DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Hierarchical Skill Discovery via Reinforcement Learning

Max Fest

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Hierarchical Skill Discovery via Reinforcement Learning

Hierarchische Entdeckung von Fähigkeiten via Reinforcement Learning

Author:	Max Fest
Supervisor:	Prof. Dr.-Ing. habil. Alois Knoll
Advisor:	Dr. rer. nat. Zhenshan Bing
Submission Date:	15.11.2021

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.11.2021 Max Fest

Abstract

Reinforcement learning (RL) has traditionally focussed on agents that achieve mastery in some narrow task, generalising poorly even against minor variations. Countering this, recent *skill discovery* methods have sought to learn behaviours in a completely task-agnostic manner, exploring what is possible in an environment and how to achieve it. We want to take advantage of the strong parallel between skills and options in HRL, and provide a framework for learning a hierarchy of skills in an unsupervised, bottom-up manner. We adjust a prior work and propose cDIAYN, a skill discovery method that learns skills suitable for hierarchical composition using a continuous skill space. We show that we can recursively apply cDIAYN to the same reward-free environment to learn a multi-layer skill hierarchy.

Contents

1. Introduction

Since its foundation, the goal of artificial intelligence (AI) has been to allow computers to perform more complex tasks with less human supervision. We want to develop problem-solvers that are immediately applicable to broad classes of problems, perhaps the broadest of which is studied in Reinforcement Learning (RL). RL concerns itself with problems that can be modelled as an agent making decisions in an environment over time (figure 1.1).

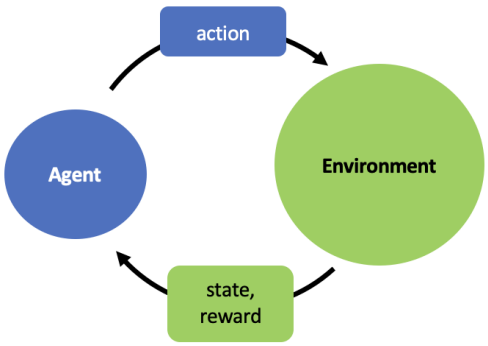


Figure 1.1.: The RL loop.

At every time-step, the agent selects an action which modifies the environment state and emits a corresponding reward signal. The basic idea here is rooted in behaviourism: the agent should learn to seek out rewarding states and to avoid states that are unrewarding or punishing [1]. The agent starts out as a blank slate, and must learn by trial-and-error interaction with the environment how to behave in order to maximise total reward. This formulation implies a few fundamental challenges:

1. **The curse of dimensionality**: For higher-dimensional input spaces, we need exponentially more points to cover it. For example, if we are asked to choose an action M times, with N possible actions, we are searching a space of N^M possible action-sequences. If we wanted to find the best one by brute-force and without making any further assumptions, this is how many we would have to evaluate.

2. **The exploration-exploitation dilemma**: Assuming that we can't cover the space of possible behaviours exhaustively, there is a fundamental tradeoff between exploitation and exploration, i.e. between choosing the *best* action according to our current state of

knowledge, and exploring other actions that could be better.

3. **The credit assignment problem**: When we reach a rewarding state, this was likely preceded by many actions. Which of these were most important in reaching this final state? At which could we have acted better?

4. **The reward bottleneck**: In the basic RL formulation, the reward function is the only channel through which we can communicate to the agent which behaviours we want it to learn. Engineering these to elicit complex behaviours very quickly becomes difficult, but less informative reward functions make the problem more difficult to solve for the agent.

Temporal abstraction has long been hypothesised to be a key ingredient for better managing all of the above [2]. This can be intuited from a simple cooking analogy: a recipe generally doesn't describe the individual joint movements required to correctly manipulate the ingredients, but provides a short list of sub-goals that the cook should know how to accomplish, like *dice the onions* or *cook the noodles*. Each of these sub-goals can be decomposed into further sub-goals, and so on until sub-goals really do reflect joint movements. In this way, we can decompose a single RL problem into many smaller problems, each of which can ideally be evaluated in isolation of the others.

However, these approaches generally still rely on an external reward signal, limiting their potential to learning the behaviours we can effectively encode. These are usually also task-specific, making them difficult to generalise, and meaning that an RL agent starts from a blank slate (i.e. from individual joint movements) when faced with a new task (or recipe).

Of course, there should be much to gain from interaction with the environment even in the absence of such a reward signal. Like a baby builds a model of the world, we would like agents to be capable of exploring the realm of possibilities and then using this experience to inform future actions (for example when maximise an external reward). In other words, we want to design reward functions that are task agnostic, usually termed *intrinsic rewards* in RL. Where external rewards characterise a top-down approach to learning, intrinsic rewards may be viewed as their bottom-up counterpart, in that we are attempting to model the inductive biases that enable learning in a more general sense.

One way to benefit from unsupervised interaction with the environment, is to discover *skills*. A skill can be thought of as the solution to a task, but in skill discovery the agent learns to propose tasks to itself. If a human were put on a patch of grass with a ball, they would eventually discover that they can kick it, or pick it up and bounce it. Given enough time and nothing else to do, they can find out what else is possible with a ball, and how to do it efficiently and consistently.

Many of the goals we set an agent presuppose certain instrumental knowledge. For

example, if we want a simulated humanoid agent to move to some goal-location, it must first learn to move at all. The larger the dimension of state- and action-spaces, the more ill-defined the initial navigation goal therefore is. We would like to learn different gaits and movement patterns prior to optimising goals that build on these skills.

