \\ &= \mathbb{E}_{p \sim b_t(p \mid \tau_{0:t})}\left[p\left(s_{t+1} \mid s_t, a_t\right)\right] \delta\left(b_{t+1} = p\left(r, p \mid \tau_{0:t+1}\right)\right) \\ r^+\left(s_t^+, a_t, s_{t+1}^+\right) &= \mathbb{E}_{r \sim b_{t+1}}\left[r\left(s_t, a_t\right)\right] \end{aligned}$$

where $\delta(x)$ denotes the Dirac delta distribution. Intuitively, that means that the new MDP state $s_{t+1}$ is sampled from the expected transition function for the given experience $\tau_{0:t}$ and the new belief $b_{t+1}$ is just $p\left(r, p \mid \tau_{0:t+1}\right)$ deterministically by definition.

Defining $p^+(\tau \mid \pi)$ analogously to $p(\tau \mid \pi)$ with $\mathcal{S}^+, r^+, p^+$instead of $\mathcal{S}, r, p$, the goal of the policy $\pi\left(a_t \mid s_t^+\right)$is maximizing

$$\mathbb{E}_{\tau \sim p^+(\tau \mid \pi)}\left[\sum_{t=0}^{H^+-1} \gamma^t r^+\left(s_t^+, a_t, s_{t+1}^+\right)\right]$$

A policy maximizing this objective is called Bayes-optimal. Whereas the belief about the current transition and reward function $b_t = p\left(p, r \mid \tau_{0:t}\right)$ changes with the agent exploring the environment. [5]

## A.4. Generative Model

$$
\begin{aligned}
&\mathbb{KL}(q(c,k,z \mid x) \| p(c,k,z \mid x)) \\
=&\mathbb{E}_{c,k,z\sim q(c,k,z|x)}\left[\log \frac{q(c,k,z \mid x)}{p(c,k,z \mid x)}\right] \\
=&\mathbb{E}_{c,k,z\sim q(c,k,z|x)}[\log q(c,k,z \mid x) - \log p(c,k,z \mid x)] \\
=&\mathbb{E}_{c,k,z\sim q(c,k,z|x)}[\log q(z \mid x,c,k) + \log q(c,k \mid x) \\
&- \log p(x \mid c,k,z) - \log p(z \mid c,k) - \log p(c,k)] + p(x) \\
=&\mathbb{E}_{c,k,z\sim q(c,k,z|x)}\left[-\log p(x \mid c,z) + \log \frac{q(z \mid x,c,k)}{p(z \mid c,k)} + \log \frac{q(c,k \mid x)}{p(c,k)}\right] + \log p(x) \\
=&\mathbb{E}_{c,k\sim q(c,k|x)}\left[\mathbb{E}_{z\sim q(z|x,c,k)}[-\log p(x \mid c,z)]\right. \\
&\left.+\mathbb{E}_{z\sim q(z|x,c,k)}\left[\log \frac{q(z \mid x,c,k)}{p(z \mid c,k)}\right] + \log \frac{q(c,k \mid x)}{p(c,k)}\right] + \log p(x) \\
=&\mathbb{E}_{c,k\sim q(c,k|x)}\left[\mathbb{E}_{z\sim q(z|x,c,k)}[-\log p(x \mid c,z)] + \mathbb{KL}(q(z \mid x,c,k) \| p(z \mid c,k))\right] \\
&+ \mathbb{KL}(q(c,k \mid x) \| p(c,k)) + \log p(x) \\
\Leftrightarrow \log p(x) =&\mathbb{KL}(q(c,k,z \mid x) \| p(c,k,z \mid x)) + \mathbb{E}_{c,k\sim q(c,k|x)}\left[\mathbb{E}_{z\sim q(z|x,c,k)}[\log p(x \mid c,z)]\right. \\
&\left.- \mathbb{KL}(q(z \mid x,c,k) \| p(z \mid c,k))\right] - \mathbb{KL}(q(c,k \mid x) \| p(c,k))
\end{aligned}
$$

