Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Identifying and Using Task Relations in Multi-Task Reinforcement Learning

Master's Thesis

Johannes Ackermann

johannes.ackermann@tum.de

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Oliver Richter
Prof. Dr. Roger Wattenhofer

October 28, 2020

Acknowledgements

I would firstly like to thank Oliver Richter, who supervised this thesis, and without whose support this project would not have been possible. Considering that I unexpectedly was not able to relocate to Zurich after all, I'd like to especially thank him and Professor Wattenhofer for supervising this thesis despite it only being possible remotely.

I would also like to thank the other members of the research group for helpful feedback and interesting discussions in the reading group. They helped create a sense of community, especially in the early days of lockdown.

Parts of this thesis, especially Chapter 4, were written in collaboration with Oliver Richter and Professor Roger Wattenhofer as part of a submission currently under review.

Abstract

In the typical Reinforcement Learning (RL) setting we are given a single task and then learn an optimal policy for it. However, in many real-world settings, multiple different tasks are available for training. Depending on whether we aim to perform well on one or all of the tasks we have one of two problem settings: Curriculum Learning, in which we have a set of tasks and want to use them to quickly learn a specific target task, or Multi-Task RL, in which we want to achieve a good performance in all given tasks. While there is previous work in both settings, these approaches usually assume — implicitly or explicitly — all tasks to be uniformly similar to each other. We argue that this assumption is limiting and therefore investigate two ways of identifying similarities between tasks. We then use the learned task relations to speed up training.

Firstly, we propose to use a variational autoencoder to learn a task embedding in which distances correspond to similarities. We then use the embedding to determine a curriculum that is able to quickly learn a difficult exploration task by using experience from easier tasks. We show that our approach is able to learn a meaningful representation of the task space and improve sample complexity on a simple discrete task. However, it fails to outperform baselines in a complex continuous control environment.

Secondly, we investigate the classical Multi-Task setting, for which we propose a method that clusters together similar tasks. Inspired by the expectation-maximization algorithm, we use a set of policies and alternately assign each task to a policy and train the policies on their assigned tasks. This leads to each policy focusing on a set of related tasks, producing a meaningful clustering that avoids negative transfer and can speed up training. As our approach can be combined with any underlying RL method, we evaluate it on a a varied set of simple discrete and continuous control tasks, as well as complex bipedal walker tasks and Atari games. Results show a competitive or favorable performance in all tested environments.

Contents

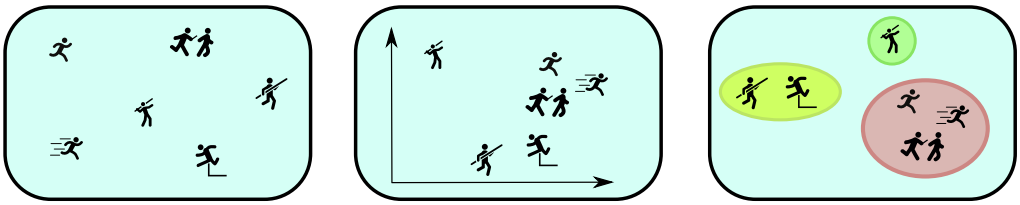
Chapter 1

Introduction

Reinforcement Learning (RL) is a subfield of Machine Learning that addresses the problem of how an agent can learn to act optimally in an unknown environment [1]. The agent receives the current state s, chooses an action a, and receives a reward r and a new state s'. This formulation is as powerful as it is general and has led to many breakthroughs that were previously thought decades beyond the current state of the field. It managed to, for the first time, beat professional Go players with AlphaGo in 2015 [2] after which the next big challenge was seen in complex strategy games such as Dota2 and StarCraft2. In both, RL based approaches managed to defeat the best human players in 2019 with AlphaStar [3] and OpenAIFive [4]. However, while there is work on applying reinforcement learning to real-world tasks, such as robotics [5] or health-care decision making [6], most remains academic in nature with few exceptions [7, 8] being applied in the real-world.

The large difference between games and tasks in the real-world is that training data is readily available in games while in real-world applications the availability of training data is limited [9]. To beat humans in Dota2, tens of thousands of years of game time were necessary [4] and in StarCraft2 180 000 years of game time were used [3]. Generating such large amounts of training data is not feasible in real-world tasks. We, therefore, need to use the data that we can generate as efficiently as possible. While there are many approaches that deal with this issue, we focus on a setting in which we have access to multiple tasks, known as Multi-Task RL [10].

Approaches in this field generally focus on reusing experience from one task in another. The goal is either to achieve a good performance on all tasks or to achieve a good performance in a single target task, as in Curriculum Learning approaches [11]. However, most of the presented methods do not use information about the relationship between tasks and assume that they are uniformly similar to each other. We argue that this is an inappropriate assumption in many settings. Consider the example in Figure 1.1, showing different track and field disciplines. Most approaches treat them as a set of tasks without any specific similarity relationships, as shown in the left. However, sprinting, running



Pictograms: Flaticon.com

Figure 1.1: In many settings in Reinforcement Learning we have access to a set of multiple tasks, here represented by track and field disciplines. Most approaches treat all tasks as being uniformly similar to each other (left). We hypothesize that this is disadvantageous in many settings and propose two methods that learn the relation between tasks: By constructing a task space in which the distance between tasks corresponds to their dissimilarity (middle) and an approach inspired by Expectation-Maximization that clusters similar tasks together in an unsupervised way (right).

and relay are clearly more similar to each other than they are to hurdle runs or pole vaulting. Yet they are all more similar to each other than to javelin toss. A learned task space (middle), in which distance corresponds to dissimilarity, could capture such relations. However, learning such a detailed structure can be difficult and costly in complex environments. A simpler structure is to learn a clustering, as shown on the right in Figure 1.1. Here we only assume that tasks in a cluster will be more similar to each other than to tasks in other clusters, without constraining the relations further.

We propose two corresponding approaches: First, we look into learning a task space, in which the distance between tasks corresponds to their similarity. We propose a method to identify such a task space based on a Variational Autoencoder (VAE) and then use this task space for automatic curriculum learning. Secondly, we look into directly identifying a clustering of tasks without learning a full task space. To do so, we propose an approach inspired by Expectation-Maximization [12] that uses a set of independent policies. It iteratively evaluates the policies on all tasks, assigns tasks to policies and then trains each policy on its assigned tasks. We show that both methods are able to generate interpretable representations of the relations between tasks and can improve sample complexity.

Chapter 2

Background

2.1 Tasks and Markov Decision Processes

In RL, tasks are typically specified by a Markov Decision Process (MDP) defined as tuple $(\boldsymbol{S}, \boldsymbol{A}, P, R, \gamma)$, with state space $\boldsymbol{S}$, action space $\boldsymbol{A}$, transition function $P(s'|s,a)$, reward function $R(s,a,s')$ and decay factor γ [1]. As we are interested in reusing experience from different tasks, we require a shared state-space $\boldsymbol{S}$ and action-space $\boldsymbol{A}$ across tasks. Note, however, that this requirement can be omitted depending on the policy structure. Following prior work, we use task specific output layers in the Multi-Layer Perceptron (MLP) used in our Atari experiments, to account for the different action spaces. In all other experiments tasks only differ in their transition function and reward function. We therefore describe a task as $\tau = (P_\tau, R_\tau)$ and refer to the set of given tasks as $\mathcal{T}$.

For a single task our aim is to find a policy that maximizes the discounted return $G = \sum_{t=0}^{t=\infty} \gamma^t r_t$. For multiple tasks we regard two settings: In Curriculum RL we want to achieve an optimal return in a target task τ^* with the least amount of training steps on all tasks $\tau \in \mathcal{T}$. In Multi-Task RL, we aim to, for each task $\tau \in \mathcal{T}$, maximize the discounted return $G_\tau = \sum_{t=0}^{t=L} \gamma^t r_t^\tau$ where $r_t^\tau \sim R_\tau(s_t, a_t)$ is the reward at time step t and L is the episode length. Given a set of policies $\{\pi_1, ..., \pi_n\}$, we denote the return obtained by policy π_i on task τ as $G_\tau(\pi_i)$.

2.2 Reinforcement Learning

Our goal in RL is to learn a policy $\pi : \boldsymbol{S} \to \boldsymbol{A}$ that maximizes the expected return $G_\tau(\pi)$. A policy that achieves this is called an optimal policy π^*. A foundational method that can find this optimal policy is Q-Learning [13]. In it we learn the Q-value of each state-action pair, defined as the expected discounted return starting with the given state and action $Q(s,a) = \mathbb{E}[G_\tau(\pi)|s_0 = s, a_0 = a] = \mathbb{E}[\sum_{t=0}^{t=\infty} \gamma^t r_t | s_0 = s, a_0 = a]$. We can estimate this value by performing roll-outs of an ϵ-greedy policy, that chooses the optimal action $a^* = \arg\max_a Q(s,a)$ with probability $(1-\epsilon)$ and performs a randomly chosen action with probability

ϵ. We update the values of the state-action pairs that occur during the trajectory with the temporal-difference rule

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(r_t + \gamma \max_{a'} Q(s',a')\right), \tag{2.1}$$

with α being the learning rate and s', a' being the following state and action. This method has been shown to converge to the correct optimal policy π^*, with appropriate values for α and sufficient exploration [13].

2.2.1 DQN

A disadvantage of simple Q-Learning is that it relies on a tabular representation of the value function which makes it impossible to scale to high-dimensional environments. An alternative is to use MLPs as function approximators for the Q-function, parameterized as Q_θ. However, this was problematic for a long time due to a set of problems known as the deadly triad - function approximation, boot-strapping, and off-policy learning - leading to a poor performance [1].

These problems were addressed by Mnih et al. [14] with the introduction of Deep Q-Networks (DQN). While using essentially the same update-rule as in (2.1), they use a target network $Q_{\theta'}$ that is used for bootstrapping and is updated to the parameters of the online network Q_θ periodically $\theta' \leftarrow \theta$. Additionally, they use a replay buffer $\boldsymbol{D}$ that stores previous transitions and sample batches of transitions (s,a,r,s') from it to use in the updates. The MLP Q_θ is therefore trained to minimize the loss

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim \boldsymbol{D}}\left[\left(Q_\theta(s,a) - (r + \gamma \max_{a'} Q_{\theta'}(s',a'))\right)^2\right]. \tag{2.2}$$

2.2.2 Distributional RL

While most approaches only learn the expected return as value $Q(s,a) = \mathbb{E}(G|s_0 = s, a_0 = a)$, distributional RL aims to provide more accurate estimates by learning the full distribution over returns. This was shown to improve performance in Atari games by Bellemare et al. [15]. They introduced C51, an approach that approximates the distribution with a categorical estimate. Implicit Quantile Network (IQN) [16] builds on this work by representing the distribution by quantiles.

2.2.3 Policy Gradient Methods

While the methods listed above work well in environments with discrete action spaces, policies based only on a value function can not directly be applied to

environments with continuous action spaces.

Instead, policy gradient methods [17] were proposed that represent the policy π_ψ as a function parameterized by a set of parameters ψ. With the target $J(\psi) = \mathbb{E}_{s\sim p^\pi, a=\pi_\psi}[\sum_{t=0}^{t=\infty} \gamma^t r_t]$ and p^π being a policy dependent state distribution, the policy gradient theorem [17] states that the gradient resolves to:

$$\nabla_\psi J(\psi) = \mathbb{E}_{s\sim p^\pi, a=\pi_\psi}[\nabla_\psi \log \pi_\psi(a|s) Q^\pi(s,a)]\,. \tag{2.3}$$

This allows the policy to be trained by gradient descend, however, it requires an estimate of the value function $Q^\pi(s,a)$. The estimate can be obtained with monte-carlo methods, generating some sample trajectories under the current policy or it can be estimated by a separate Q-value estimator. In the latter case, the value function is referred to as *critic* and the approach becomes an actor-critic method.

This result was further extended to the case of deterministic policies by Silver et al. [18] and then further combined with MLPs as function approximators by Lillicrap et al. [19]. Similarly to DQN, they use a replay buffer and a target network for the policy and value function approximators, that is updated as an exponential moving average $\theta' \leftarrow \alpha\theta + (1-\alpha)\theta'$. The gradient then resolves to

$$\nabla_\psi J(\psi) = \mathbb{E}_{s\sim \boldsymbol{D}, a=\pi_\psi}[\nabla_\psi \mu_\psi(a|s) \nabla_a Q^\pi(s,a)]\,, \tag{2.4}$$

with μ being a deterministic policy $\mu_\psi : \boldsymbol{S} \to \boldsymbol{A}$. This method is known as Deep Deterministic Policy Gradient (DDPG).

Finally, Fujimoto et al. [20] extend DDPG to Twin Delayed Deep Deterministic Policy Gradient (TD3) and propose three improvements: Firstly, they use two separate networks $(Q_{\theta_1}, Q_{\theta_2})$ to represent the value function and use the minimum of both as target. This reduces a tendency of DDPG to overestimate the Q-values. Secondly, they add noise to the target updates, to serve as regularization. Finally, they update the value function more frequently than the policy to assure an accurate value estimate for use in the policy update.

$$\begin{aligned} \nabla_\psi J(\psi) &= \mathbb{E}_{s\sim \boldsymbol{D}, a=\pi_\psi}[\nabla_\psi \mu_\psi(a|s) \nabla_a Q^\pi_{\theta_1}(s,a)] \\ \mathcal{L}(\theta_i) &= \mathbb{E}_{(s,a,r,s')\sim \boldsymbol{D}}\left[\left(Q_{\theta_i}(s,a) - (r + \gamma \min_{j=1,2} \max_{a'} Q_{\theta'_j}(s',a'))\right)^2\right]\,. \end{aligned} \tag{2.5}$$

2.3 Expectation-Maximization

Expectation-Maximization (EM) [12] is an iterative algorithm for identifying the parameters of a model with an unobserved latent variable. Its best known application is the determination of parameters of a Gaussian mixture model (GMM)

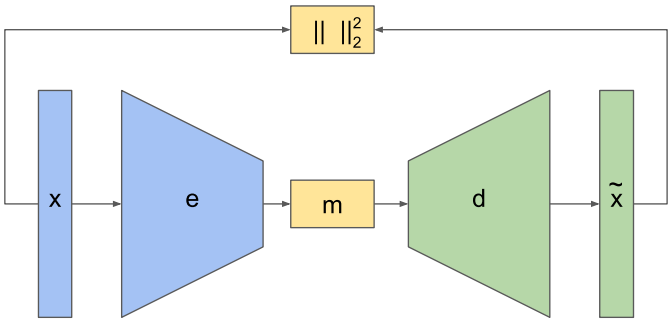


Figure 2.1: Illustration of an autoencoder. Samples x are input in the encoder e, which outputs a low-dimensional m, which is then decoded by the decoder d. For training, the euclidean error $||x - d(e(x))||^2$ is minimized.

$p(x) = \sum_{k=1}^{L} w_k \mathcal{N}(x; \mu_k, \Sigma_k)$. After the GMM-parameters are initiated, the algorithm consists of two repeating steps: The expectation (E)-step, in which the responsibility of each Gaussian for a given data point x_i is determined with the current parameters as

$$\eta_{i,k} = \frac{\hat{w}_k \mathcal{N}(x_i; \hat{\mu}_k, \hat{\Sigma}_k)}{\sum_{k'=1}^{K} \hat{w}'_k \mathcal{N}(x_i; \hat{\mu}'_k, \hat{\Sigma}'_k)},$$

which is followed by a the maximization (M)-step, in which the parameters of the GMM are adjusted to maximize the likelihood under the current responsibilities:

$$\hat{w}_k = \frac{1}{n} \sum_{i=1}^{N} \hat{\eta}_{i,k}, \quad \hat{\mu} = \frac{\sum_{i=1}^{N} \eta_{i,k} x_i}{\sum_{i=1}^{N} \eta_{i,k}}, \quad \hat{\Sigma}_k = \frac{\sum_{i=1}^{n} \hat{\eta}_{i,k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^\top}{\sum_{i=1}^{n} \hat{\eta}_{i,k}}$$

More intuitively, Expectation-Maximization (EM) determines soft assignments of samples to one of the clusters during the E-step and uses these assignments to reweigh the samples when calculating the Gaussian parameters in the M-step.

2.4 Autoencoders

Autoencoders are a special type of MLP popularized by Hinton et al. [21]. The goal of an autoencoder is to find a lower dimensional representation for the elements of a given dataset. This is achieved by having a network which is trained to learn an identity function but includes a bottleneck, a hidden layer with significantly smaller size than the input. An autoencoder can be trained like any MLP by using the same sample as input and desired output and training the weights via gradient descent. The layers before the bottleneck can then be regarded as an encoder $e(x)$ with x being the input sample. The layers following

the bottleneck can be regarded as a decoder $d(m)$, with $m = e(x)$ being the latent code corresponding to sample x from a dataset $x \sim D$. Generally we want the dimensionality of m to be significantly smaller than that of x: $\dim m \ll \dim x$.