Such unsupervised skill discovery has been explored in a number of prior works [3, 4, 5], described in this thesis as Variational Option Discovery (VOD) [6] algorithms. These are built around the *diversity assumption*, by which we can learn many skills simply by declaring that they be different. VOD has also formed the basis for hierarchical approaches [7, 8], however skill discovery is generally only performed in a single layer. In this thesis, we aim to fuse VOD and HRL, proposing a mechanism for learning *hierarchies of skills* in a completely unsupervised manner. In this, we adopt a more flexible definition of skills in which they are composed of other skills, such that the diversity assumption is fulfilled at different scales of temporal abstraction.

2. Preliminaries

2.1. Reinforcement Learning

Reinforcement Learning aims to solve complex sequential decision-making problems, wherein an agent interacts with an environment to optimise some kind of reward. This problem class is usually modelled as a *Markov Decision Process* (MDP), characterised in its simplest form by the tuple $\langle \mathcal{S}, \mathcal{A}, P, r \rangle$, where:

- $\mathcal{S}$ is the set of states. A state includes any information about the environment and the agents place in the environment that the agent has access to.
- $\mathcal{A}$ is the set of actions the agent can take. It defines the parts of the MDP that the agent has direct control over.
- $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the environment transition function, i.e. if action $a \in A$ is taken in state $s \in S$, you end up in state $s' \in S$ with probability $P(s'|s, a)$. In contrast with dynamic programming, we do not assume knowledge of transition dynamics prior to learning, only that we can sample from them via environment interaction.
- $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. It encapsulates how 'good' an action that you take from a specific state is. We abbreviate the reward achieved on timestep t with r_t.

The behaviour of the agent is summarised by its *policy* $\pi : \mathcal{S} \to \mathcal{A}$, the function that defines what action the agent takes in each possible state. From this we derive the RL problem: how do we learn an optimal policy π^*? More broadly (since optimality is defined in terms of reward), how do we map an arbitrary reward function r to a policy π^*?

Typically we aim to maximise the expected cumulative reward, i.e. we solve:

$$\pi^* = arg \max_{\pi} \mathbb{E}_{\tau \sim \pi(s_0)} \sum_{t=0}^{T} r_t \tag{2.1}$$

In this, $\tau = (s_0, a_0, s_1, a_1, ..., a_{T-1}, s_T)$ is a *trajectory* produced by rolling out the policy in the environment for T time-steps.

However, this is just the most concise formulation. For example, this problem is usually extended into the infinite horizon setting ($T = \infty$), in which case (2.1) is made solvable by multiplying r_t with a discount factor γ^t, $\gamma \in [0, 1)$. It also encodes the notion that we care more about immediate than distant rewards. In practice, both the optimisation problem itself and how we solve it are adjusted to better fit different learning mechanisms we expect the agent to display.

To optimise (2.1), RL algorithms estimate the expected reward achievable from a state under the current policy, as given by the *value function* V^π:

$$\begin{aligned} V^\pi(s) =& \mathbb{E}_\pi\left\{ \sum_{k=1}^{\infty} \gamma^k r_{t+k} | s_t = s \right\} \\ =& \mathbb{E}_\pi\left\{ r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s \right\} \end{aligned} \tag{2.2}$$

From this, we can define the *action-value function* Q^π:

$$Q^\pi(s, a) = \mathbb{E}_\pi\{ r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s, a_t = a \} \tag{2.3}$$

Intuitively, if $Q^\pi(s, a) > V^\pi(s)$, then we could increase the expected reward of our policy by encouraging it to take the action a in state s. This idea is the basis of *Actor-Critic* methods [9]. We can see that the problem of finding an optimal policy is closely linked to that of finding an optimal value function, though the latter problem is slightly broader. In *policy iteration*, we use (2.2) to evaluate the current policy, and directly change the policy based on this evaluation, whereas in *value iteration* approaches we aim to find the optimal value function, from which we subsequently derive the optimal policy (at each state we take the action that maximises action-value).

2.1.1. Deep Reinforcement Learning

One important adjustment was in the use of neural networks to approximate the policy and/or value functions, giving birth to the field of deep reinforcement learning. The first deep RL (DRL) algorithm to show significant promise: 'Deep Q-Networks (DQN)' [10] learned to play Atari games from raw, unstructured pixel data to learn a parameterised approximation of equation 2.3 via stochastic gradient descent with the loss:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s',a') \sim U(\mathcal{D})}\left\{ \left(r + \gamma \max_{a'} Q_{\theta_i^-}(s', a') - Q_{\theta_i}(s, a) \right)^2 \right\} \tag{2.4}$$

Not only did DQN play Atari games at a level comparable to that of humans, but it did so without the need for handcrafted state representations, and was thus immediately applicable to a wide range of problems. The primary contribution was in the formulation of a few techniques that allow for more stable training of deep function approximators in the RL

setting. Most importantly, they used *Experience Replay*, which refers to the use of a *replay buffer* $\mathcal{D}$ to store experience (s, a, r, s') collected from the environment, from which we can then sample (to decorrelate training data) to train the neural network, as well as an outdated target q-function parameterised by θ_i^- (updated to equal θ_i every few iterations).