# B. Figures

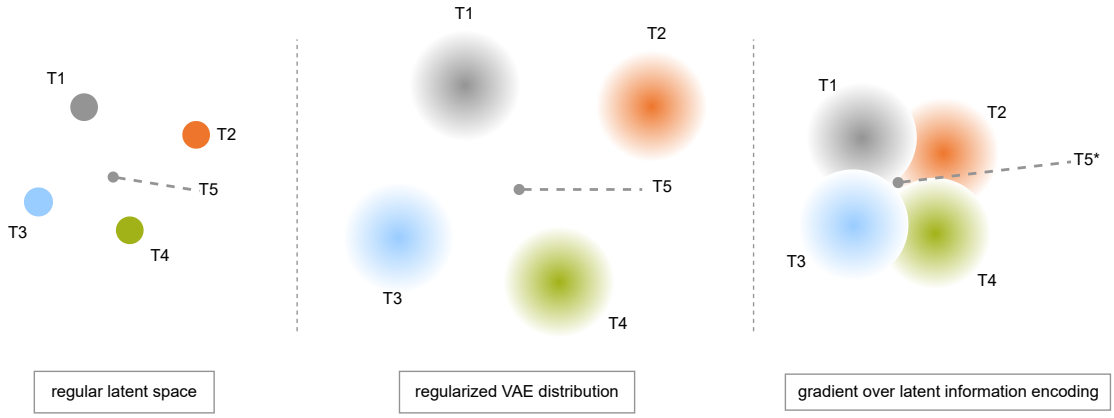



Figure B.1.: Left: Encoding tasks in a regular fashon without the use of stochastic distributions; Middle: Using a Gaussian Disstribution for different Task Encodings and picking task $T5$ for action; Right: Using a combination of learned task encodings with help of the gradient over them

# Bibliography


[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529 (2016), pp. 484–503. URL: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

[2] P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei. *Deep reinforcement learning from human preferences*. 2017. DOI: 10.48550/ARXIV.1706.03741. URL: https://arxiv.org/abs/1706.03741.

[3] T. Hester and P. Stone. "Texplore: real-time sample-efficient reinforcement learning for robots". In: *Machine learning* 90 (2013), pp. 385–429.

[4] T. S. Cohen and M. Welling. "Group Equivariant Convolutional Networks". In: *arXiv e-prints*, arXiv:1602.07576 (Feb. 2016), arXiv:1602.07576. DOI: 10.48550/arXiv.1602.07576. arXiv: 1602.07576 [cs.LG].

[5] J. Jürß. "Exploiting Symmetries in Context-Based Meta-Reinforcement Learning". In: (2022).

[6] K. Rakelly, A. Zhou, D. Quillen, C. Finn, and S. Levine. "Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables". In: *arXiv e-prints*, arXiv:1903.08254 (Mar. 2019), arXiv:1903.08254. DOI: 10.48550/arXiv.1903.08254. arXiv: 1903.08254 [cs.LG].

[7] D. Lerch. "Meta-Reinforcement Learning in Non-Stationary and Dynamic Environments". In: (2020).

[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: *arXiv e-prints*, arXiv:1801.01290 (Jan. 2018), arXiv:1801.01290. DOI: 10.48550/arXiv.1801.01290. arXiv: 1801.01290 [cs.LG].

[9] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. "Soft Actor-Critic Algorithms and Applications". In: *arXiv e-prints*, arXiv:1812.05905 (Dec. 2018), arXiv:1812.05905. DOI: 10.48550/arXiv.1812.05905. arXiv: 1812.05905 [cs.LG].

[10] 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". In: *arXiv e-prints*, arXiv:1803.11347 (Mar. 2018), arXiv:1803.11347. DOI: 10.48550/arXiv.1803.11347. arXiv: 1803.11347 [cs.LG].

[11] M. O. Duff. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. https://scholarworks.umass.edu/dissertations/AAI3039353. University of Massachusetts Amherst, 2002.