The loss function for an autoencoder resolves to

$$\mathcal{L}_{\texttt{AE}} = \mathbb{E}_{x \sim D} \left[||x - d(e(x))||_2^2 \right] \tag{2.6}$$

2.4.1 Variational Autoencoders

VAEs are an extension of autoencoders, proposed by Kingma & Welling [22]. While VAEs can be motivated from a variational inference perspective, for our purposes the main difference is that the encoder parameterizes a distribution $q(m|x) = \mathcal{N}(\mu, \Sigma)$. During training, a sample is drawn from this distribution and then decoded. The loss is calculated between the decoded sample and the input. The main advantage, for us, is that the stochastic nature of the encoder acts as regularization, which encourages a more meaningful, more disentangled latent representation of the inputs. The loss function of a VAE is known as ELBO (evidence lower bound) and consists of two terms:

$$\mathcal{L}_{\texttt{ELBO}} = -\mathbb{E}_{z=q(z|x)} \log p(x|z) - D_{\mathbf{KL}}(q(z|x)||p(z)) \,, \tag{2.7}$$

where $D_{\mathbf{KL}}$ is the Kullback-Liebler distance. The term $D_{\mathbf{KL}}(q(z|x)||p(z))$ can be interpreted as a regularization, that ensures the distribution of embeddings does not deviate too far from a previously determined prior $p(z)$, usually a symmetric zero-mean multivariate Gaussian $\mathcal{N}(\mathbf{0}, I)$. I is the identity matrix. The other term $\mathbb{E}_{z \sim q(z|x)}[\log p(x|z)]$ is usually referred to as reconstruction loss, as it corresponds to a probabilistic version of the term $\mathbb{E}_{x \sim D}[||x - d(e(x))||_2^2]$ in the normal autoencoder.

This VAE was then further extended to the β-VAE by Higgins et al. [23]. They allow for an adjustable trade-off between reconstruction and regularization loss, by changing the loss function to

$$\mathcal{L}_{\texttt{ELBO}} = -\mathbb{E}_{z=q(z|x)} \log p(x|z) - \beta D_{\mathbf{KL}}(q(z|x)||p(z)) \,, \tag{2.8}$$

with $\beta \in \mathbb{R}^+$ controlling the amount of regularization.

Chapter 3

Curriculum Learning with Task Embeddings

The main idea of curriculum learning is motivated by the way humans learn, progressing from easy to more difficult tasks. A common example is that children first learn to crawl, then to walk and finally to run. This idea was introduced to machine learning in 1985 by Selfridge et al. [24] who propose a learning scheme to train a controller for cart pole tasks. It was then significantly popularized again by Bengio et al. in 2009 [25]. They investigate whether curricula are helpful in a simple supervised learning tasks and find a curriculum starting with easy tasks and slowly progressing to difficult tasks to be advantageous. Since then, curriculum learning has been applied to Reinforcement Learning and has shown to be helpful in various settings [11]. If using the right curriculum, it allows a quicker convergence to an optimal performance, or a higher final reward. However, automatically identifying a helpful curriculum is a significant challenge [11]. While methods have been proposed to identify the optimal curriculum, for example by phrasing task selection as an MDP [26], these methods require a large amount of environment interactions to evaluate different curricula before finally using the best found curriculum for training. These methods are helpful if we want to identify the optimal curriculum. If our goal, however, is to solve a target task in the least amount of required environment interactions such approaches are not efficient.

We therefore propose an approach that determines an effective curriculum without training a policy on the task itself. Our method works as follows: We first generate random trajectories in all tasks and then use these to learn a task embedding using a modified VAE. We then determine the order of tasks in this embedding space and finally train a policy using this curriculum.

3.1 Related Work

The related work here is split into two sub-sections: Related work on curriculum learning, and related work on task embeddings and similar topics.

3.1.1 Curriculum Learning

As there are many applications of curriculum learning, such as learning a locomotion policy for uneven surfaces [27] or even in AlphaStar [3], implicitly using curricula through its *AlphaStar League*, we will focus mainly on ways of determining curricula.

Narvekar et al. [26] rephrase curriculum learning into a higher level MDP in which the state corresponds to the tasks previously trained on and the actions correspond to picking a task to train on. Svetlik et al. [28] propose to not only consider sequential curricula, but allow all acyclical graphs as valid curricula. They propose an approach to identify curricula by a measure of transfer potential, however, their measure of transfer potential requires full knowledge of the underlying MDP of each task and is not feasible in complex tasks. Foglino et al. [29] look into finding a task sequence that results in the maximum cummulative reward over the full sequence. However all three approaches still use a large number of training steps to first determine the curriculum, while we aim to learn a curriculum without much environment interaction. Matiisen et al. [30] propose an approach with two agents, a teacher and a student. The teacher successively proposes new tasks that the student learns to solve. They evaluate several different teacher approaches. Sukhbaatar et al. [31] rephrase the teacher-student interaction as self-play. These approaches are both limited to simple grid-worlds and it is not obvious how they could be scaled to more complex tasks that we regard in our work.

In the area of curricula for goal-based RL, Racaniere et al. [32] propose an approach in which one agent trains to set attainable goals and another agent trains to solve them. Zhang et al. [33] propose an approach that chooses goals based on epistemic uncertainty, estimated by the variance of three separate value functions. With ALP-GMM, Portelas et al. [34] propose an approach that chooses which task to train on in a setting where tasks are continuously parameterized. They do so by fitting a GMM to datapoints consisting of the task configuration concatenated with the absolute learning progress on the task. They define absolute learning progress as the difference between the reward on a given task and the reward on the closest already attempted task. POET [35] also consists of a student learning tasks presented by a teacher, however, here the new tasks are generated by a genetic algorithm in which the genes encode the characteristics of the task. These four approaches all have access to a task space from which they can select tasks with known parameterizations. In our setting we only have

access to a fixed set of tasks and do not know their underlying parameters, but instead learn a task space that they should correspond to.

3.1.2 Task Embeddings and Task Relations

While we are not aware of any previous research learning an embedding space over tasks, there is some previous work on identifying distances between tasks.

Ferns et al. [36, 37] define a metric between different states in an MDP based on bisimulation and the Kantorovich and total variation metrics between their transition functions. This was then extended to a metric between full MDPs [38]. Unfortunately this metric requires an accurate model of the value of all states of the MDP and does not scale to complex tasks. Fernandez et al. [39] defines the distance between tasks based on the reward an agent trained on one task can obtain in the other task. This distance is thus limited to very similar environments, and is sensitive even to reward scaling. Lazaric et al. [40] provide a sample based method to calculate a distance between multiple tasks, as well as to identify the current task while interacting with it. However, this approach relies on having an additional model of the environment and iterating over all collected transitions for each new sample from a target environment. It thus does not scale to larger tasks. Sinapov et al. [41] assumes that the different tasks are determined by a set of pre-determined features and then uses the distance between these as distance between their MDPs. This is of course a strong assumption on the structure of the environment, which can only be fulfilled in specifically hand-crafted environments. Wang et al. [42] propose a way to represent MDPs as graphs and then adapt a graph similarity measure, SimRank [43], for usage with MDPs. This similarly requires access to the full MDPs and does not scale to larger tasks.

These approaches could be used to construct a task space, however, as all rely on usually unavailable knowledge over the underlying MDPs or require trained agents, we do not pursue them further. Instead, we present an approach that only uses trajectories from the current agent to determine a task space.

3.2 Variational Task Embedding

We propose a method of learning task embeddings based on variational inference. We use a VAE with a recurrent encoder that receives a full trajectory $T = (s, a, r, s')_{1:L}$ and outputs the parameterization of a Gaussian distribution $q(m|T) = \mathcal{N}(\mu, \Sigma)$. A sample $m = e(T)$, with e being the encoder, is drawn from this distribution and used as additional input to a reward decoder and transition decoder. They correspond to reward function R and transition function P, which together specify our task $\tau = (R_\tau, P_\tau)$. The reward decoder learns

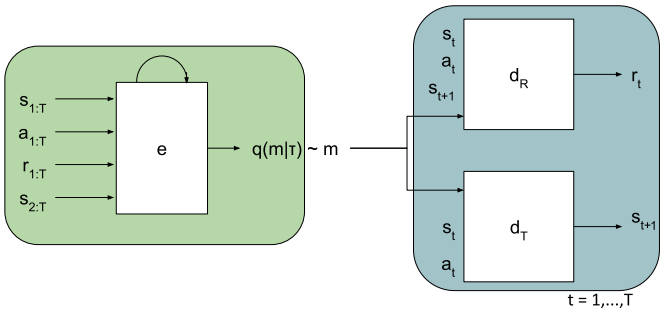


Figure 3.1: Shown is the structure of the sequential VAE used to learn an embedding over tasks. A full trajectory is input into the recurrent encoder, which outputs a parameterization of a normal distribution. From this distribution a sample is drawn representing the current task. This sample is then used as input for two decoders, representing the reward and transition function. In addition the decoders receive the other inputs necessary to predict reward (s, a, s') and transition (s, a).

a function $\hat{r} = d_r(s, a, s', m)$, while the transition decoder learns the function $\hat{s}' = d_s(s, a, m)$. As we defined our tasks to be determined by different reward and transition functions, it is not possible to accurately predict these values without knowledge of the current task. Therefore, in order to minimize the reconstruction loss by accurately predicting the reward and transition, the encoder has to learn a representation of the current task from the given trajectory. While this leads to the embeddings m representing the task, it does not necessarily lead their distances corresponding to dissimilarity. However, as the embeddings are stochastic, placing similar tasks next to each other reduces the expected loss. This is advantageous as, in the cases where a drawn embedding is closer to the mean of a different task than its own, the error of the decoder will be smaller if the tasks are more similar. Because we calculate the Kullback-Leibler distance D_{KL} to a symmetric Gaussian $\mathcal{N}(0, I)$, we can assume the distribution over embeddings of each task to also be approximately symmetric. We can therefore use the euclidean distance $||\cdot||_2$ between task as approximation of their dissimilarity, as we will do in the next section.

The structure of our VAE is shown in Figure 3.1. It is related to the VAE used in VariBAD [44], however, while they provide an embedding for every transition in the trajectory, we only care about one embedding over the full trajectory and train our VAE accordingly. Additionally, we use skip connections in the decoders from the sampled embedding to each hidden layer. This has been shown to lead to more meaningful latent spaces being learned [45]. Details about the VAE and hyper-parameters can be found in Appendix A.1.1.

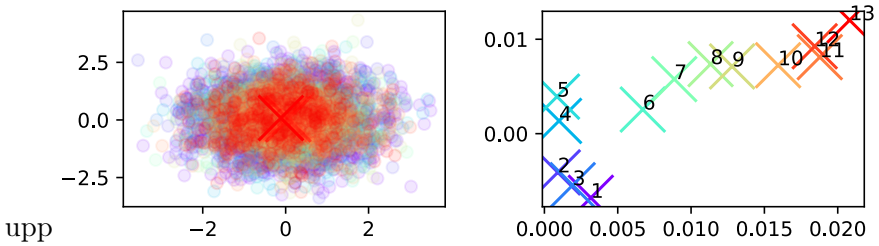


Figure 3.2: Shown is the embedding learned by our VAE on a 2D-gridworld task. The numeric labels of the means correspond to the edge length of the gridworld, we can therefore see that the relation between embeddings corresponds approximately to the true underlying relations between tasks.

3.3 Curriculum Algorithm

Now that we have an embedding for each task, we want to use these embeddings to determine a curriculum. While the embedding gives us an estimate of similarity between tasks, we also need to estimate the learnability of each task, as we want to start in easy tasks and then progress towards our target task. As an easy surrogate for learnability, we use the variance of the episodic return received over multiple trials $l(\tau) = \texttt{Var}[G_\tau(\pi)]$. This is based on the assumption that, in a task that is solved, the agent will always perform well, and in a task which is too difficult the agent will always perform poorly. In learnable, intermediate tasks, however, the agent will sometimes perform well and sometimes fail, leading to a higher reward variance. Another option that we evaluated is to use epistemic uncertainty as a proxy for learnability, as was proposed for goal oriented RL by Zhang et al. [33].

Using the embedding and measure for learnability, we propose two different ways of determining a curriculum.

3.3.1 Greedy Curriculum

The first approach we propose is a simple greedy curriculum, in which we begin at the task with the highest learnability and then successively add the closest (most similar) task till we reach the target task τ^*. We first initialize a policy π and generate N_T random trajectories $\{T_{\tau,1}, ..., T_{\tau,N_T}\}$ on each task $\tau \in \mathcal{T}$, to obtain training data for our VAE. To determine convergence we perform early stopping with a separate set of validation trajectories. We then input all trajectories to the VAE and record the mean x_τ of the mean parameterization μ for each task τ.

$$x_\tau = \frac{1}{N_T} \sum_{i=1}^{N_T} e_\mu(T_{\tau,i}) \,,$$

Algorithm 1: Greedy Automatic Curriculum

Initialize policy π. Generate N_T random trajectories $(T_{\tau,1}, ..., T_{\tau,N})$ on each $\tau \in \mathcal{T}$
Train VAE V with encoder e on all trajectories
for $\tau \in \mathcal{T}$ **do**
  $x_\tau = \frac{1}{N_T} \sum_{i=1}^{i=N_T} e_\mu(T_{\tau,i})$
$\tau \leftarrow \arg\max_\tau \texttt{Var}[R_\tau]$
while target task not solved **do**
  $\tau \leftarrow \arg\min_{\tau' \in \mathcal{T} \setminus \tau} ||\mu_\tau - \mu_{\tau'}||_2^2$
  Train π on τ till solved.
Solved target task, terminate.

with e_μ representing the mean parameterization output by the encoder.

We do not use the output covariance matrix Σ as we did not find it to be helpful in practice. We now have an embedding space over tasks, an example is shown in Figure 3.2 with the means x_τ being shown on the right.

Our curriculum is supposed to start at the easiest task and then lead to the target task among similar tasks. We therefore start at the task with the highest learnability $\tau_0 = \arg\max_\tau l(\tau)$ and then greedily choose the closest task that was not trained on yet $\tau_{t+1} = \arg\min_{\mathcal{T} \setminus \{\tau_0,...,\tau_t\}} ||x_{\tau_t} - x_{\tau'}||_2$ until we reach the target task. This could lead to some issues, such as picking tasks that are more unlike the target task than the previous ones, however, we did not encounter such issues in our experiments. Having determined our sequence of tasks, we train the agent on each task until it manages to solves it or exceeds a maximum number of training steps. We then continue with the next task. This approach is also shown in Algorithm 1.

3.3.2 Sampling Based Approach

A problem of the approach presented above is that it relies heavily on the distances in the initial embedding corresponding to similarities between tasks. If the initial embedding does not achieve this but we keep using it for task selection, we might even require more interactions with the environment than if we directly train on the target task. Additionally, not all given tasks might be helpful to learn the target task. Requiring the curriculum to be a sequence containing all tasks could, therefore, slow down training. We thus propose an alternative, in which we probabilistically sample tasks, in order to make the approach more robust to incorrect or varying embeddings. This is not strictly a curriculum as in Bengio et al. [25] but serves to achieve the same purpose and is in line with recent research on curriculum learning, e.g. [34]. Our goal in this approach is

Algorithm 2: Iterative Sampling-Based Curriculum

Initialize policy π.
while $G_{\tau^*}(\pi_\theta) < G_{\tau^*}(\pi^*)$ **do**
  Generate N_T trajectories with $(T_{\tau,1}, ..., T_{\tau,N})$ on each $\tau \in \mathcal{T}$ with π_θ
  Train VAE V with encoder e on all trajectories T
  for $\tau \in \mathcal{T}$ **do**
    $x_\tau = \frac{1}{N_T} \sum_{i=1}^{i=N_T} e_\mu(T_{\tau,i})$
    Calculate $l(\tau)$
  $p(\tau) \propto (1-d)\mathcal{N}(\bar{x}_{\tau^*}, \hat{\Sigma})(x_\tau) + d\tilde{l}(\tau)$
  $t \leftarrow 0$
  while $t < T_{\texttt{it}}$ **do**
    $\tau \sim p$
    Train π on τ for an episode with length L, update θ
    $t \leftarrow t + L$
Solved target task, terminate.

to sample tasks more frequently that 1) are similar to the target task and 2) are learnable for our policy in its current state.