2.1.2. Multi-task and Goal-conditioned RL

In the standard MDP, we assume a single reward function r, corresponding to the optimisation of a single target. Multi-task RL generalises this by introducing the notion of tasks $\mathcal{T}_i = \langle \mathcal{S}_i, \mathcal{A}_i, p_i(s_0), p_i(s'|s,a), r_i(s,a) \rangle \in \mathcal{T}$. The goal is then to learn a policy that generalises well over all these tasks, finding and exploiting the commonality between them, and more quickly adapting to previously unseen tasks. This is a deliberately broad definition, but we may choose any subset of the above to vary between tasks. An overview of this variability is given in [11].

In Multi-task RL, each task defines its own MDP. Goal-conditioned RL can be seen as a special case of this, in that a family of reward functions $r_g : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is induced by a set of goals $g \in \mathcal{G}$ over a single MDP, i.e. $\mathcal{S}$, $\mathcal{A}$, $p(s'|s,a)$, and $p(s_0)$ generally remain constant between goals.

The goal-conditioned RL (GCRL) framework was introduced as Universal Value Function Approximation [12]. Intuitively, we determine the value $V_g(s)$ of a state $s \in \mathcal{S}$ within the context of a goal $g \in \mathcal{G}$ we are currently trying to reach, and thus derive a goal-conditioned policy $\pi(a|s,g)$. In many prior works, the goal space is simply a subset of the state space $\mathcal{G} \subset \mathcal{S}$, but others introduce representation learning to learn a more meaningful goal space [13, 14].

2.1.3. Intrinsic Motivation

Traditionally, an MDP supplies a reward function which the agent aims to optimise. However, designing such a reward function comes with many challenges. If the reward is sparse (i.e. most transitions (s, a, r, s') have $r = 0$), then these transitions offer the agent no guidance on how to adjust its behaviour. On the other hand, crafting a task-specific dense reward (*reward shaping*) usually requires a domain expert and is notoriously difficult to get perfectly right, potentially introducing unintended optima.

Work in Intrinsic Motivation (IM) seeks to develop task-agnostic reward functions, based on general principles like information gain [15], empowerment [16], or surprisal [17]. It is based on the idea that agents can learn from environment interaction even in the absence of

an external reward signal, for example to explore the environment more effectively. The agent can autonomously gather knowledge that later facilitates quicker adaptation to new tasks. In general, intrinsic motivation is integrated into the RL framework simply by replacing the usual reward function r_{ext} with a weighted sum of internal and external rewards [18]:

$$r(s,a) = r_{IM}(s,a) + \beta r_{ext}(s,a)$$

IM can therefore be used to turn a sparse external reward signal into a dense one, providing a mechanism for evaluating environment interaction that would otherwise have offered no information for the RL agent.

2.1.4. Model-based RL)

In Model-based RL (MBRL) we attempt to learn the environment transition dynamics $p(s'|s,a)$ in order to simulate environment interaction without actually having to perform it. The agent is imbued with a model of the world, which it may use to ask itself "what if?", to reason about trajectories it never actually experienced. It may choose actions not only according to its value estimate of the current state, but also of the possible future states, thus introducing a mechanism for planning into the future.

Model-based RL promises to improve sample efficiency by weakening the dependence on real environment interaction. With a perfect model $p(s'|s,a)$, we do not require environment interaction at all, although we still encounter a tradeoff between acting and planning. Knowing the dynamics does not imply value-function or policy given some reward function, but it gives us more flexibility in deciding how we want to traverse the state-space in order to maximise rewards.

As we might expect, acting on a flawed model of the world can cause an agent to learn a suboptimal policy, and model-based RL agents may attain lower asymptotic performance than their model-free counterparts [19]. One key problem in model-based RL (similar to that of off-policy RL) is allowing the agent to learn from experience generated from a flawed or outdated model.

2.1.5. Hierarchical Reinforcement Learning

At every time-step, an RL agent chooses an action to execute. The longer an agent interacts with the environment, the larger the space of actions the agent could have taken instead. In the absence of an effective exploration mechanism, flat RL algorithms generally have trouble with such long-horizon environments. Hierarchical RL (HRL) aims to decompose a long-horizon task into a hierarchy of subproblems, each of which is simpler to solve than the

full problem. More concretely, the action space of the agent is augmented or replaced with behaviours that are executed for more than one time-step.

This idea was pioneered in the Options framework [20]: an option ω is defined as a tuple $(I_\omega, \pi_\omega, \beta_\omega)$, where I_ω is the initiation set of the option, π_ω is the policy taken by the agent acting under ω, and $\beta_\omega : \mathcal{S} \to [0, 1]$ is the termination probability. In training the option policy π_ω we typically encounter an option-specific reward r_ω, different from the main reward, providing a link to multi-task RL.

In general, hRL approaches learn a high-level *controller*, which chooses options to optimise the main reward, where each option is learned and executed by a *worker*, thus effectively forming a two-level hierarchy [21, 22, 23]. Alternatively, we can build an *Option hierarchy* [24, 25]: a sequence of sets of options $(\Omega_1, \Omega_2, ..., \Omega_n)$, where the action space for options $\omega \in \Omega_i$ is Ω_{i-1}, and the action space for Ω_1 is $\mathcal{A}$. In the following, we use the terms *option* and *skill* interchangeably.

2.2. Information Theory

Information theory provides a framework for the mathematical analysis of the coding of information, for storage, quantification or communication. In the paper that created the field [26], Claude Shannon demonstrated the unity of information media.

In RL, information-theoretic measures allow us to formalise very high-level ideas about how we want the agent to behave. Two such measures which are particularly relevant, are *entropy* $\mathcal{H}(\cdot)$ and *Mutual Information (MI)* $\mathcal{I}(\cdot;\cdot)$.