[12] D. P. Kingma and M. Welling. "Auto-Encoding Variational Bayes". In: *arXiv e-prints*, arXiv:1312.6114 (Dec. 2013), arXiv:1312.6114. DOI: 10.48550/arXiv.1312.6114. arXiv: 1312.6114 [stat.ML].

[13] E. van der Pol, D. E. Worrall, H. van Hoof, F. A. Oliehoek, and M. Welling. "MDP Homomorphic Networks: Group Symmetries in Reinforcement Learning". In: *arXiv e-prints*, arXiv:2006.16908 (June 2020), arXiv:2006.16908. DOI: 10.48550/arXiv.2006.16908. arXiv: 2006.16908 [cs.LG].

[14] I. Arnekvist, D. Kragic, and J. A. Stork. *VPE: Variational Policy Embedding for Transfer Reinforcement Learning*. 2018. DOI: 10.48550/ARXIV.1809.03548. URL: https://arxiv.org/abs/1809.03548.

[15] D. Reynolds. "Gaussian Mixture Models". In: *Encyclopedia of Biometrics*. Ed. by S. Z. Li and A. K. Jain. Boston, MA: Springer US, 2015, pp. 827–832. ISBN: 978-1-4899-7488-4. DOI: 10.1007/978-1-4899-7488-4_196. URL: https://doi.org/10.1007/978-1-4899-7488-4_196.

[16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[17] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. "beta-vae: Learning basic visual concepts with a constrained variational framework". In: *International conference on learning representations*. 2017.

[18] Z. Bing, L. Knak, F. O. Robin, K. Huang, and A. Knoll. "Meta-Reinforcement Learning in Broad and Non-Parametric Environments". In: *arXiv e-prints*, arXiv:2108.03718 (Aug. 2021), arXiv:2108.03718. DOI: 10.48550/arXiv.2108.03718. arXiv: 2108.03718 [cs.LG].

[19] P. Widmann. "Task Inference for Meta-Reinforcement Learning in Broad and Non-Parametric Environments". In: (2022).

[20] L. Zintgraf, K. Shiarlis, M. Igl, S. Schulze, Y. Gal, K. Hofmann, and S. Whiteson. "VariBAD: A Very Good Method for Bayes-Adaptive Deep RL via Meta-Learning". In: *arXiv e-prints*, arXiv:1910.08348 (Oct. 2019), arXiv:1910.08348. DOI: 10.48550/arXiv.1910.08348. arXiv: 1910.08348 [cs.LG].

[21] L. M. Zintgraf, L. Feng, C. Lu, M. Igl, K. Hartikainen, K. Hofmann, and S. Whiteson. "Exploration in Approximate Hyper-State Space for Meta Reinforcement Learning". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by M. Meila and T. Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, July 2021, pp. 12991–13001. URL: https://proceedings.mlr.press/v139/zintgraf21a.html.

[22] L. Yang and A. Shami. "On hyperparameter optimization of machine learning algorithms: Theory and practice". In: *Neurocomputing* 415 (Nov. 2020), pp. 295–316. DOI: 10.1016/j.neucom.2020.07.061.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. *Playing Atari with Deep Reinforcement Learning*. 2013. DOI: 10.48550/ARXIV.1312.5602. URL: https://arxiv.org/abs/1312.5602.

[24] J. Rothfuss, D. Lee, I. Clavera, T. Asfour, and P. Abbeel. *ProMP: Proximal Meta-Policy Search*. 2018. DOI: 10.48550/ARXIV.1810.06784. URL: https://arxiv.org/abs/1810.06784.

[25] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *CoRR* abs/1801.01290 (2018). arXiv: 1801.01290. URL: http://arxiv.org/abs/1801.01290.

[26] C. Finn, P. Abbeel, and S. Levine. *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks*. 2017. DOI: 10.48550/ARXIV.1703.03400. URL: https://arxiv.org/abs/1703.03400.

We confirm that this final report is our own work and we have documented all sources and material used.

Munich, 10.02.2023 Gentrit Fazlija, Sören Glaser-Gallion, Anton Mauz, Michel Osswald