To do so, we propose an approach that fits a multivariate Gaussian distribution $\mathcal{N}(\mu, \hat{\Sigma})$ with its mean set to the embedding of the target task $\mu = x_{\tau^*}$. We then determine the covariance matrix $\hat{\Sigma}$ as the maximum-likelihood solution given the embedding means of all tasks $\{x_\tau | \tau \in \mathcal{T}\}$. This is inspired by ALP-GMM [34], however, they use their approach to improve generalization and address a setting in which a task space is given, as the tasks are parameterized by continuous variables. Instead, we learn a task space with our VAE, allowing an application to a broader range of tasks. To achieve 2), we combine the probabilities from the GMM with the normalized learnability score $\tilde{l}(\tau) = \frac{l(\tau) - \min_{\tau \in \mathcal{T}} l(\tau)}{\max_{\tau \in \mathcal{T}} l(\tau) - \min_{\tau \in \mathcal{T}} l(\tau)}$. Put together this result in the following sampling probability:

$$p(\tau) \propto (1-d)\mathcal{N}(x_{\tau^*}, \hat{\Sigma})(x_\tau) + d\tilde{l}(\tau) \tag{3.1}$$

where d is a hyperparameter weighting the learnability and GMM values. We retrain the embedding and recalculate the probabilities every N_E training steps. This could therefore be considered an on-policy curriculum and our algorithm is shown in Algorithm 2. This approach could also be seen as a form of active learning, in which the agent picks which tasks to train on [46].

3.4 Experiments

We will first describe the environments used in our experiments, then look at the learned embeddings and finally at the performance of the curriculum.

3.4.1 Environments

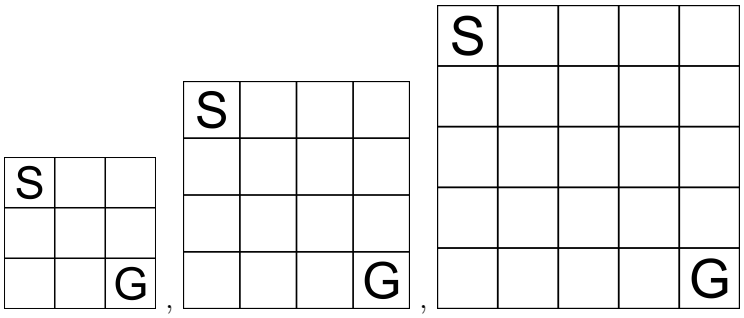


Figure 3.3: Shown are three tasks from our grid-world task set. The agent starts in the top-left and is rewarded only for reaching the goal in the bottom-right corner. While we only show three examples, in our experiments we use 13 different grid-world sizes.

Our first test environment is a two-dimensional grid-world, in which the agent always starts in the top-left corner and has to move to a target location in the bottom right corner. No reward is given, except for a reward of $r = 10$ upon reaching the goal. We use 13 different variations of this environment, with edge-lengths in $\{1, ..., 13\}$. This task is illustrated in Figure 3.3.

Bipedal Walker

As a more complex environment, we also look at a continuous control task known as *BipedalWalker* from the OpenAI gym suite [47] which has previously been used in multi-task and generalization literature [34, 35, 48]. In this environment a simple two-legged robot is placed in a two-dimensional world with the goal being to move to the right with a high velocity. We here use two different sets of tasks. The first set of tasks varies the length of the legs and the spacing of obstacles, but always uses the same default reward function. The second set of tasks always uses the same robot, but proposes different reward functions inspired by track and field events: Jumping up, jumping a long distance, running for three different distances and a hurdle run. Finally, we also add a hurdles task with sparse rewards, only given when crossing a hurdle and the rewards otherwise being $r = 0$. Details about these environments can be found in Appendix B.1.3.

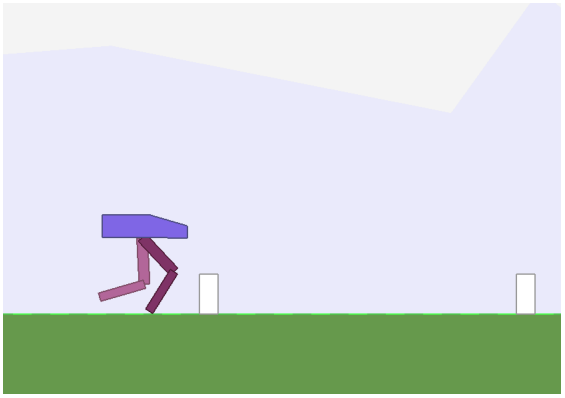


Figure 3.4: Shown is a scene from the *BipedalWalker* task with obstacles. The agent (robot) is rewarded for moving to the right quickly and is punished for falling over or using too much torque.

3.4.2 Gridworld Results

With regards to the embeddings learned by our VAE, our goal is that they represent an underlying similarity between tasks in such a way that the euclidean distance between related tasks is smaller than between unrelated tasks.

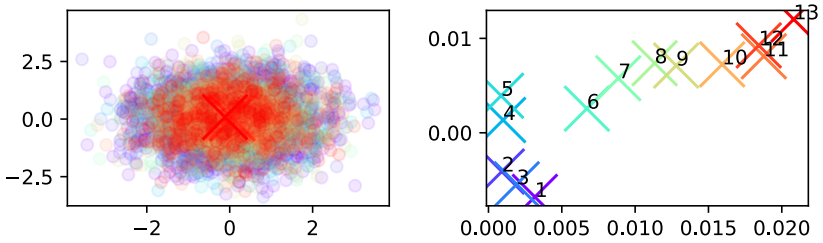


Figure 3.5: Shown are the embeddings learned in a randomly chosen trial on the grid-world set of tasks. On the left the samples generated by the decoder of the VAE are shown. On the right the mean embeddings $\bar{x}_\tau$ are shown for each task. The numeric labels correspond to the edge-lengths of the grid-worlds.

We first look at the embeddings learned in the grid world task. The embeddings from a randomly chosen run are shown in Figure 3.5. The color of each dot signifies which task generated it. If we just look at the drawn embeddings from the left plot, it is difficult to see a pattern. However, if we instead look at the means of the embeddings x_τ, shown in the right plot, we find an expected pattern. We can see that the embedding has captured the meaning of the underlying tasks, at least at a local level, with similar tasks being close to each other, and dissimilar tasks being far away from each other.

The curriculum generated from this embedding is shown in Figure 3.6. While the optimal, or oracle, curriculum would go strictly from smaller to larger tasks

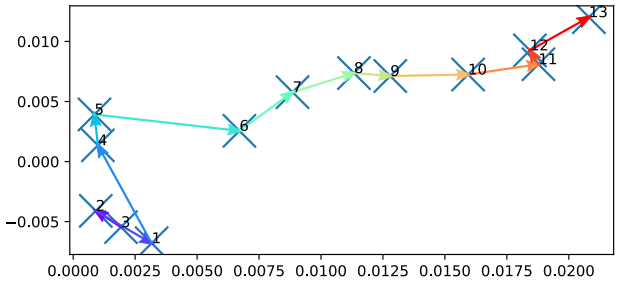


Figure 3.6: Shown is the curriculum determined from the embedding in Figure 3.5. The colors (blue to red) and arrows show the order of the curriculum. The optimal curriculum would be in sequence from 1 to 13, here the determined sequence starts with 3-2-1-4, but is optimal afterwards.

(1,2,...,13), the curriculum generated by our VAE is incorrect for the first two steps, starting as (3,2,1,4,5,...) and then continuing correctly. This kind of mistake is frequent in our experiments. We assume that this is due to the smaller grid-worlds being more similar to each other than the larger ones and therefore being clustered more closely together.

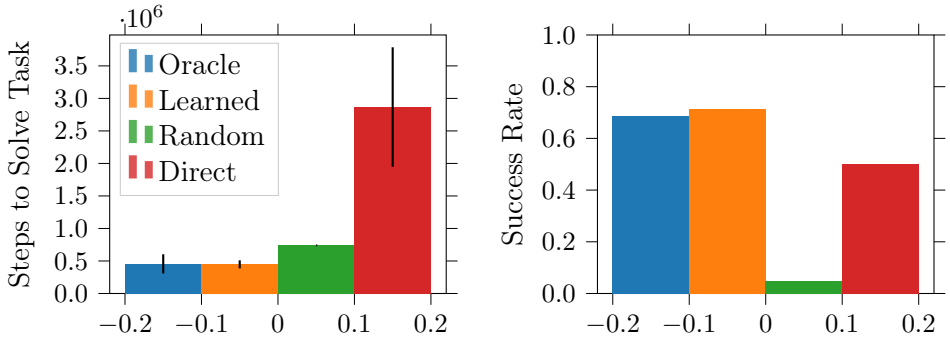


Figure 3.7: Shown are the average number of steps required to solve the target task, as well as percentage of successful trials, averaged across 20 trials each. The error bars show the standard deviation of the mean. We can see that the learned curriculum performs similar to the optimal oracle curriculum while the random curriculum rarely succeeds at the task. The *direct* approach, only training on the target tasks, requires significantly more steps to converge.

Next, we will compare our learned, greedy curriculum approach with three baselines: *Oracle* which has knowledge of the optimal curriculum (1,2,...,13), *Random* which randomly picks a new task that has not been trained on yet, and *Direct* which only trains on the target task. We perform 20 trials with each approach and report the number of steps necessary to solve the target task, as well as the portion of trials in which the approach managed to solve the target task, in the maximum 10 million time-steps. The results are shown in Figure 3.7.

They show that the curriculum learned by our approach performs similarly to the optimal oracle curriculum while the random curriculum rarely solves the target task in the allotted number of steps. Directly training on the target task achieves a similar success rate to using the oracle curriculum, but requires about six times as many environment steps. The VAE was trained with 1.5×10^4 steps per task, for a total of $\approx 2 \times 10^5$ environment steps. These are not accounted for in the bar chart as our aim here is to evaluate the efficacy of the learned curriculum. Adding them, however, does not significantly affect the comparison and the used number of training steps for the VAE could probably be reduced further.

3.4.3 Bipedal Walker Results

Having shown the effectiveness of our approach on a simple grid-world task, we will now evaluate our approach on the more complex Bipedal Walker task.

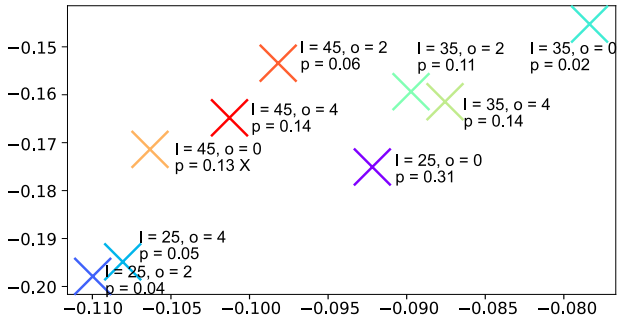


Figure 3.8: Shown is the embedding learned for the BipedalWalker leg length, obstacle spacing taskset. l is the lenght of the legs of the robot, and o is the spacing of the obstacles, with $o = 0$ meaning no obstacles were used. Here we use the sampling based curriculum with p denoting the probability of sampling each task. We can see that tasks closer to the target task ($l = 25, o = 0$) are more likely to be selected.

We begin by looking at the embedding learned by the VAE in the first task set, consisting of different leg lengths and obstacle spacings. The embedding from a randomly chosen trial is shown in Figure 3.8. While we can see a separation into the different leg lengths, it is not as clear as in the grid-world tasks.

As we do not expect a sequence of tasks to be helpful, we here use the sampling-based approach shown in Algorithm 2. The task probabilities in our example trial are also shown in Figure 3.8. The results from our approach and baselines are shown in Figure 3.9. Unfortunately, we found that in this set of tasks no tested curriculum, learned or hand-crafted, lead to an improvement over randomly picking tasks. This can be explained by the agent being able to infer the spacing of obstacles and its leg length from the LIDAR measurements it receives. This leads to the agent being able to learn a good policy for the target

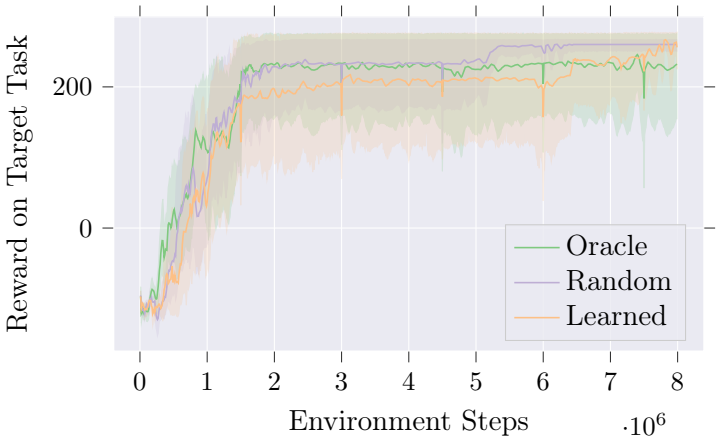


Figure 3.9: Shown is the reward on the target task for our leg length and obstacle dataset. We perform 10 trials per approach and the shaded region shows the 95% CI on the mean. We do not find any curricula to be helpful in this set of tasks.

task, by training on a different task, making the actual task choice less important.

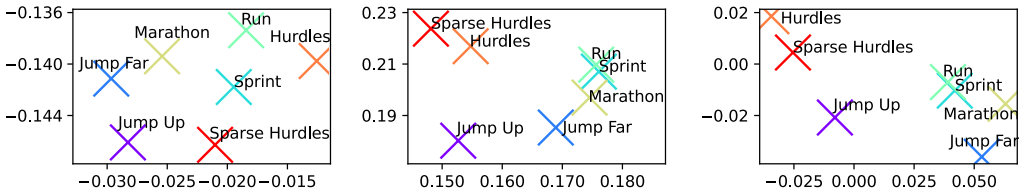


Figure 3.10: Shown are the embeddings for our track and field set of tasks, using random policies (left), intermediate policies (middle) and close to optimal policies (right).

We therefore continue with the second task set, in which we assume that a curriculum starting with running, then hurdles and finally the sparse hurdle task, should be advantageous. The embedding learned in this set of tasks is shown in Figure 3.10 on the left side. We can see that here our approach struggles to separate the tasks meaningfully and the embedding seems mostly random. This might be due to the reward not being very informative when using random policies. To investigate whether the random trajectories are the issue, we also train embeddings for agents achieving medium or high rewards on these tasks, shown in the middle and right of Figure 3.10. Looking at the intermediate and expert policies, we see a meaningful clustering with the running tasks being separate from the jump and hurdles tasks. Further, the two hurdles tasks are clustered together.

The results from our approach are shown in Figure 3.11. We also show the results of the proposed oracle curriculum, uniformly random task sampling and naively training on the target task without using any other tasks. The oracle

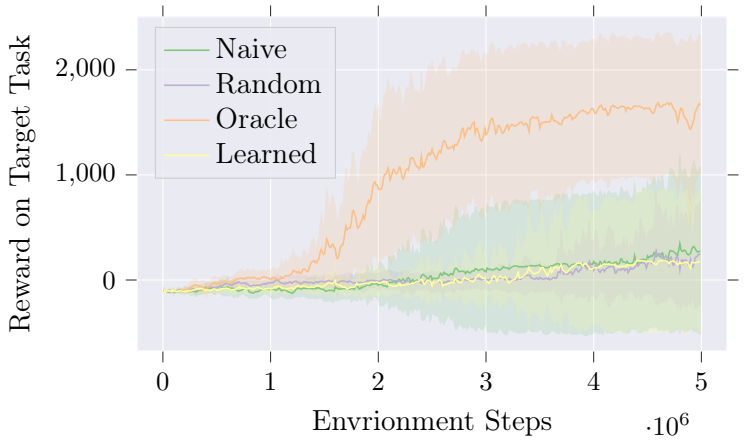


Figure 3.11: Shown are the results from different curriculum methods on the track and field task set. We perform 10 trials per approach and the shaded region shows the 95% CI on the mean. The target task here is the sparse hurdles task.

curriculum converges significantly quicker than all other approaches, while our learned approach and the baselines perform similar. This failure is due to the difficulty of identifying a useful embedding from random trajectories. As we expect the performance of the agent to approach that of intermediate agents with some training, we attempted a variant of our curriculum training in which we retrain the embedding after every N training episodes. Unfortunately, the initially random curriculum prevents our approach from reaching a performance similar to that of the trained agents shown in Figure 3.10. It therefore performed identical to our approach using only the initially determined embedding.