The entropy of a random variable (such as of the policy $\pi(\cdot|s_t)$) quantifies the uncertainty contained in it. A policy with maximal entropy follows a uniform distribution over the action-space $\mathcal{A}$ for a given s_t, whereas a policy with minimal entropy is deterministic, i.e. $\pi(a|s_t) = 1$ for some $a \in \mathcal{A}$ and $\pi(a'|s_t) = 0$ for $a' \neq a$. *Maximum-entropy RL* aims to find a balance between maximising rewards and using various actions to achieve this, by augmenting the standard RL reward with an entropy term:

$$\pi^* = arg \max_\pi \mathbb{E}_{\tau \sim \pi(s_0)} \sum_{t=0}^{T} [r_t + \alpha \mathcal{H}(\pi(\cdot|s_t))]$$

The temperature parameter α controls the stochasticity of the policy, with higher α more strongly encouraging exploration. Prior works have found Maximum-entropy RL not only enhances exploration, but leads to more robust policies and improved learning speed [27].

We note that although the definition of discrete entropy is drawn from the axioms of information theory, this is not true for continuous (or differential) entropy, and it loses some

of the appealing properties of the discrete form (e.g. non-negativity) [28]. However, the relative entropy between two continuous random variables and their mutual information retain their validity.

The MI between two variables refers to the amount of information we can gain (or the reduction in uncertainty) about one variable by measuring the other. It is thus also referred to as *Information gain*. MI can be expressed in terms of entropy:

$$\mathcal{I}(X;Y) = \underbrace{\mathcal{H}(X) - \mathcal{H}(X|Y)}_{forward\ MI} = \underbrace{\mathcal{H}(Y) - \mathcal{H}(Y|X)}_{reverse\ MI} \tag{2.5}$$

Because MI measures the mutual dependence between two variables, it is symmetric ($\mathcal{I}(X;Y) = \mathcal{I}(Y;X)$). One useful way of characterising MI, is as the difference between the probability distributions induced (here) by the random variables X and Y. This can be computed in terms of the *Kullback-Leibler Divergence*:

$$\mathcal{I}(X;Y) = D_{KL}(p(X,Y)||p(X)p(Y)) \tag{2.6}$$

This measures how similar the joint distribution is to the product of the marginal distributions, and thus 'how independent' X and Y are.

2.3. Variational Autoencoder

Variational Autoencoders (VAE) [29] are a type of deep generative model. Autoencoders are used to learn a lower-dimensional representation of the input data. For example, if we are trying to label images of handwritten digits, a strong candidate for such a lower-dimensional representation would be the digit itself. This can be achieved by creating a neural network which attempts to reconstruct the input as accurately as possible after passing through a bottleneck layer (figure 2.1).

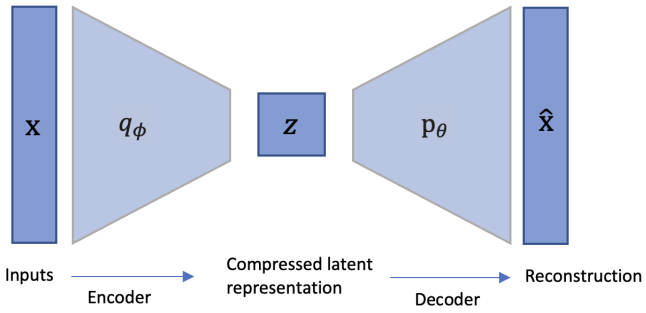


Figure 2.1.: A simple autoencoder

For reconstruction to work, we assume that the network must learn statistical regularities in the input data, allowing it to form a compressed representation at the bottleneck layer.

In this, we assume that the training data $(x_i)_{i=0\ldots n}$ is sampled from some underlying generative process $p(x|z)$. If the input data were handwritten digits, we might expect $p(z)$ to model the digit itself, as well as other properties like the slant or size. We would like to use our training data to infer the characteristics of $p(z)$, i.e. we want to compute $p(z|x)$. Bayes formula gives:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} \tag{2.7}$$

Unfortunately, this requires the computation of $p(x)$, which is generally intractable. The VAE framework circumvents this by approximating $p(z|x)$ (encoder) and $p(x|z)$ (decoder) with parameterised functions, and deriving the following optimisation problem via variational inference:

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \left\{ \underbrace{\mathbb{E}_{p_\theta(z|x)}\left[\log p_\phi(x|z)\right]}_{\textit{reconstruction loss}} - \beta \underbrace{D_{KL}\left(q_\theta(z|x) \;||\; p(z)\right)}_{\textit{regulariser}} \right\} \tag{2.8}$$

Therefore, we want to maximise the probability of reconstructing the input data x (reconstruction loss), and make the latent distribution $q_\phi(z|x)$ as similar to our prior latent distribution $p(z)$ as possible. A common choice for $p(z)$ is the standard multivariate normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{1})$, because this allows for the analytic computation of the regularisation term. In equation 2.8 we adopt the more general β-VAE [30] framework, which adds a weight to the regulariser.

The regularisation of the latent space allows us to generate samples from $p(x|z)$ simply by sampling from our latent prior $p(z)$. This is not possible with a regular autoencoder, because the latent space it produces is highly irregular, thus sampling from it is unlikely to yield useful reconstructions.

2.4. Skill Discovery

Skill discovery algorithms lie at the intersection of the ideas discussed in the previous section. Skill discovery methods aim to learn skills in a *task-agnostic* manner, and assume that these can be useful for later tasks (given by an external reward). Thus, they provide a framework for learning from completely unsupervised interaction with the environment. This inverts the usual process of skill formation, which relies on an external reward signal to determine which temporally extended behaviours are useful, preventing the agent from learning behaviours that aren't immediately useful for the given task. A recent literature review terms such

methods Intrinsically Motivated Goal Exploration Processes (RL-IMGEP) [31], and we follow their definitions closely to place skill discovery within the broader context of RL research.

First, we define skills as represented by a skill embedding z and skill-conditioned reward function r_z which measures the agents progress towards successfully performing z. Each skill embedding z implies a skill policy $\pi(\cdot|\cdot, z)$, which we refer to simply as a skill when executed for a certain number of time-steps $T > 1$. As in the classical Options framework, skills then correspond simply to temporally extended behaviours. From this, we may define the *skill-MDP*: $\langle \mathcal{S}, \mathcal{Z}, \mathcal{A}, r_{\mathcal{Z}}, \mathcal{T}, p(s_0) \rangle$, imposing the skill structure on the reward-free MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, p(s_0) \rangle$. In this, we have used the word skill abstractly to refer to any behaviour the agent might wish to engage in. Goal-reaching in the sense of attaining some final state is therefore only a subset of the broader set of possible skills, encodable in some reward function r_z.

We can now describe some different branches of RL-research introduced in section 2.1 as special cases of this skill-MDP. We may view classical RL as a skill-MDP with $|\mathcal{Z}| = 1$ and a single reward-function the agent aims to optimise, both of which are given (not learned). In intrinsic motivation, neither goal nor reward are given, and rather defined by the researcher to reflect a property of agent behaviour which may be useful across different MDP's (like exploring the environment). Skill discovery emerged naturally from intrinsic motivation by extending it to learning a similar objective across multiple skills/tasks, where each skill is learned to be different from others.

This is easily extended into goal-conditioned RL, where $|\mathcal{Z}| > 1$. Here, $\mathcal{Z}$ is usually a subset of $\mathcal{S}$ (learned or given), and $r_{\mathcal{Z}}$ is some distance metric relating the current state of the agent with the goal state. Thus, skills are simply goal-reaching behaviours.

Multi-task RL doesn't cleanly fit the skill-MDP, because it may introduce variability in the state- and action- spaces, and the transition dynamics between tasks. The focus of multi-task and meta-learners is on formulating an algorithm that can exploit the common structure of the given tasks to learn more quickly on unseen tasks. Although skill discovery approaches learn to propose their own tasks (in a more limited sense, in that only the reward function may vary), they must also learn to generalise well over these skills. Therefore advancements in multi-task and meta-RL may be seen as complementary.

Skill discovery further reduces the amount of human engineering required in the design of the agent. As an input, skill discovery methods expect the reward-free MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, p(s_0) \rangle$. Unlike GCRL, the skill space $\mathcal{Z}$ is independent of the state space, taking on the more abstract role of skill embeddings. A skill discovery method maps $\mathcal{Z}$ to some skill-conditioned reward function $r_{\mathcal{Z}}$, and this mapping defines what constitutes a skill. By sampling diverse $z \in \mathcal{Z}$ and learning corresponding skills, the agent learns to propose and solve its own tasks.

2.4.1. Variational Option Discovery

We concentrate on a number of recent approaches motivated by information-theoretic objectives that define what may constitute a skill, and term these Variational Option Discovery (VOD) algorithms following [6]. The objective defines a mapping between the skill embedding z and some property of a trajectory τ, like starting and final states s_0 and s_T ([3]), transitions (s, s') within τ ([5]), or even just states s ([4]). Broadly, the objective should specify what information flows into the this skill mapping, and what information should be ignored. Implicit in the examples given above is that we ignore the actions taken by the agent, when mapping skill embeddings to trajectories (for example because we don't want to distinguish trajectories in which actions don't have any effect on the state). If we choose to distinguish trajectories only by s_0 and s_T, we ignore all states $s_1, ..., s_{T-1}$ inbetween.

We may define a corresponding function $f(\tau)$, which we term the *trajectory preprocessing function*. We visualise f for the objectives given above in figure 2.2, and provide an overview in table 3.1.

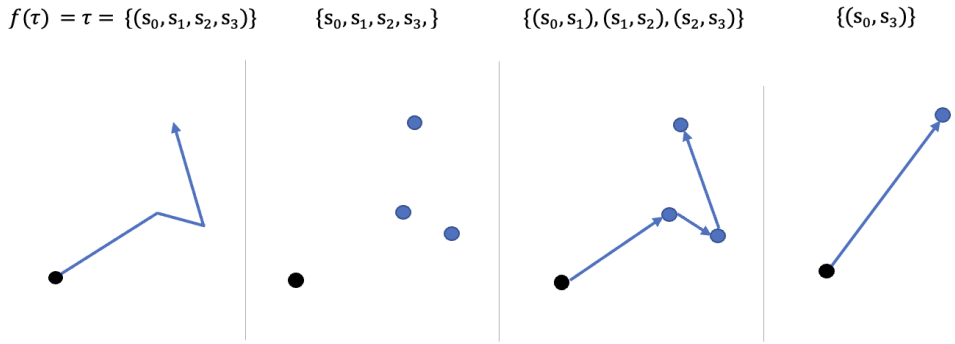


Figure 2.2.: Visualising different $f(\tau)$ for a short trajectory in 2-d.