While there are other improvements that could be pursued from this point, such as different sampling approaches, different ways of determining the embedding, and different structures for the agent, we here decided to move on to a different topic. Specifically, the cluster-like structure and failure to learn a useful policy in the presence of conflicting reward functions, as present in the track and field task, motivated us to look into Multi-Task learning with clusters of similar tasks. We investigate this setting in the next chapter. For the curious reader, we list some attempted but ultimately unsuccessful changes to our curriculum approach in the Appendix A.2.

Chapter 4

Expectation-Maximization Task Clustering

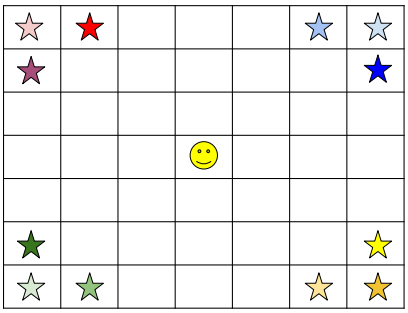


Figure 4.1: An agent (smiley) should reach one of 12 goals (stars) in a grid world. Learning to reach a goal in the top right corner helps him to learn to reach the other goals in the same corner. However, learning to reach a goal in a different corner at the same time leads to conflicting rewards, hindering training. Task clustering resolves the issue.

While in the last chapter our goal was to perform well on a single task, in this chapter we address the problem of multi-task learning in which we are given a set of tasks and wish to perform well on all of them. There are already many approaches that address this issue, as we will explain in more detail in the next section, but they all have one common feature: They all – implicitly or explicitly – assume that the tasks are uniformly similar to each other. However, if we regard more complex, more varied sets of tasks, this does seem like an inappropriate assumption in general. For example, if we look at the tasks in the track and field discipline, as illustrated in Chapter 1, we can clearly see that some tasks are more similar than others. Assuming that these tasks are all equally similar would be inaccurate and can slow down training. Negative transfer between tasks with contradicting rewards might even prevent convergence, as shown in the *BipedalWalker* experiments in Section 3.4.3. However, learning a full task space, for example with our VAE, is difficult in some settings.

Therefore, we aim to find a less strict representation of task similarity. Instead of wanting distances between tasks to correspond to similarity, we only aim to identify subsets of similar tasks that can be learned together. By doing so, we rephrase this issue as a clustering problem, in which we want to train a policy for each cluster. To solve this problem we propose an approach inspired by the expectation-maximization framework. We begin by initializing a set of policies corresponding to the presumed number of clusters. We then iteratively evaluate this set of policies on all tasks, assign tasks to policies based on their respective performance and train policies on their assigned tasks. This leads to policies naturally specializing to clusters of related tasks, yielding an interpretable decomposition of the full task set. Further, our approach naturally avoids negative transfer and can thereby improve the learning speed and final reward in multi-task RL settings.

4.1 Related Work

Here we will provide an overview over related work, split into three sections: EM in RL, task clustering and general multi-task RL.

4.1.1 Expectation Maximization in Reinforcement Learning

Expectation-Maximization (EM) has previously been used in RL to directly learn a policy. By reformulating RL as an inference problem with a latent variable, it is possible to use EM to find the maximum likelihood solution, corresponding to the optimal policy. We direct the reader to Deisenroth et al. [49] for a survey on the topic. Our approach is different: We use an EM-inspired approach to cluster tasks in a multi-task setting and rely on recent RL algorithms to learn the tasks.

4.1.2 Task Clustering

In supervised learning, the idea of subdividing tasks into related clusters was first proposed by Thrun et al. [50]. They use a distance metric based on generalization accuracy to cluster tasks. A variety of other methods have been proposed in the supervised learning literature, for brevity we direct the reader to the survey by Zang et al.[10], which provides a good overview of the topic. Our work differs in that we focus on RL, where no labeled data set exists.

In RL, task clustering has in the past received attention in works on transfer learning. Carroll and Seppi [51] proposed to cluster tasks based on a distance function. They propose distances based on Q-values, reward functions, optimal policies or transfer performance. They propose to use the clustering to guide transfer. Similarly, Mahmud et al. [52] propose a method for clustering Markov

Decision Processes (MDPs) for source task selection. They design a cost function for their chosen transfer method and derive an algorithm to find a clustering that minimizes this cost function. Our approach differs from both in that we do not assume knowledge of the underlying MDPs and corresponding optimal policies. Furthermore, the general nature of our approach allows it to scale to complex tasks, where comparing properties of the full underlying MDPs is not feasible. We also do not use the clustering for source selection, but instead use it during training to improve learning speed in a multi-task setup.

4.1.3 Multi-Task Reinforcement Learning

Related research on multi-task RL can be split into two categories: works that focus on very similar tasks with small differences in dynamics and reward, and works that focus on very dissimilar tasks. In the first setting, approaches have been proposed that condition the policy on task characteristics identified during execution. Lee et al. [53] use model-based RL and a learned embedding over the local dynamics as additional input to their model. Yang et al. [54] train two policies, one that behaves in a way that allows an identification of the environment dynamics and another policy that uses an embedding over the transitions generated by the first as additional input. Zhang et al. [55] treat multi-task RL as a hidden-parameter-MDP and learn a model over the dynamics using Bisimulation to provide performance guarantees. They evaluate on MuJoCo tasks with varying masses, friction, foot length or finger size. Zintgraf et al. [44] train an embedding over the dynamics that accounts for uncertainty over the current task during execution and condition their policy on it.

Our approach is more general than these methods as our assumption on task similarity is weaker. Another set of works focuses on a setting in which just the reward changes. This is also referred to multi-goal RL. For example, some papers use successor features to get a way to easily do policy evaluation [56]. Hansen et al. [57] combine this with variational inference to be able to scale to complex Atari games. Zhang et al. [33] use value disagreement to select which goals to sample for training their policy. Our approach is more general than these works, as it allows for both differing dynamics and rewards.

In the second group of papers, the set of tasks is more diverse. Most approaches here are searching for a way to reuse representations from one task in the others. Riemer et al. [58] present an approach to learn hierarchical options, and use it to train an agent on 21 Atari tasks. They use the common NatureDQN network [14] with separate final layers for option selection policies, as well as separate output layers for each task to account for the different action spaces. They investigate the option frequencies per task and find some similarities between games. Eramo et al. [59] show how a shared representation can speed up training. They then use a network strucuture with separate heads for each task but

share the hidden layers. They evaluate their approach on MuJoCo tasks [60] and show an advantage compared to using randomly initialized policies. Our multi-head baseline is based on these works. Bram et al. [61] propose a method that addresses negative transfer between multiple tasks by learning an attention mechanism over multiple sub-networks. However, as all tasks yield experience for one overarching network, their approach still suffers from interference between tasks. We limit this interference by completely separating policies. They focus on simple discrete tasks and do not assume a setting with clusters of similar tasks. Wang et al. [48] address the problem of open-ended learning in RL by iteratively generating new environments. Similar to us, they use policy rankings as a measure of difference between tasks. However, they use this ranking as a measure of novelty to find new tasks, addressing a very different problem. Hessel et al. [62] present PopArt for multi-task deep RL. They address the issue that different tasks may have significantly different reward scales. Sharma et al. [46] look into active learning for multi-task RL on Atari tasks. They show that uniformly sampling new tasks is suboptimal and propose different sampling techniques. Yu et al. [63] propose Gradient Surgery, a way of projecting the gradients from different tasks to avoid interference. These last three approaches also address learning games in Atari, but they are orthogonal to our work and could be combined with EM-clustering. We see this as an interesting direction for future work.

4.2 Approach

As the growing body of literature on meta-, transfer- and multi-task learning suggests, we can expect a gain through positive transfer if we train a single policy π_i on a set of related tasks $\mathcal{T}_k \subset \mathcal{T}$. On the flip side, the policy π_i might perform poorly on tasks $\tau \notin \mathcal{T}_k$. Moreover, training policy π_i on a task $\tau \notin \mathcal{T}_k$ might even lead to a decrease in performance on the task set $\mathcal{T}_k$ through negative transfer. We incorporate these insights into our algorithm by modeling the task set $\mathcal{T}$ as a union of K disjoint task clusters $\mathcal{T}_1, \dots, \mathcal{T}_K$, i.e., $\mathcal{T} = \bigcup_{k=1}^{K} \mathcal{T}_k$ with $\mathcal{T}_i \cap \mathcal{T}_j = \emptyset$ for $i \neq j$. Tasks within a cluster allow for positive transfer while the relationship of tasks of different clusters is unconstrained. Tasks in different clusters can therefore even have conflicting objectives. Note that the assignment of tasks to clusters is not given to us and therefore needs to be inferred by the algorithm. Note also that this formulation only relies on minimalistic assumptions. It is therefore applicable to a much broader range of settings than many sophisticated models with stronger assumptions. As generality is one of our main objectives, we see the minimalistic nature of the model as a strength rather than a weakness.

Given this problem formulation we note that it reflects a clustering problem, in which we have to assign each task $\tau \in \mathcal{T}$ to one of the clusters $\mathcal{T}_k$, $k \in \{1, \dots, K\}$. At the same time, we want to train a set of policies $\{\pi_1, ..., \pi_n\}$ to

solve the given tasks. Put differently, we wish to infer a hidden latent variable (cluster assignment of the tasks) while optimizing our model parameters (set of policies). An Expectation-Maximization (EM) [12] inspired algorithm allows us to do exactly that. On a high level, in the expectation step (E-step) we assign each of the tasks $\tau \in \mathcal{T}$ to a policy π_i representing cluster $\mathcal{T}_i$. We then train the policies in the maximization step (M-step) on the tasks they got assigned, specializing the policies to their clusters. These steps are alternatingly repeated — one benefiting from the improvement of the other in the preceding step — until convergence.

Algorithm 3: EM-Task-Clustering

Initialize n policies $\{\pi_1, ..., \pi_n\}$
while not converged do
  ▷ E-Step
  $\mathcal{T}_i \leftarrow \emptyset$ for $i \in \{1, \ldots, n\}$
  for $\tau \in \mathcal{T}$ **do**
    $k \leftarrow \arg\max_i G_\tau(\pi_i)$
    $\mathcal{T}_k \leftarrow \mathcal{T}_k \cup \tau$
  $\mathcal{T}_i \leftarrow \mathcal{T}$ where $\mathcal{T}_i = \emptyset$
  ▷ M-Step
  for $\pi_i \in \{\pi_1, ..., \pi_n\}$ **do**
    $t \leftarrow 0$
    while $t < T_M$ **do**
      $\tau \sim \mathcal{T}_i$
      Train π_i on τ for an episode of L steps
      $t \leftarrow t + L$

Given this general framework we are left with determining the details. Specifically, how to assign tasks to which policies (E-step) and how to allocate training time from policies to assigned tasks (M-step).

For the assignment in the E-step we want the resulting clusters to represent clusters with positive transfer. Given that policy π_i is trained on a set of tasks $\mathcal{T}_i$ in a preceding M-step, we can base our assignment of tasks to π_i on the performance of π_i: Tasks on which π_i performs well likely benefited from the preceding training and therefore should be assigned to the cluster of π_i. Specifically, we can evaluate each policy $\pi_i \in \{\pi_1, \ldots, \pi_n\}$ on all tasks $\tau \in \mathcal{T}$ to get an estimate of $G_\tau(\pi_i)$ and base the assignment on this performance evaluation. To get to an implementable algorithm we state two additional desiderata for our assignment: (1) We do not want to constrain cluster sizes in any way as clusters can be of unknown, non-uniform sizes. (2) We do not want to constrain the diversity of the tasks. This implies that the assignment has to be independent of the reward scales of the tasks, which in turn limits us to assignments based on the relative performances of the policies $\pi_1, ..., \pi_n$. We found a greedy assignment — assign-

ing each task to the policy that performs best — to work well. A soft assignment based on the full ranking of policies might be worth exploring in future work.

In the M-step, we take advantage of the fact that clusters reflect positive transfer, i.e., training on some of the assigned tasks should improve performance on the whole cluster. We can therefore randomly sample a task from the assigned tasks and train on it for one episode before sampling the next task. Overall we train each policy for a fixed number of updates T_M in each M-step with T_M independent of the cluster size. This independence allows us to save environment interactions as larger clusters benefit from positive transfer and do not need training time proportional to the number of assigned tasks.

Note that the greedy assignment (and more generally any assignment fulfilling desiderata 1 above) comes with a caveat: Some policies might not be assigned any tasks. In this case we sample the tasks to train these policies from all tasks $\tau \in \mathcal{T}$, which can be seen as a random exploration of possible task clusters. This also ensures that, early on in training, every policy gets a similar amount of initial experience. For reference, we provide a simplified pseudo code of our approach in Algorithm 3. Note that our approach is independent of the RL algorithm used to train the policies in the M-step and can therefore be combined with any state-of-the-art RL algorithm.

4.3 Experiments

As a proof of concept we start the evaluation of our approach on two discrete tasks. The first environment consists of a chain of discrete states in which the agent can either move to the left or to the right. The goal of the agent is placed either on the left end or the right end of the chain. This gives rise to two task clusters, where tasks within a cluster differ in the frequency with which the agent is rewarded on its way to the goal. The second environment reflects the 2-dimensional grid-world presented in Figure 4.1. Actions correspond to the cardinal directions in which the agent can move and the 12 tasks in the task set $\mathcal{T}$ are defined by their respective goal. We refer an interested reader to Appendix B.1.1 for a detailed description of the environments.

We train policies with tabular Q-learning [13] and compare our approach to two baselines: In the first we train a single policy on all tasks. We refer to this as SP (Single Policy). In the other we train a separate policy per task. This is referred to as PPT (Policy per Task). Our approach is referred to as EM (Expectation-Maximization).

The learning curves as well as the task assignment over the course of training are shown in Figure 4.2 and Figure 4.3. Looking at the assignments, we see that in both environments our approach converges to the natural clustering, leading to a higher reward after finding these assignments. Both our EM-approach and

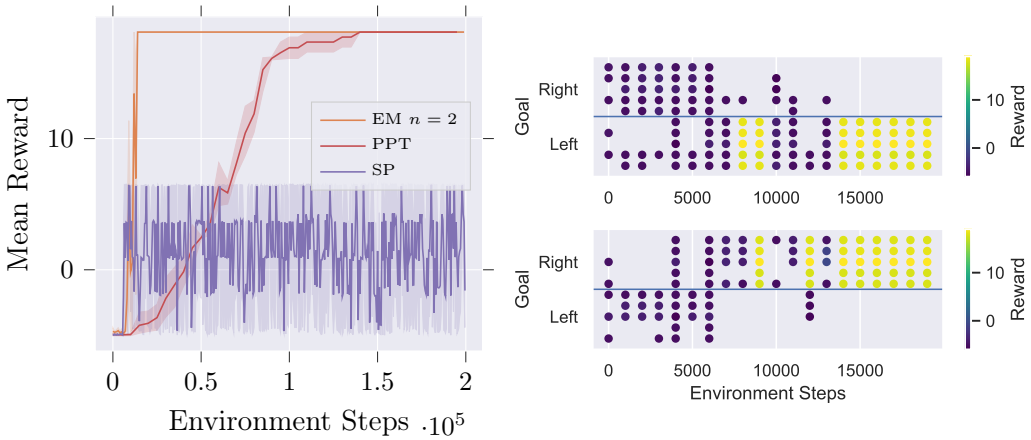


Figure 4.2: **Left:** Mean reward and 95% confidence interval (shaded area) from 10 trials when training on the chain environment. **Right:** Task assignment (dots) and task specific reward (color) over the course of training the two policies in our approach. Each plot shows one of the policies/estimated clusters. The assignments converge to the natural clustering reflected by the goal location.

PPT converge to an optimal reward in the chain environment, and a close to optimal reward in the corner-grid-world. However, PPT requires a significantly higher amount of environment steps to reach this performance, as it does not share information between tasks and therefore has to do exploration for each task separately. SP fails to achieve a high reward due to the different tasks providing contradicting objectives.

4.3.1 Pendulum

Next we consider a simple continuous control environment where tasks differ in their dynamics. We use the pendulum gym task [47], in which a torque has to be applied to a pendulum to keep it upright. Here the environment is the same in all tasks, except for the length of the pendulum which is varied in the range $\{0.7, 0.8, ..., 1.3\}$, giving a total of 7 tasks.