With $f(\tau)$ we can formulate the general objective:

$$\mathcal{F}(\theta) = \mathcal{I}(z; f(\tau)) = \underbrace{\mathcal{H}(z) - \mathcal{H}(z|f(\tau)}_{\textit{reverse MI}} \tag{2.9}$$

$$= \underbrace{\mathcal{H}(f(\tau)) - \mathcal{H}(f(\tau)|z)}_{\textit{forward MI}} \tag{2.10}$$

Most approaches choose the reverse form of the MI, because the distribution $p(f(\tau))$ is generally intractable, whereas $p(z)$ can be chosen and fixed, and its entropy therefore easily computed. This choice also determines whether our skill model is discriminative (mapping trajectory information to skill embeddings) or generative (allowing us to use the skill embedding to predict trajectory information): in simpler terms, whether z is an input or the output of the skill mapping.

The approaches are united by a *diversity assumption* [32]:

Diversity assumption: Each skill should produce repeatable/predictable trajectories in the environment, and should be different from other skills.

This can be inferred easily from equation 2.9: while we maximise $\mathcal{H}(z)$ (meaning we want the overall distribution over skill variables to be as uniform as possible), we minimise $\mathcal{H}(z|f(\tau))$, implying that the mapping from trajectory τ to skill embedding z should be as close to deterministic as possible.

In the reverse form of the MI (discussed in the following, but the forward case is analogous), we train a probabilistic discriminator model $q_\phi(z|f(\tau))$ to map trajectories τ to skills z. In practice, this is done with a Multi-layer-perceptron (MLP) or a Recurrent Neural Network (RNN). We then use this model to compute intrinsic rewards $r_z(f(\tau))$ with which we train the *skill-conditioned* policy. A common choice for the skill-rewards is then $r_z(f(\tau)) = -\log q_\phi(z|f(\tau))$. Thus, the policy is rewarded for producing rollouts which reflect the skill mapping learned by the discriminator. This process is repeated until the discriminator and policy converge, and is summarised in algorithm 1.

Algorithm 1: Variational Option Discovery

Result: Policy $\pi_\theta(a|s,z)$
Given: skill prior $p(z)$;
Initialize policy $\pi_\theta(a|s,z)$, discriminator $q_\phi(z|f(\tau))$;
while *not converged* **do**
  collect skill-trajectory pairs $\mathcal{D} = (z^i, \tau^i)_{i=1,\dots,N}$ by first sampling a skill $z \sim p(z)$ and then rolling out a trajectory τ in the environment, $\tau \sim \pi_\theta(\cdot|\cdot, z)$;
  update discriminator via supervised learning to maximise $\mathbb{E}[\log q_\phi(z|f(\tau))]$ using $\mathcal{D}$;
  compute intrinsic reward $r_z(f(\tau))$ using q_ϕ and relabel trajectories in $\mathcal{D}$;
  update π_θ using any RL algorithm;

In practice, many algorithms compute skills on a per time-step basis, rather than on complete trajectories, meaning that $f(\tau)$ decomposes τ into its constituent time steps. This arguably trades off conceptual validity for algorithmic simplicity. Intuitively, we can also more quickly collect individual (time-step, skill)-pairs than (trajectory, skill)-pairs, so our models receive more training data.

2.4.2. Diversity is all you need (DIAYN)

DIAYN is one instance of this skill discovery approach. We briefly describe it here to offer a concrete example of ISD, and because this thesis builds on it. DIAYN distinguishes itself from other comparable approaches mainly by the minimality of its objective, built around the assumption that skills need only visit diverse states.

DIAYN maximises the mutual information between skills and states $\mathcal{I}(s;z)$, such that skills dictate the states the agent visits. $p(z)$ is fixed to a uniform categorical distribution, thus $\mathcal{Z}$ is discrete and $p(z)$ is the maximum entropy distribution over $\mathcal{Z}$. By mapping states to skills and introducing the inductive bias that each state should map to only one skill (in so far that this is possible), we are effectively creating a $|\mathcal{Z}|$-partition of the state space. Furthermore, DIAYN uses Soft Actor-Critic, an entropy-regularised RL algorithm to learn the skill-conditioned policy $\pi_\theta(a|s,z)$. This encourages nearby states to be grouped under the same skill, and skills to move away from each other to more easily discriminable states.

The full objective is therefore $\mathcal{F}(\theta) = \mathcal{I}(\mathcal{S};\mathcal{Z}) + \alpha\mathcal{H}(\mathcal{A}|\mathcal{S},\mathcal{Z})$, where α determines the intensity of entropy-regularisation and is a tuneable parameter.

The authors use a learned discriminator $q_\phi(z|s)$ to approximate the unknown distribution $p(z|s)$, and use this discriminator to variationally lower bound the objective as follows:

$$\begin{aligned}\mathcal{F}(\theta) =& \mathcal{H}(\mathcal{A}|\mathcal{S},\mathcal{Z}) - \mathcal{H}(\mathcal{Z}|\mathcal{S}) + \mathcal{H}(\mathcal{Z}) \\ =& \mathcal{H}(\mathcal{A}|\mathcal{S},\mathcal{Z}) + \mathbb{E}_{z\sim p(z),s\sim\pi(z)}\big[\log p(z|s)\big] - \mathbb{E}_{z\sim p(z)}\big[\log p(z)\big] \\ \geq& \mathcal{H}(\mathcal{A}|\mathcal{S},\mathcal{Z}) + \mathbb{E}_{z\sim p(z),s\sim\pi(z)}\big[\log q_\phi(z|s) - \log p(z)\big] = \mathcal{G}(\theta,\phi)\end{aligned}$$

Building on this, DIAYN roughly follows algorithm 1, although it performs model updates after every environment step, rather than after collecting a batch of trajectories. The authors show multiple ways to use their method which extend more broadly to skill discovery, therefore we describe them in section 2.4.4.

2.4.3. An analogy with VAE

Achiam et. al [6] draw an analogy between Information-theoretic Skill Discovery (VOD) and Variational Autoencoders, by giving a direct mapping between the β-VAE objective and the objective optimised by ISD. They term these approaches (described here as VOD) *Variational Option Discovery Algorithms*. Here, we use c to refer to the latent skill (context) variable to clarify differences with the VAE objective.

$$\pi^*, q_\phi^* = \arg\max_{\pi, q_\phi} \left\{ \mathop{\mathbb{E}}_{c\sim p(c)} \Big[\mathop{\mathbb{E}}_{\tau\sim\pi(\cdot|\cdot,c)} \underbrace{[\log q_\phi(c|\tau)]}_{\textit{reconstruction loss}} + \beta \underbrace{\mathcal{H}(\pi|c)]}_{\textit{regulariser}} \right\} \tag{2.11}$$

There is a direct correspondence between the input data x and the context variables c, reflecting their role as 'ground truth' in reconstruction training. Counterintuitively, this means that the trajectory τ maps to the VAE latent variable z. Regularisation is performed by the entropy term $\mathcal{H}(\pi|c)$, which encourages the policy to explore within each context. In practice, the VOD objectives require variational inference to be made tractable, resulting in these similarities.

This provides a useful framework for thinking about VOD. In this mapping, the discriminator q_ϕ is the decoder, or generative network, whereas the policy acts as the encoder, encoding latent variables into trajectories. Again, intuitively we would expect this to be the other way around, to align more closely with the information bottleneck principle the VAE is based on, since the skill embedding z represents a compressed form of the skill. However, this is a purely semantic difference, as illustrated in figure 2.3.

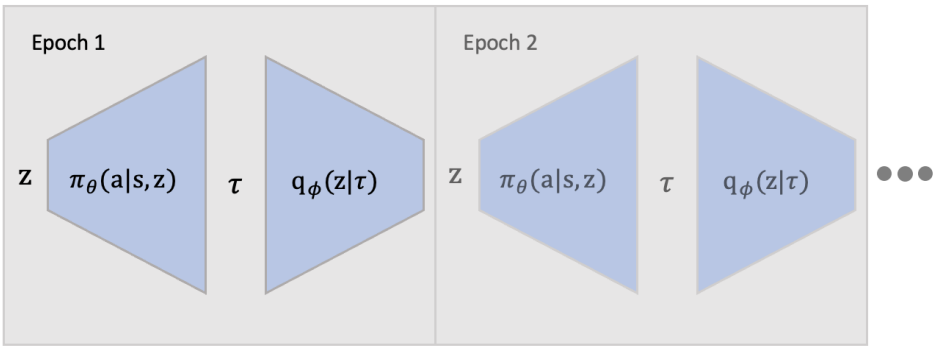


Figure 2.3.: The skill variable z is an information bottleneck in VOD.

The important difference is that we are performing reconstruction on the latent skill variables, whereas we are regularising the trajectories collected in the environment. This is because in skill discovery we assume knowledge of the generative latent factor (the skill embedding), whereas in the VAE framework we aim to discover it.

The β-coefficient in the VAE framework corresponds to the α-coefficient weighting the entropy-regularisation term. This provides another useful intuition. Like β, α determines the level of *disentanglement* between skills [30], and thus increasing α should lead to smoother latent space. If a skill behaves more randomly, a wider range of trajectories is grouped under this same skill, and there is likely a large amount of overlap between skills 'close' to one another. If we set the entropy to 0 (or make it negative for continuous $\mathcal{Z}$), it would not be difficult for the policy to learn trajectories for each skill that are completely disentangled, i.e. that have no overlap. However, this would also limit the meaningfulness of the learned

skills, as closeness in the latent space might not correspond with any notion of closeness in the trajectory space. This is problematic, since skills necessarily derive their meaning within the context of other skills. For example, DIAYN with one skill can never escape random behaviour.

This analogy gives us another way of thinking about skill discovery and justifies trying to leverage advances made in the training of VAEs for VOD.

2.4.4. Applications

There are a number of other ways to leverage the skill-conditioned policy $\pi_\theta(a|s,z)$ and discriminator $q_\phi(z|\tau)$ learned during skill discovery [4].

1. **Options in HRL**: We may elevate the agents action space by replacing (or augmenting) it with learned skills, hopefully simplifying later tasks.

2. **Finetuning**: We can initialise an agent policy with the weights θ of the skill-conditioned policy, similar to the use of pre-trained models in computer vision. Although this is a conceptually weaker idea because it throws away the structure of the learned policy, it may improve sample efficiency.

3. **Adding supervision**: Skill discovery also provides a straightforward avenue for introducing human supervision into the learning process. For example, we might learn skills $q : f(\mathcal{S}) \to \mathcal{Z}$ from some function f of the observations. In the locomotion case, f could map the current state to the agents center of mass, and therefore learn skills that effectively manipulate the agents center of mass. Alternatively, f could simply constrain the state to the (x, y) coordinates of the agent on the environments 'ground', to learn skills that move in different directions.
   Perhaps more importantly, it can make agent behaviours more intelligible (and therefore easier to debug) to humans. An agents solution to a given task would decompose to a shorter sequence of temporally extended behaviours, rather than a large number of actions that are difficult to assess individually.