To train our agent, we use TD3 [20] with hyperparameters optimized as discussed in Appendix B.1.2. We compare against SP, PPT, a multi-head network structure similar to the approach used by Eramo et al. [59], and a multi-output network structure as common in related work [62]. Each policy in our approach uses a separate replay buffer. The multi-head network has a separate replay-buffer and a separate input and output layer per task, referred to as *Multi-Head*. The multi-output network also has a separate replay buffer and output layer per task, but uses the same input layer. We refer to it as *Multi-Out*. We assume that Multi-Head should perform better in settings with more diverse tasks, as it has a

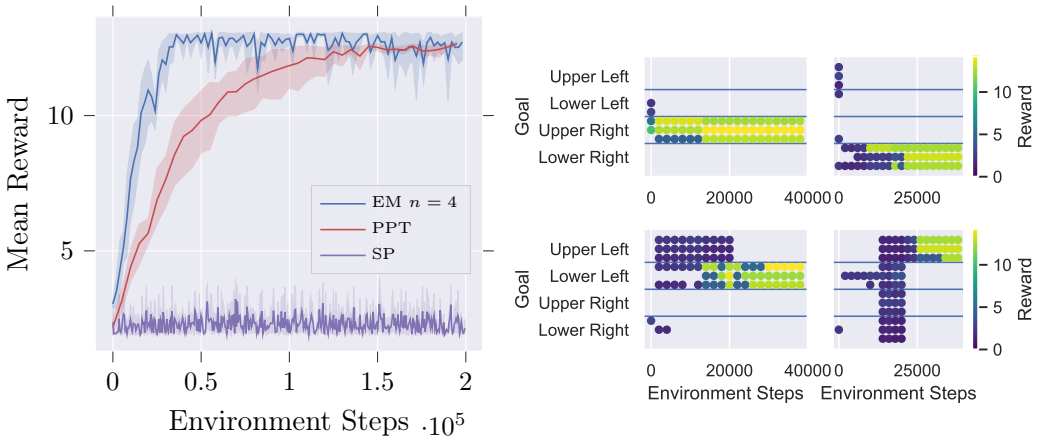


Figure 4.3: **Left:** Mean reward and 95% confidence interval (shaded area) from 10 trials when training on the grid-world environment depicted in Figure 4.1. **Right:** Task assignment (dots) and task specific reward (color) over the course of training for the $n = 4$ policies (estimated clusters) in our approach. The assignment naturally clusters the tasks of each corner together.

more task specific structure. Accordingly, we assume that Multi-Out will perform better if all tasks are very similar to each other. We adjust the network size of the Multi-Head and Multi-Out baselines to avoid an advantage of our method due to a higher parameter count, see Appendix B.1.2 for details.

The results are shown in Figure 4.4. We again observe that our approach clusters similar tasks together, leading to a better performance than the SP agent and a faster convergence than PPT. Multi-Out initially performs worse than EM $n = 2$, and has a similar final performance to EM $n = 4$. However, it performs slightly better than our approach during an intermediate period. Multi-Head requires more experience to converge than our approach in this setup, even more than the PPT approach. We believe this is due to the inherent interference of learning signals in the shared layers. The cluster assignment in our approach is also intuitive, with two clusters focusing on the extremes (cf. Figure 4.4).

4.3.2 Bipedal Walker

As a more complex continuous control environment we again use the *Bipedal-Walker* from the OpenAI Gym [47]. It consists of a bipedal robot in a two-dimensional world, where the default task is to move to the right with a high velocity. The action space consists of continuous torques for the hip and knee joints of the legs and the state space consists of joint angles and velocities, as well as hull angle and velocity and 10 lidar distance measurements.

To test our approach, we reuse the tasks from Section 3.4.1, consisting of 6

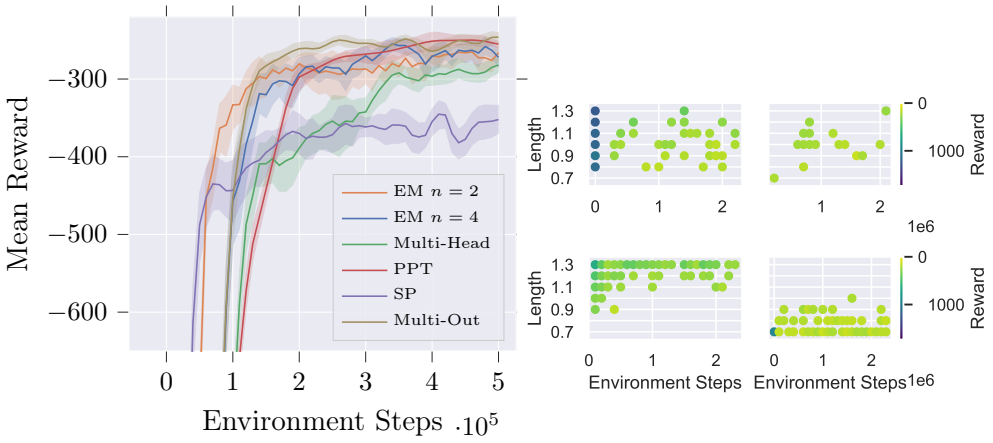


Figure 4.4: **Left:** Mean reward and 95% confidence interval (shaded area) from 10 trials when training on the pendulum environment. The curves are smoothed by a rolling average to dampen the noise of the random starting positions. **Right:** Task assignment (dots) and task specific reward (color) from a sample run. Two policies focus on long and short, while the others focus on medium lengths.

tasks inspired by track and field sports: Jumping up at the starting position, jumping forward as far as possible, a short, medium and long run and a hurdle run. We also reuse the second set of tasks, consisting of 9 tasks with varied leg length as well as spacing of obstacles. This task set is inspired by task sets in previous work [34]. Note that we keep the objective — move forward as fast as possible — constant here. We again use TD3 and tune the hyperparameters of the Multi-Head baseline and our approach (with $n = 4$ fixed) with grid-search. Experiment details and hyperparameters are given in Appendix B.1.3.

The results in Figure 4.5 (left) on the track and field tasks show a significant advantage in using our approach over Multi-Head or SP and a slightly better initial performance than PPT, with similar final performance. SP fails to learn a successful policy altogether due to the conflicting reward functions. Multi-Out performs comparably to our approach. In contrast, the results in Figure 4.5 (right) from the second task set show that SP can learn a policy that is close to optimal on all tasks here. Multi-Head and PPT approaches suffer in this setup as each head/policy only gets the experience from its task and therefore needs more time to converge. Our approach can take advantage of the similarity of the tasks. Multi-out performs slightly worse than our approach during the majority of training, and achieves a lower final performance. We note that the experiments presented here reflect two distinct cases: One in which it is advantageous to separate learning, reflected by PPT outperforming SP, and one where it is better to share experience between tasks, reflected by SP outperforming PPT. Our approach demonstrates general applicability as it performs competitively in both.

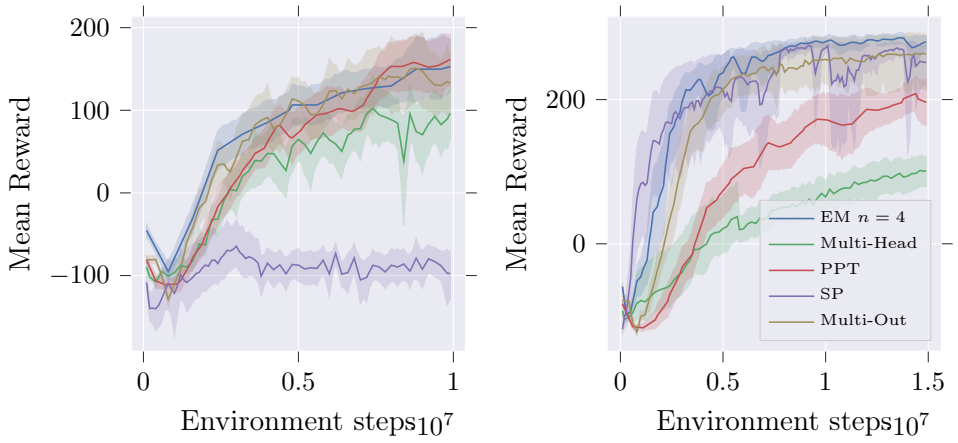


Figure 4.5: Evaluation of the BipedalWalker experiments. The shaded areas show the 95% confidence interval on the mean task reward. **Left:** Track and field task set; 6 tasks with varying objectives. Results reflect 20 trials of each approach. **Right:** Task set with varying leg lengths and obstacles; 9 tasks with the same reward function. Results reflect 10 trials of each approach.

We provide an insight into the assignment of tasks to policies in Appendix B.2.1.

4.3.3 Atari

To test the performance of our approach on a more diverse set of tasks, we evaluate on a subset of the Arcade Learning Environment (ALE) tasks [64]. Our choice of tasks is similar to those used by [58], but we exclude tasks containing significant partial-observability. This is done to reduce the computational burden as those tasks usually require significantly more training data. We built our approach on top of the IQN implementation in the Dopamine framework [16, 65]. We chose IQN due to its sample efficiency and the availability of an easily modifiable implementation. As the different ALE games have different discrete action spaces, we use a separate final layer and a separate replay buffer for each game in all approaches. We use the hyperparameters recommended by [65], except for a smaller replay buffer size to reduce memory requirements. We evaluate our approach with the number of policies set to $n = 4$ and $n = 8$. We choose the size of the network such that each approach has the same number of total tunable parameters. We provide the details in Appendix B.1.4.

The results, in form of the median human-normalized rewards, are given in Figure 4.6. Our EM approach performs similar to PPT and does not show a large difference between using $n = 4$ and $n = 8$. We note also that the Multi-Head approach is unable to learn any useful policy here due to negative transfer

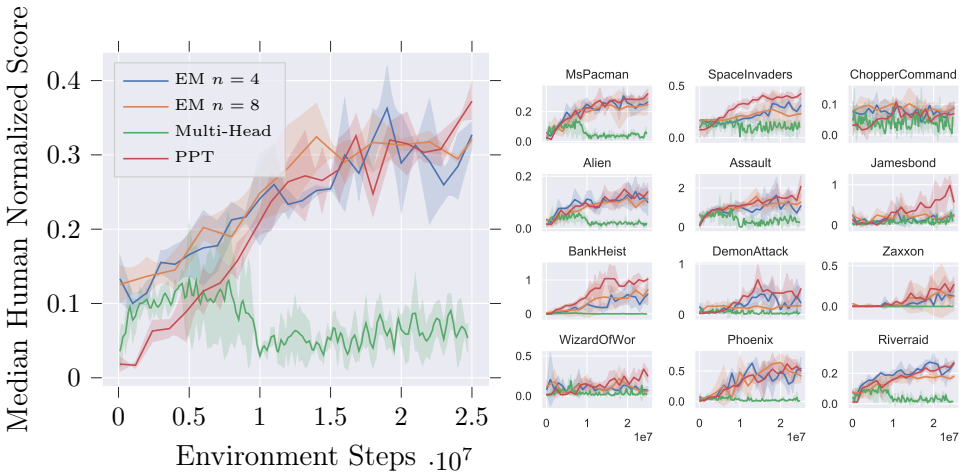


Figure 4.6: Shown are the results of our experiments on a subset of the Atari Learning Environment games. The median reward across games is calculated for each trial, then the mean of this value is taken across 3 trials. The shaded region shows the standard deviation of this mean. On the right we show the mean reward per task.

between tasks. This is in line with experiments in other research [62]. Our approach manages to overcome most of this negative interference, even with just 4 clusters. However, while it is able to avoid negative transfer, we do not see a large benefit from sharing experience between tasks, except in the very beginning of the training. We also note that our approach performs similar to PPT in most games. A notable exception is JamesBond, where PPT significantly outperforms our approach. Task assignments in our approach are given in Appendix B.2.2, as well as an additional discussion of the mean human-normalized reward.

4.3.4 Ablations

To gain additional insights into our approach, we perform three ablation studies on the discrete corner-grid-world environment and the pendulum environment.

First, we investigate the performance of our approach for different numbers of policies n. The results in Figure 4.7 show that using too few policies can lead to a worse performance, as the clusters cannot distinguish the contradicting objectives. On the other hand, using more policies than necessary increases the number of environment interactions required to achieve a good performance in the pendulum task, but does not significantly affect the final performance.

As a second ablation, we are interested in the effectiveness of the clustering. It might be possible that simply having fewer tasks per policy is giving our approach

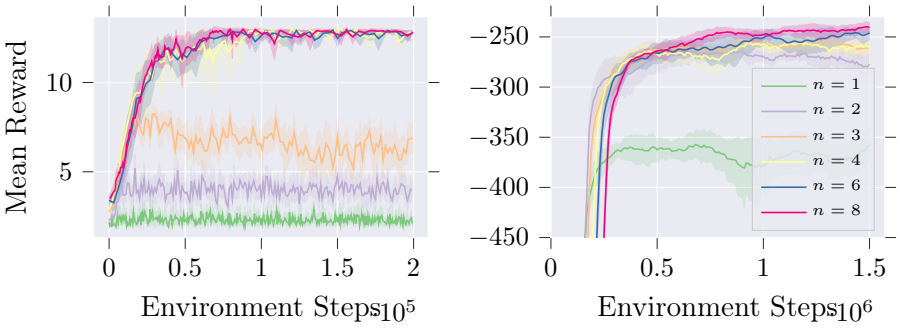


Figure 4.7: Ablations for different number of policies n. Shaded areas show the 95% confidence interval of the mean reward from 10 trials each. **Left:** Corner-grid-world tasks. **Right:** Pendulum tasks, learning curves smoothed.

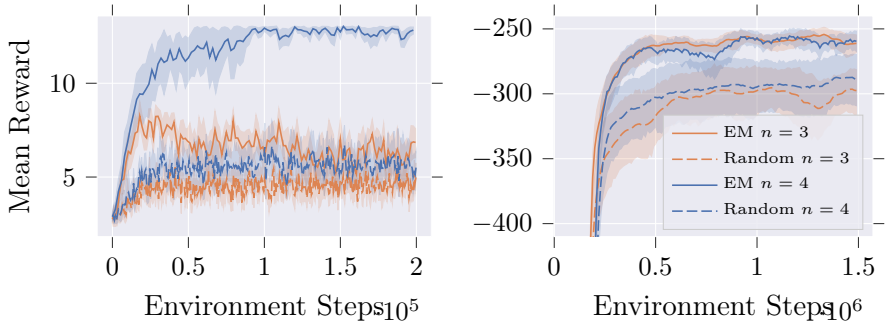


Figure 4.8: Comparison of our approach against randomly assigning tasks to policies at the start of training. Shaded areas show the 95% confidence interval of the mean reward. **Left:** Corner-grid-world tasks, 10 trials each. **Right:** Pendulum tasks, 10 trials each, learning curves smoothed.

an advantage compared to SP or Multi-Head. We therefore provide an ablation in which task-policy assignments are determined randomly at the start and kept constant during the training. Results from this experiment can be seen in Figure 4.8. The results show that using random clusters performs significantly worse than using the learned clusters. This highlights the importance of clustering tasks meaningfully.

In our EM-approach we use a separate replay buffer for each policy, while in the Multi-Out and Multi-Head baselines we use a separate replay buffer for each task. The baselines might therefore have an advantage, as in our approach the assignments frequently switch during the earlier iterations. This leads to transitions being added to the replay buffers that do not match the tasks the policy is assigned to later on. These transitions transitions are then not used to train a policy that might later be assigned to this task, slowing training an hindering specialization.

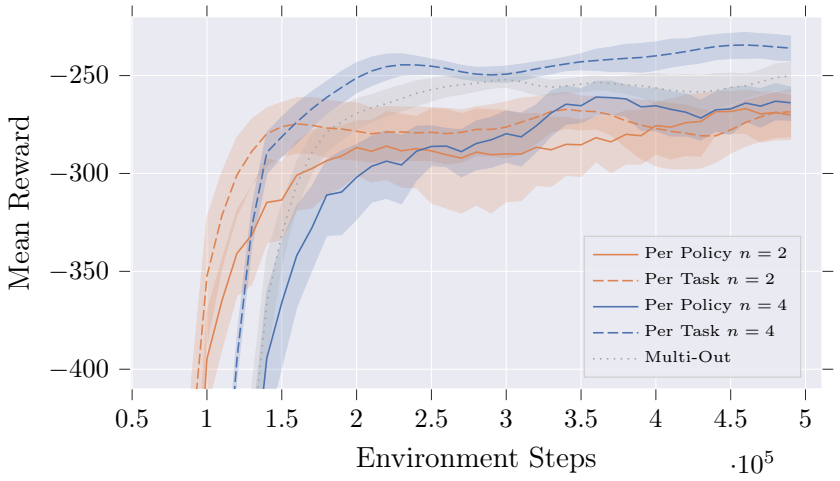


Figure 4.9: Comparison of using a separate replay buffer per policy or per task. We show 10 trials each with smoothed learning curves, shaded regions show the 95% confidence intervals of the mean. We also show the Multi-Out baseline for comparison.

We therefore investigate an ablation of our approach, in which we use a replay buffer per task instead of per policy. We investigate this on our Pendulum environment with $n = 2$ and $n = 4$ policies and show the results in Figure 4.9. This experiment shows a clear advantage of using a replay buffer per task instead of per policy. The difference is significantly larger for $n = 4$ than it is for $n = 2$. This can be explained by the transitions being split between four policies instead of two, therefore more transitions being "wasted". For comparison we also show the performance of the Multi-Out baseline in gray. While our EM approaches with buffers per policy perform similar to this baseline, $n = 4$ with a policy per task performs significantly better.

Chapter 5

Conclusion

In this work, we have proposed two ways of identifying and using relations between tasks in Reinforcement Learning (RL). In the first approach, we use a variational autoencoder to learn an embedding over tasks in which distances correspond to similarities. We then use this embedding in the setting of Curriculum Learning to identify a sequence of tasks that lets us quickly learn a given target task. We combine the embeddings with a heuristic for task learnability and use these measures to automatically determine the curriculum. While we show that this approach works well in simple discrete tasks, it fails to perform well in more complex environments.

In the second setting, we look at Multi-Task RL in which the goal is to train an agent that performs well on all tasks. We present an approach inspired by expectation-maximization that automatically clusters tasks into related subsets, avoiding negative transfer. Our approach uses a set of policies and alternatingly evaluates the policies on all tasks, assigns each task to the best policy, and trains policies on their assigned tasks. Since our approach can be combined with any underlying RL method, we evaluate it on a diverse set of environments. We show its performance on sets of simple discrete tasks, simple continuous tasks, two complex continuous control task sets and a set of Atari games. We show that our approach is able to identify clusters of related tasks and use this structure to achieve a competitive or superior performance in all experiments when compared to other methods. We further performed additional ablations in order to provide insights into how our approach works.

As future work, we plan to investigate different assignment strategies in our clustering approach.

Bibliography

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction.* MIT Press, 2017.

[2] 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," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[3] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[4] OpenAI, :, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with large scale deep reinforcement learning," 2019.

[5] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-End Training of Deep Visuomotor Policies," *J. Mach. Learn. Res.*, vol. 17, pp. 1334–1373, 2016.

[6] E. Yom-Tov, G. Feraru, M. Kozdoba, S. Mannor, M. Tennenholtz, and I. Hochberg, "Encouraging Physical Activity in Patients With Diabetes: Intervention Using a Reinforcement Learning System," *J. Med. Internet Res.*, vol. 19, no. 10, p. e338, 2017.

[7] N. Lazic, T. Lu, C. Boutilier, M. Ryu, E. Wong, B. Roy, and G. Imwalle, "Data center cooling using model-predictive control," in *NeurIPS*, 2018.

[8] A. Satariano and C. Metz, "A Warehouse Robot Learns to Sort Out the Tricky Stuff," *New York Times*, 2020.

[9] J. Kober, J. Andrew Bagnell, and J. Peter, "Reinforcement Learning in Robotics: A Survey," in *Springer Tracts Adv. Robot.*, 2014, vol. 97, pp. 9–67.

[10] Y. Zhang and Q. Yang, "A Survey on Multi-Task Learning," 2017. [Online]. Available: https://arxiv.org/abs/1707.08114

[11] R. Portelas, C. Colas, L. Weng, K. Hofmann, and P.-Y. Oudeyer, "Automatic Curriculum Learning For Deep RL: A Short Survey," 2020.

[12] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.

[13] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, 1989.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. a. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, pp. 529–533, 2015.

[15] M. G. Bellemare and W. Dabney, "A Distributional Perspective on Reinforcement Learning," in *ICML*, 2017.

[16] W. Dabney, G. Ostrovski, D. Silver, and R. Munos, "Implicit quantile networks for distributional reinforcement learning," in *ICML*, 2018.

[17] R. S. Sutton, D. Mcallester, S. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," in *NeurIPS*, 1999.

[18] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," in *ICML*, 2014.

[19] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous Control with Deep Reinforcement Learning," in *ICLR*, 2016.

[20] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," in *ICML*, 2018.

[21] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science (80-.).*, vol. 313, no. 5786, pp. 504–507, 2006.

[22] D. P. Kingma and M. Welling, in *Auto-encoding variational bayes*, 2014.

[23] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "B-VAE: Learning basic visual concepts with a constrained variational framework," *ICLR*, 2017.

[24] O. G. Selfridge, R. S. Sutton, and A. G. Barto, "Training and Tracking in Robotics," in *IJCAI*, 1985.

[25] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, in *ICML*, New York, New York, USA, 2009.

[26] S. Narvekar and P. Stone, "Learning curriculum policies for reinforcement learning," in *AAMAS*, 2019.

[27] Z. Xie, H. Y. Ling, N. H. Kim, and M. van de Panne, "ALLSTEPS: Curriculum-driven Learning of Stepping Stone Skills," 2020. [Online]. Available: http://arxiv.org/abs/2005.04323

[28] M. Svetlik, M. Leonetti, J. Sinapov, R. Shah, N. Walker, and P. Stone, "Automatic curriculum graph generation for reinforcement learning agents," in *AAAI*, 2017.

[29] F. Foglino, C. C. Christakou, R. L. Gutierrez, and M. Leonetti, "Curriculum learning for cumulative return maximization," in *IJCAI*, 2019.

[30] T. Matiisen, A. Oliver, T. Cohen, and J. Schulman, "Teacher-Student Curriculum Learning," *IEEE Trans. Neural Networks Learn. Syst.*, 2019.

[31] S. Sukhbaatar, Z. Lin, I. Kostrikov, G. Synnaeve, A. Szlam, and R. Fergus, "Intrinsic motivation and automatic curricula via asymmetric self-play," *ICLR*, 2018.

[32] S. Racaniere, A. K. Lampinen, A. Santoro, D. P. Reichert, V. Firoiu, and T. P. Lillicrap, "Automated Curricula Through Setter-Solver Interactions," in *ICLR*, 2020.

[33] Y. Zhang, P. Abbeel, and L. Pinto, "Automatic Curriculum Learning through Value Disagreement," 2020. [Online]. Available: http://arxiv.org/abs/2006.09641

[34] R. Portelas, C. Colas, K. Hofmann, and P.-Y. Oudeyer, "Teacher algorithms for curriculum learning of Deep RL in continuously parameterized environments," in *CoRL*, 2019.

[35] R. Wang, J. Lehman, J. Clune, and K. O. Stanley, "Paired Open-Ended Trailblazer (POET): Endlessly Generating Increasingly Complex and Diverse Learning Environments and Their Solutions," 2019. [Online]. Available: http://arxiv.org/abs/1901.01753

[36] N. Ferns, P. Panangaden, and D. Precup, "Metrics for finite markov decision processes," *Proc. Natl. Conf. Artif. Intell.*, 2004.

[37] N. Ferns, P. Panangaden, and D. Precup, "Metrics for Markov decision processes with infinite state spaces," in *UAI*, 2005.

[38] J. Song, Y. Gao, H. Wang, and B. An, "Measuring the distance between finite Markov decision processes," in *AAMAS*, 2016.

[39] F. Fernández and M. Veloso, "Probabilistic policy reuse in a reinforcement learning agent," in *AAMAS*, 2006.

[40] A. Lazaric, M. Restelli, and A. Bonarini, "Transfer of samples in batch reinforcement learning," in *ICML*, 2008.

[41] J. Sinapov, S. Narvekar, M. Leonetti, and P. Stone, "Learning inter-task transferability in the absence of target task samples," in *AAMAS*, 2015.

[42] H. Wang, S. Dong, and L. Shao, "Measuring structural similarities in finite MDPs," in *IJCAI*, 2019.

[43] G. Jeh and J. Widom, "SimRank: A measure of structural-context similarity," in *SIGKDD*, 2002.

[44] 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 *ICLR*, 2020.

[45] A. B. Dieng, Y. Kim, A. M. Rush, and D. M. Blei, "Avoiding Latent Variable Collapse With Generative Skip Models," in *AISTATS*, 2019.

[46] S. Sharma, A. K. Jha, P. S. Hegde, and B. Ravindran, "Learning to multi-task by active sampling," in *ICLR*, 2018.

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

[48] R. Wang, J. Lehman, A. Rawal, J. Zhi, Y. Li, J. Clune, and K. O. Stanley, "Enhanced POET: Open-Ended Reinforcement Learning through Unbounded Invention of Learning Challenges and their Solutions," in *ICML*, 2020.

[49] M. P. Deisenroth, G. Neumann, and J. Peter, "A Survey on Policy Search for Robotics," *Found. Trends Robot.*, vol. 2, no. 1-2, pp. 1–142, 2013.

[50] S. Thrun and J. O'Sullivan, "Discovering Structure in Multiple Learning Tasks : The TC Algorithm," in *ICML*, 1996.

[51] J. L. Carroll and K. Seppi, "Task similarity measures for transfer in reinforcement learning task libraries," in *IJCNN*, 2005.

[52] M. M. H. Mahmud, M. Hawasly, B. Rosman, and S. Ramamoorthy, "Clustering Markov Decision Processes For Continual Transfer," 2013. [Online]. Available: http://arxiv.org/abs/1311.3959

[53] K. Lee, Y. Seo, S. Lee, H. Lee, and J. Shin, "Context-aware Dynamics Model for Generalization in Model-Based Reinforcement Learning," 2020. [Online]. Available: http://arxiv.org/abs/2005.06800

[54] J. Yang, B. Petersen, H. Zha, and D. Faissol, "Single Episode Policy Transfer in Reinforcement Learning," in *ICLR*, 2020.

[55] A. Zhang, S. Sodhani, K. Khetarpal, and J. Pineau, "Multi-Task Reinforcement Learning as a Hidden-Parameter Block MDP," 2020. [Online]. Available: http://arxiv.org/abs/2007.07206

[56] A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul, H. Van Hasselt, and D. Silver, "Successor features for transfer in reinforcement learning," in *NeurIPS*, 2017.

[57] S. Hansen, W. Dabney, A. Barreto, T. Van de Wiele, D. Warde-Farley, and V. Mnih, "Fast Task Inference with Variational Intrinsic Successor Features," in *ICLR*, 2019.

[58] M. Riemer, M. Liu, and G. Tesauro, "Learning abstract options," in *NeurIPS*, 2018.

[59] C. D. Eramo, D. Tateo, A. Bonarini, M. Restelli, P. Milano, and J. Peters, "Sharing Knowledge in Multi-Task Deep Reinforcement Learning," in *ICLR*, 2020.

[60] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IROS*, 2012.

[61] T. Bräm, G. Brunner, O. Richter, and R. Wattenhofer, "Attentive Multi-task Deep Reinforcement Learning," in *ECML PKDD*, 2019.

[62] M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. Van Hasselt, "Multi-Task Deep Reinforcement Learning with PopArt," in *AAAI*, 2019.

[63] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, "Gradient Surgery for Multi-Task Learning," 2020. [Online]. Available: http://arxiv.org/abs/2001.06782

[64] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. J. Hausknecht, and M. Bowling, "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents," *Journal of Artificial Intelligence Research*, vol. 61, pp. 523–562, 2018.

[65] P. S. Castro, S. Moitra, C. Gelada, S. Kumar, and M. G. Bellemare, "Dopamine: A Research Framework for Deep Reinforcement Learning," 2018. [Online]. Available: http://arxiv.org/abs/1812.06110

[66] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *EMMNLP*, 2014.

Appendix A

Appendix Curriculum Learning

In addition to the details provided here, the implementation of all experiments is available online, but will not be linked here for now due to the anonymity requirements of a submission.

A.1 Experiment Details

A.1.1 Grid World Curriculum Experiment

In our first experiment, we use a 2D-grid-world with an edge-length in $\{1, 2, ..., 13\}$. The agent always starts in the top-left corner and the goal position is in the bottom-right corner. The observation of the agent consist of its current position, the possible actions are the cardinal directions. A reward of $r = 10$ is given upon reaching the target position, otherwise $r = 0$. The decay factor $\gamma = 0.9$ is used.

To train the agents we use tabular Q-learning with ϵ-greedy exploration. We begin with $\epsilon_0 = 0.2$ and decay this value with $\epsilon_t = \epsilon_0^{\gamma_\epsilon^t}$, with $\gamma_\epsilon = 1 - 1 \times 10^{-5}$. Here t corresponds to the number of time-steps in a single task and is therefore reset whenever a new task in the curriculum is started. This is necessary to allow exploration in the new tasks.

Embedding Details

As encoder for the Variational Autoencoder (VAE) we use a structure based on the structure used by Zintgraf et al. [44]. The encoder receives state s, action a, reward r and next state s'. The states s, s' are input into a fully-connected layer with 32 ReLU units, the actions a and reward r are input into two separate fully-connected layer with 16 ReLU units. The outputs from these layers are then concatenated and used as input for a fully-connected layer with 64 units, followed by a gated recurrent unit (GRU) [66] layer with 64 units. All transitions (s, a, r, s') from a trajectory are used as input to the encoder, only the final output of the GRU is used as output. It is then input into two fully connected

layers with $n_{\mathcal{N}}$ units corresponding to the dimensionality of the embedding. The outputs are used to parameterize the mean and diagonal covariance matrix of the multivariate normal distribution in the VAE. A sample m is then drawn from this distribution and used to represent the current task in the decoder. As the decoders are meant to represent the reward and transition function, we use two separate decoders and all transitions as input. Therefore, the reward decoder receives (s,a,s',m) as input and should output r. Similarly, the transition decoder receives (s,a,r,m) as input and should output s'. Both have two fully connected layers with 64 and 32 tanh units respectively, followed by a linear output layer with dimension corresponding to the state shape or a single scalar for the reward. We then average the mean-squared errors per trajectory, to avoid focusing on long trajectories only, and sum up the losses.

$$\begin{aligned}\mathcal{L}_{\texttt{total}} &= \beta D_{\mathbf{KL}}(q(z|x)||p(z)) + \mathcal{L}_r + \mathcal{L}_t \\ &= \beta D_{\mathbf{KL}} + \frac{1}{N_T}\sum_{T=1}^{T=N_T}\Big(\frac{1}{L}\sum_{t=1}^{t=L}||\tilde{s}'_t - s'_t||_2 + \frac{1}{L}\sum_{t=1}^{t=L}||\tilde{r}_t - r_t||_2^2\Big)\end{aligned} \tag{A.1}$$

with N_T being the number of trajectories, L being length of each episode (might vary) and $\tilde{s}'$ and $\tilde{r}$ being the outputs of the decoders.

We use the squared euclidean norm $||\cdot||_2^2$ which formally corresponds to assuming a symmetric normal distributions over transitions and rewards. As we use the re-parameterization [22] trick to represent the normal distribution, we can propagate the gradient of this loss to the encoders, and train the whole model by gradient descent. All hyper-parameters are shown in Table A.1.

Table A.1: Hyperparamters for Curriculum Grid-World Experiments.

Hyperparameter	Value
learning-rate α	0.5
random action prob. ϵ	0.1
eps-decay γ_ϵ	$1 - 1 \times 10^{-5}$
reward decay γ	0.9
VAE training epochs	20
VAE regularization β	10^3
VAE learning rate	1×10^{-4}
VAE batch size	100 trajectories
Trajectory subsampling	–
VAE Training steps per task	1.5×10^4

A.1.2 BipedalWalker

Most details about the tasks can be found in Section B.1.3, as the tasks are almost the same. However, in the track and field task set we also use a sparse version of the hurdles task, in which a reward $r = 10$ is only given when a hurdle is crossed, and otherwise only the torque regularization reward is used. We terminate episodes after 2000 steps or when the agent reaches the target or falls over.

We train the VAE with the hyperparameters shown in Table A.2. As the trajectories here are a lot longer than in the grid-world. Here we use the sampling-based curriculum shown in Algorithm 2. All policies are trained using Twin Delayed Deep Deterministic Policy Gradient (TD3). We did not perform a hyper-parameter search in this task, as we assume that the parameters should affect all approaches similarly.

Table A.2: Hyperparamters for Curriculum BipedalWalker Experiments.

Hyperparameter	Value
batch-size	1000
update-rate	5
policy-update-frequency	3
network size	$(400, 300)$
exploration noise σ	0.1
exploration noise clipping	$[-0.5, 0.5]$
target policy smoothing noise σ	0.2
buffer-size	5×10^6
decay γ	0.99
VAE training epochs	200
VAE regularization β	10^3
VAE learning rate	1×10^{-4}
VAE batch size	100 trajectories
Trajectory subsampling	50
VAE Training steps per task	1.5×10^5

A.2 Failed Attempts in the BipedalWalker Taks

In the first task-set for the BipedalWalker, in which we vary leg-length and obstacle spacing, we found curricula to be unnecessary, as this task can be learned directly quite quickly, and therefore did not continue working on these tasks. In the second set of tasks, inspired by track and field events, we found an oracle curriculum to be helpful. However, both of our two proposed approaches,

the greedy and sampling-based curricula, did not manage to identify a similarly good curriculum. They therefore performed similar to a random curriculum. To avoid this, we attempted several changes to our curriculum approach. Ultimately none of them proved successful. We therefore briefly list them here and give some insight into why they were attempted and why they failed or were not pursued further.

The main problem in the BipedalWalker experiments, as stated above, is that embeddings generated by trajectories of a random policy do not capture the structure of the underlying tasks. Therefore any curriculum generated from these embeddings is similar to a random curriculum. An easy way to avoid this issue is to learn an embedding with previously trained policies. While we show that doing so results in a meaningful embedding, as shown in Figure 3.10, we do not pursue this further as it contradicts our motivation of quickly learning a single policy.

Sometimes a policy trained with randomly sampled tasks converges sufficiently to generate a meaningful embedding. We then have to decide which tasks to sample. Simply basing it on distance to the target task does not work, as it will result in mainly training directly on the target task. If training on the target task directly is feasible, this will work, but in this case using a curriculum is ill-advised from the beginning. In most settings, such as in ours, we therefore want to train on related but easier tasks. We therefore add a learnability function $l : \mathcal{T} \to \mathbb{R}^+$. We first tried using the variance of the return over different trials of the same task, as was successful in the grid-world experiments. However, this was not successful, as the variance of the reward is similar on all tasks, and estimating it reliably requires a large number of roll-outs, which contradicts the objective of quickly learning a good performance on the target task. We therefore, motivated by Zhang et al. [33], attempted to use epistemic uncertainty about the value as a proxy for learnability. We therefore trained 3 Q-functions separately from the one used to train our agent and calculate the epistemic uncertainty as $l_Q = \frac{1}{l} \sum_{t=1}^{t=l} \frac{1}{3} \sum_{k=1}^{k=3} (Q_k(s_t, a_t) - \bar{Q}(s_t, a_t))^2)$. While this reduces the number of required roll-outs, it has the different issue of value disagreement being high on each task, that the agent has not trained on.

We also tried variants on the VAE, such as not using the reward in the input or output of the VAE, keeping a persistent VAE model and just fine-tuning it every N episodes, using the latent state of the GRUs instead of their output, and many other structural changes.

Appendix B

Appendix Task Clustering

B.1 Experiment Details

B.1.1 Discrete Multi-Task Experiments

In the first discrete task set we use a one-dimensional state-chain with 51 states, in which the agent starts in the middle and receives a reward for moving toward either the left or right end. As a reward we use $r = \frac{1}{|x_{\text{ag}} - x_{\text{goal}}|}$ where x_{ag} is the position of the agent and x_{goal} is the goal position (either the left or right end of the chain). We give a reward of $r = 20$ if the goal position is reached. Depending on the task, the reward is given every 2, 4, 8 or 16 steps, or only at the goal position, and otherwise replaced by $r = 0$.

For our corner grid-world task set we use a 2D-grid-world with edge length 7 and three goal positions per corner (as depicted in Figure 4.1). The agent always starts in the center and receives a reward based on the distance to the target $r = \frac{1}{||x_{\text{ag}} - x_{\text{goal}}||_2}$, with $|| \cdot ||_2$ being the Euclidean norm. A reward of $r = 10$ is given when the agent reaches the goal position.

In both tasks we use tabular Q-Learning with ϵ-greedy exploration. We start with $\epsilon_0 = 0.2$ and decay the value as $\epsilon_t = \epsilon_0^{\gamma_\epsilon^t}$ with $\gamma_\epsilon = 1 - 1 \times 10^{-6}$. We use a learning rate of $\alpha = 0.2$ to update the value estimates, as from the perspective of a single agent the environment can be regarded as stochastic. Further, we use a discount factor of $\gamma = 0.9$ and $T_M = 500$ training steps per policy in each M-step and evaluate each policy on each task for three episodes during the E-step, using the greedy policy without exploration.

B.1.2 Pendulum

In our pendulum tasks we use a modified version of the *Pendulum* environment provided in OpenAI gym [47]. This environment consists of a single pendulum and the goal is to balance it in an upright position. The observation consists of the current angle θ, measured from the upright position, and current angular

velocity represented as $(\sin\theta, \cos\theta, \dot{\theta})$. The reward for each time step is $r_t = -(\theta^2 + 0.1\dot{\theta}^2 + 0.001a^2)$, with a being the torque used as action. Every episode starts with a random position and velocity. To provide a set of tasks we vary the length of the pendulum in $\{0.7, 0.8, ..., 1.3\}$.

Network Structures

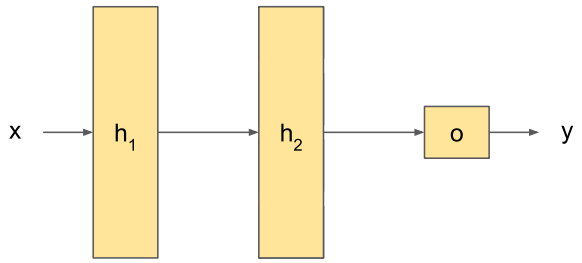
As we have three different network structures, we provide illustrations of each in Figure B.1. We use completely separate networks for the actor and two critic functions, however, they all share the same network structure, except for the differing input and output values. We therefore use x and y to represent inputs and outputs respectively in the diagrams. In the case of the actor, $x := s, y := a$, while in the case of the critic $x := (s, a), y := Q(s, a)$.

Hyperparameters

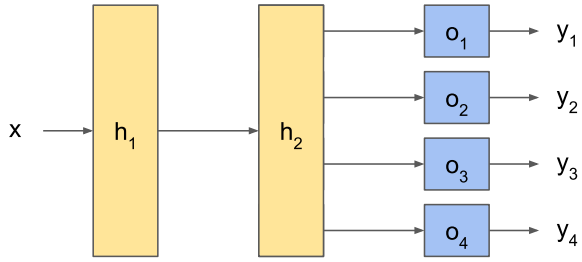
Hyperparameters for our EM-TD3 and multi-head TD3 were tuned on the pendulum task set by grid search over learning rate $\alpha = \{1\times10^{-2}, 3\times10^{-3}, 1\times10^{-3}\}$, batch-size $b = \{64, 128\}$ and update-rate $u = \{1, 3, 5\}$, specifying the number of collected time-steps after which the value-function is updated. We increased the network size for multi-head TD3, so that it overall had more parameters than EM-TD3. This is done to eliminate a potential advantage of our approach stemming from a higher representational capacity. The tuned hyperparameters are given in Table B.1. To represent the value functions and policies we use fully connected multi-layer perceptrons (MLPs) with two hidden layers with 64 units each. As activations we use ReLU on all intermediate layers, and tanh activations on the output. The values are then scaled to the torque limits per dimension. In EM, SP and PPT we use a separate network for each policy. For our multi-head baseline we share the hidden layers between tasks, but use separate input and output layers per task. Additionally, we increase the size of the first hidden layer to 96 in the multi-head approach, such that it has a similar total number of parameters as our EM approach. For the Multi-Out baseline we use the same hyper-parameters as in the Multi-Head baseline, but increase the network size to $(800, 600, 9 \cdot 1)$. For SP and PPT we reuse the hyper-parameters from our EM approach. The different network structure of each approach are shown in Figure B.1. During the M-step, we train the agent for 5×10^4 steps per policy and during the E-step we evaluate each agent on each task by running 20 episodes without added exploration noise.

B.1.3 BipedalWalker

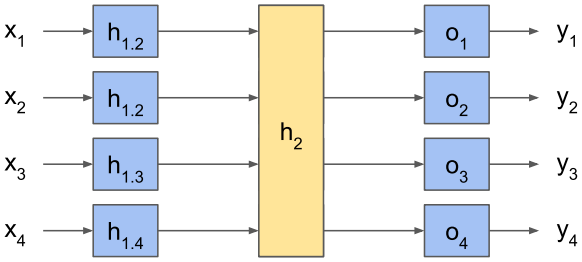
For the BipedalWalker tasks we look at two different sets of tasks. The first set of tasks consists of different reward functions with mostly similar environments,



(a) The normal network structure, used in EM-TD3, PPT and SP.



(b) The Multi-Out structure. Here only the output layers are task-specific.



(c) The Multi-Head structure, identical to Eramo et al. [59]. Here both output and input layers are task-specific, but a shared layer is used between them.

Figure B.1: Shown are the three different network structure used in our Pendulum and Bipedal Walker experiments. They are all shown for $k = 4$ tasks, while in practice the number of tasks varies by settings. Yellow layers are shared among tasks while blue layers are task-specific. In the case of the actor, $x := s, y := a$, while in the case of the critic $x := (s, a), y := Q(s, a)$.

Table B.1: Hyperparamters for pendulum experiments.

Hyperparameter	EM-TD3	Multi-head TD3
learning-rate α	3×10^{-3}	3×10^{-3}
batch-size b	128	128
update-rate u	1	1
policy-update-frequency	3	3
n - EM	4	-
network size	$4 \cdot (64, 64, 1)$	$(9 \cdot 96, 64, 9 \cdot 1)$
exploration noise σ	0.05	0.05
exploration noise clipping	$[-0.5, 0.5]$	$[-0.5, 0.5]$
target policy smoothing noise σ	0.1	0.1
buffer-size	2×10^6 per policy	2×10^6 per task
decay γ	0.99	0.99
T_M	5×10^4	-

inspired by track and field events. The tasks are jumping up, jumping a long distance, runs for different distances and a run with obstacles. In all tasks a reward of $-\epsilon||a||_1$ is given to minimize the used energy. The position of the hull of the bipedal walker is denoted as (x, y). In the jump up task a reward of $y - |x|$ is given upon landing, and $\epsilon = 3.5 \times 10^{-4}$. For the long jump task a reward of $x - x_0$ is given upon landing, with x_0 being the hull position during the last ground contact, $\epsilon = 3.5 \times 10^{-4}$. The three runs consist of a sprint over a length of 67 units, with $\epsilon = 3.5 \times 10^{-4}$, a run over 100 units, with $\epsilon = 3.5 \times 10^{-4}$, and a long run over 200 units with $\epsilon = 6.5 \times 10^{-4}$. The hurdles task is identical to the long run, but every 4 units there is an obstacle with a height of 1. Additionally, a reward of $0.1\dot{x}$ — a reward proportional to the velocity of the agent in the x-direction — is given during the run and hurdle tasks, to reward movement to the right.

The second set of tasks consists of varying obstacles and robot parameters. We vary the length of the legs in $\{25, 35, 45\}$ and either use no obstacles, or obstacles with a spacing of 2 or 4 units apart and height of 1. This results in a total of 9 tasks. Here we use the standard reward for the BipedalWalker task $r = 4.3\dot{x} - 5|\theta| - ||a||_1$ with θ being the angle of the walker head. Additionally, in all experiments $r = -100$ is given if the robot falls over or moves to far to the left.

Hyperparameters

Hyperparameters for our EM-TD3 and multi-head TD3 approaches were tuned on the track and field task set by grid search over $\alpha = \{1 \times 10^{-3}, 3 \times 10^{-4}, 1 \times 10^{-4}\}$,

batch-size $b = \{100, 1000\}$ and update-rate $u = \{1, 3, 5\}$, u specifying the number of collected time-steps after which the value-function is updated. We reuse the optimal parameters found here on the task set with varying leg lengths and obstacles. For the SP and PPT baselines we reused the parameters from EM-TD3. We increased the network size for multi-head TD3, so that it overall had more parameters than EM-TD3. All hyperparameters are given in Table B.2. For the Multi-Out baseline we use the same hyper-parameters as in the Multi-Head baseline, but increase the network size to $(800, 600, 9 \cdot 1)$. During the M-step, we train the EM agent with 2×10^5 steps per policy and during the E-step we evaluate each agent on each task by running 20 episodes without added exploration noise.

Table B.2: Hyperparameters for BipedalWalker experiments.

Hyperparameter	EM-TD3	Multi-head TD3
learning-rate	1×10^{-3}	1×10^{-3}
batch-size	1000	1000
update-rate	3	5
policy-update-frequency	3	3
n - EM	4	-
network size	$4 \cdot (400, 300, 1)$	$(6 \cdot 400, 400, 6 \cdot 1)$
exploration noise σ	0.1	0.1
exploration noise clipping	$[-0.5, 0.5]$	$[-0.5, 0.5]$
target policy smoothing noise σ	0.2	0.2
buffer-size	5×10^6 per policy	5×10^6 per task
decay γ	0.99	0.99
T_M	2×10^5	-

B.1.4 Atari

To test our approach on a more complex task, we evaluate it on a subset of the Atari games. The set of chosen games consists of Alien, Assault, BankHeist, ChopperCommand, DemonAttack, JamesBond, MsPacman, Phoenix, RiverRaid, SpaceInvaders, WizardOfWor and Zaxxon. As stated above, this task set is similar to the set of games used in [58], but without tasks requiring a large amount of exploration to save computation time.

Our implementation is based on the Implicit Quantile Network (IQN) implementation in the Dopamine framework [16, 65]. As hyperparameters we use the default values recommended by Dopamine for Atari games, except the changes listed below: Due to the different action spaces, we use a separate replay buffer for each game, as well as a separate output, both for our EM, multi-head and PPT approaches. Due to the nature of the reparameterization layer in IQN, dif-

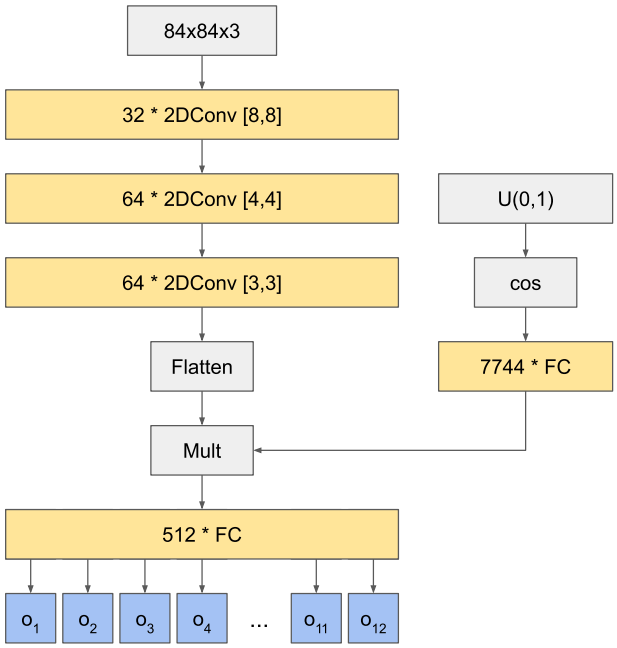


Figure B.2: Shown is the structure of our atari network, used in our EM approach and multi-head baseline. $U(0,1)$ here denotes a uniform distribution between 0 and 1, which is used in the reparameterization. All layers are shared between tasks except the output layer, shown in blue. Shown are the default parameter counts, in practice we scale the layers to have the same number of total parameters per approach.

ferent multi-head structures could be considered. We show our chosen network structure in Figure B.2. We reduce the size of the replay buffer to 3×10^5 compared to 1×10^6 in the original paper, to reduce the memory demand. We use the normal NatureDQN network, but scale the size of the layers to ensure that each approach has a similar number of parameters. For our EM approach, we use $T_M = 2.5 \times 10^5$ trainings steps per M-step, and evaluate all policies on all tasks for 27000 steps in the E-step, using the greedy policy without random exploration. In both EM and the multi-head approach, we record how many transitions were performed in each M-Step and sample the task with the least transitions as next training task. This is done to ensure a similar amount of transitions and training steps per game, as episode lengths vary. This approach was proposed in [58].

B.2 Additional Results

B.2.1 Bipedal Walker

In Figure B.3 the assignments for 4 randomly chosen trials on the track and field task set are shown. We can see that in all trials the runs over different distances are grouped together with the long jump task. This is likely due to these tasks aligning well, as they both favor movements to the right. It is possible to learn the hurdles task with the same policy as the runs, due to the available LIDAR inputs. The hurdle task therefore sometimes switches between policies, but usually is learned by a separate policy. The jump up task is very different from the other tasks, as it is the only one not to involve movement to the right, and is therefore assigned to a separate policy.

In Figure B.4 the assignments for 4 randomly chosen trials on the leg-length and obstacle task set are shown. As illustrated by the good performance of the SP approach shown in Figure 4.5, it is possible to learn a nearly optimal behavior with a single policy here. This makes learning a meaningful clustering significantly harder and sometimes leads to a single policy becoming close to optimal on all tasks, as in Trial 2. In most other trials the task set is separated into two or three different clusters based on the different leg lengths.

B.2.2 Atari

In Figure B.6 the assignments of all three trials of our approach on the Atari task set are shown. While we see a consistency in assignments, we cannot identify a clearly repeated clustering across trial. We assume this is due to the high diversity of tasks preventing the identification of clearly distinguishable clusters. This lack of clearly distinguishable clusters might also be the reason for failing to reach the performance of PPT. Yet, the specialization of policies in our approach helps to avoid negative transfer as seen in Figure 4.6.

In Section 4.3.3 we reported the median reward across games, as is common for Atari games. This is done to avoid a large difference in a single game skewing the results significantly. However, because we stated our goal as achieving the highest mean reward in the beginning of our thesis, we show the mean reward in the top of Figure B.5 and below it we again show the rewards per task . We notice that in this metric PPT outperforms our proposed approach, both with $n = 4$ and $n = 8$. When we look at the rewards per game, we can see that main reason for that is PPT outperforming our approaches by 10x in *JamesBond* while the reward is similar in the other games.

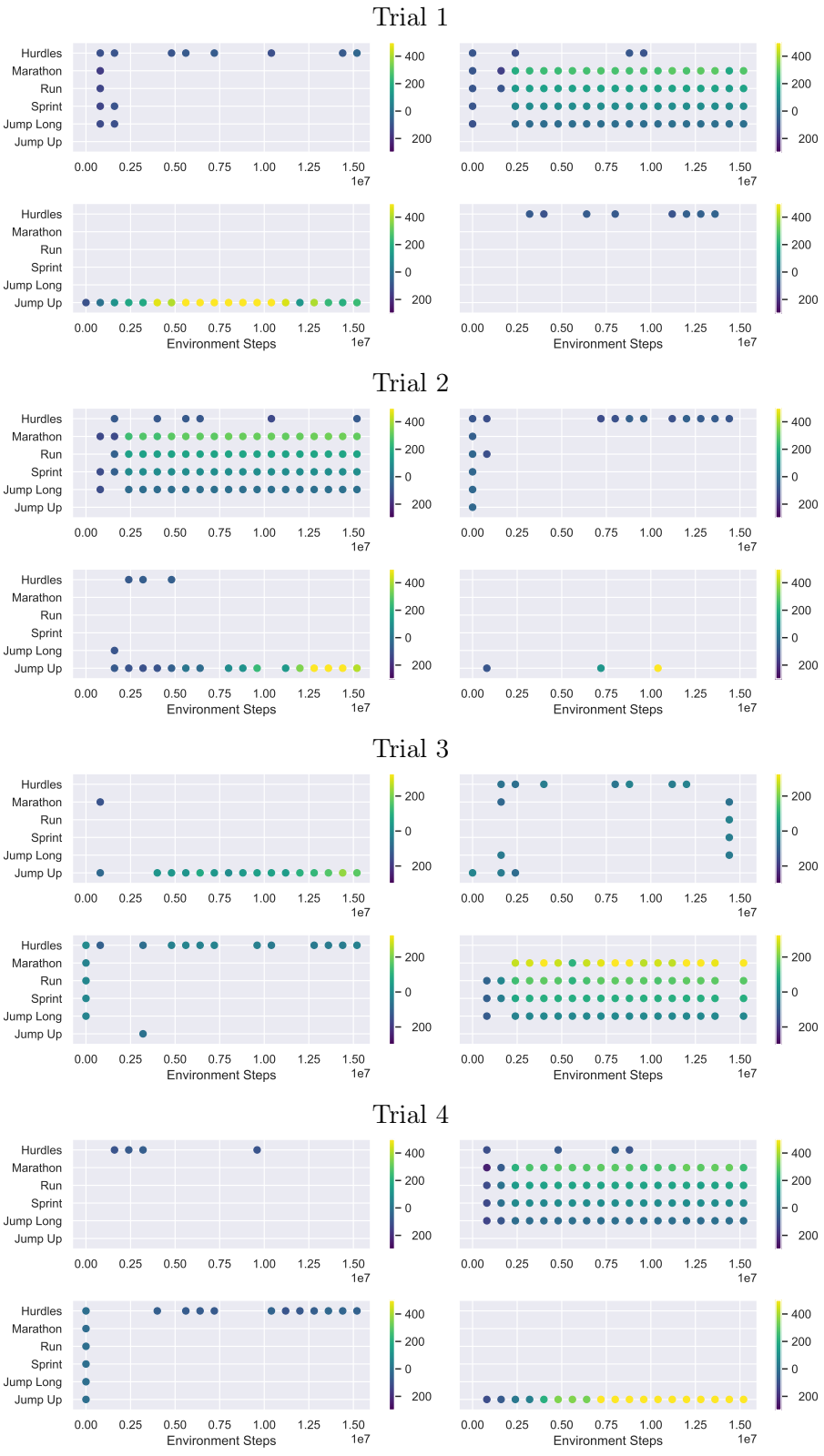


Figure B.3: Shown are the assignments from 4 randomly picked trials on the track and field BipedalWalker task set.

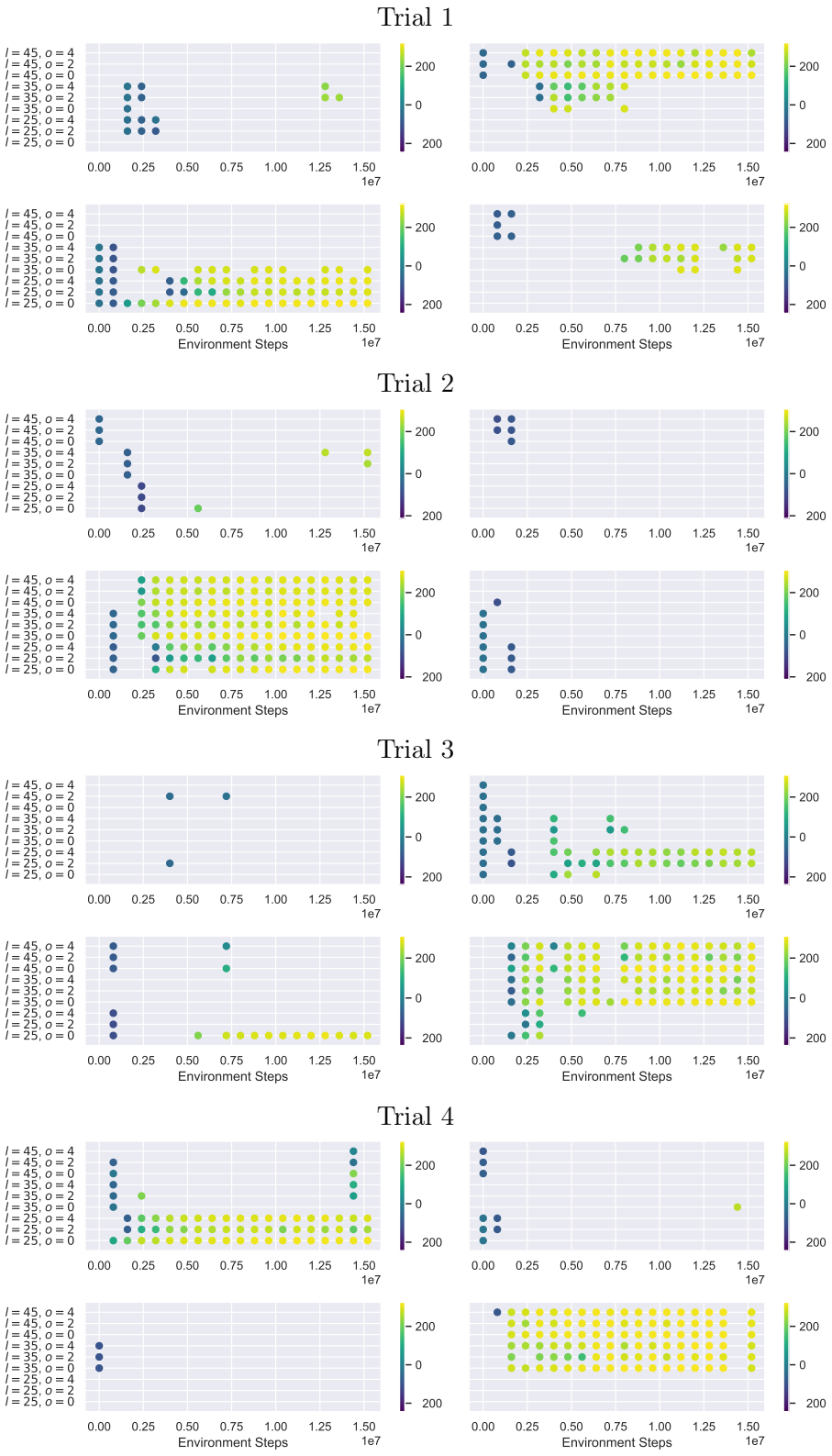


Figure B.4: Shown are the assignments from 4 randomly picked trials on the first BipedalWalker task set. l refers to the lenghts of the legs, o refers to the frequency of obstacles.

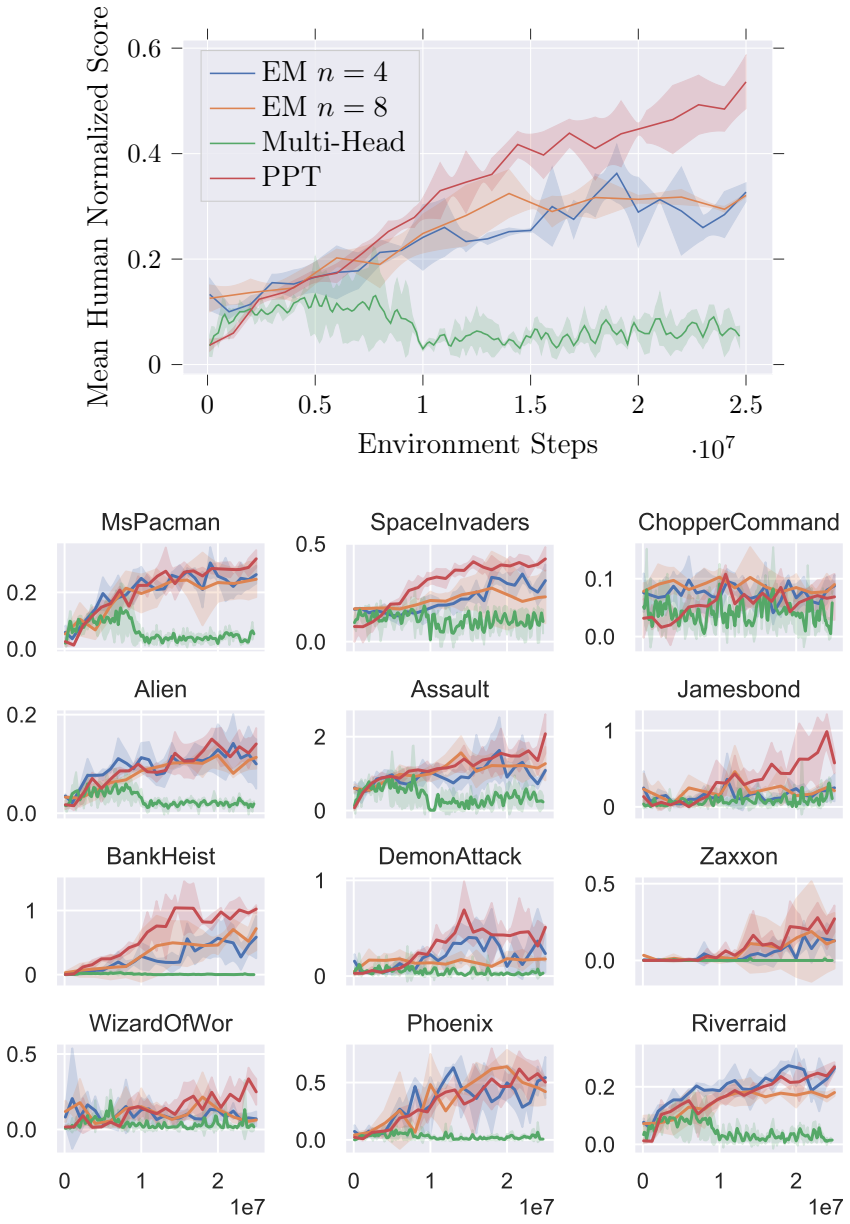


Figure B.5: Atari mean human-normalized reward over games per trial, averaged across three trials per approach (top). Atari mean human-normalized reward per game (bottom). We can see that, unlike the median, the mean is dominated by PPT outperforming the EM approaches in JamesBond.

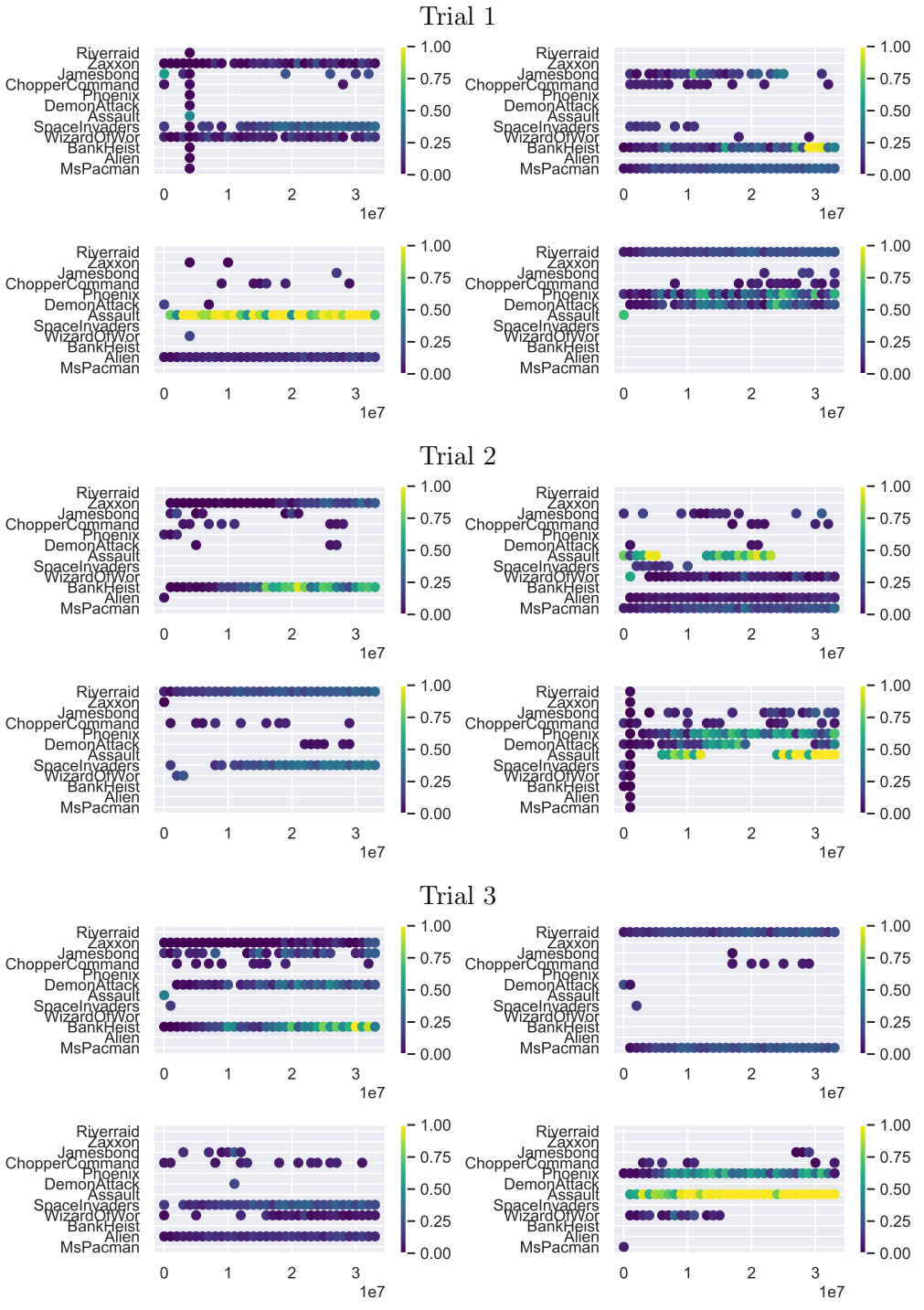


Figure B.6: Shown are the assignments of all three trial that were run on the set of Atari games. The color represents the human-normalized score per game.

List of Figures