4. **Imitation Learning**: We may want to allow our agent to learn from experience collected by another (usually more expert) agent. In imitation learning, the agent is provided with trajectories τ_e produced by such an agent, either consisting of state-action sequences $\tau_e = \{(s_t, a_t)\}$, or in the broader case observations $\tau_e = \{(o_t)\}$. Instead of trying to learn a policy that reflects the expert knowledge encoded in the trajectory, we can use the discriminator to infer the learned skill that most closely matches the trajectory by computing $z_* = \max_{z \in \mathcal{Z}}[q_\phi(z|\tau_e)]$.

5. **Model-based control**: If instead of a discriminator, we learn a dynamics model using the forward MI, we may use this model for planning in the latent space. DADS [5] learns $q_\phi(s'|s,z)$, allowing us to predict the next state s' given s and z.

3. Related Works

As described in section 2.4, skill discovery methods fall within a very broad context of RL techniques. Here, we will limit ourselves to a discussion of skill discovery methods based on maximising mutual information, as well as hierarchical reinforcement learning methods that use learned options and/or a multi-layer hierarchy. However, because of the broadness of the proposed method, there is overlap with many other research areas, not mentioned here. Intrinsic motivation is particularly relevant, as it may be viewed as the single-skill predecessor to skill discovery. [18] reviews works in intrinsic motivation, and [31] concentrates on a subclass of approaches at the intersection of intrinsic motivation and goal-conditioned RL.

3.1. Variational Option Discovery Algorithms

We describe the key design decisions in VOD methods, and give an overview of their variability. Most methods adopt the same iterative approach described in section 2.4, but another recent work *Explore, Discover, Learn* [33] calls this into question, and proposes an alternative structure.

1. **Objective**
   VOD methods employ a variety of objectives, and implement these by mapping skills to different elements of trajectories. We summarise some of these in table **??**.
   Because the authors of the different methods generally don't fix the other aspects of the

Algorithm	Objective	$f(\tau)$
DIAYN [4]	$\mathcal{I}(s;z)$	$(s_0),(s_1),...,(s_T)$
VIC [3]	$\mathcal{I}(s_T;z\|s_0)$	(s_0,s_T)
VALOR [6]	$\mathcal{I}(\tau;z)$	$(s_0,s_{11}-s_0,s_{22}-s_{11},s_T-s_{T-11})$
DADS [5]	$\mathcal{I}(s';z\|s)$	$(s_0,s_1),(s_1,s_2),...(s_{T-1},s_T)$
DISCERN [34]	$\mathcal{I}(g;s_T)$	(s_T)
Skew-Fit [35]	$\mathcal{I}(s;g)$	$(s_0),(s_1),...,(s_T)$

Table 3.1.: VOD algorithms and their objectives

   algorithms, it is difficult to assess what difference the chosen objective makes. However,

it implies the trajectory decomposition $f(\tau)$, and thus what training data the skill model receives during supervised learning, both in terms of the dimensionality of inputs and outputs and the amount of training data.
Subsequent to choosing an objective, we must still choose the forward or reverse MI to optimise it. Prior works have generally chosen the reverse MI because it does not involve the intractable $p(f(\tau))$, with DADS and EDL being examples of exceptions.

2. **Reward function mapping (Skill Model)**
We then perform supervised learning with the skill-trajectory pairs (τ, z) by preprocessing according to $f(\tau)$ and rearranging according to the chosen forward or reverse form of the MI. For example, in DADS training data is of the form $((s, z), s')$, whereas in DIAYN it is of the form (s, z).

   a) **Skill prior**
   In most prior works, the skill prior $p(z)$ over the skill space $\mathcal{Z}$ is chosen and fixed beforehand, although in VIC the authors experiment with learning $p(z)$. They argue that if distinct skill embeddings lead to very similar trajectories, this should be reflected in $p(z)$. However, in DIAYN, the authors point out that this leads to adverse training dynamics, wherein only a small subset of the skill space is actually trained. Since skill discovery is a fundamentally bottom-up approach, it is not really clear why we would choose to learn $p(z)$.
   Since we want to maximise the entropy $\mathcal{H}(p(z))$ in the reverse form of the MI, the obvious choice is a uniform distribution over $\mathcal{Z}$, and this is adopted by most prior works. In most prior works, $\mathcal{Z}$ is chosen to be discrete, with the number of skills $N = |\mathcal{Z}|$ left as a hyperparameter. N induces a tradeoff: large N are difficult to learn but have the potential to be more descriptive (due to the diversity assumption), whereas small N are easier to learn but less meaningful. In VALOR, the authors propose a *curriculum approach* by which N is slowly increased during training as the skill model becomes more confident. They found that this stabilised training.
   The authors of DADS show how to perform skill discovery with a continuous skill space $\mathcal{Z} = [-1, 1]^D$ with dimensionality D, with larger D implying a more expressive skill space, similar to N in the discrete case.

   b) **Model Architecture**
   The skill model is usually a neural network. DIAYN uses a simple MLP, DADS uses a mixture-of-experts MLP (probably because it is learning a more complex function), and in VALOR the authors use a recurrent LSTM model. This is generally a consequence of the chosen objective and skill prior, and the structure of the training data implied by these. The authors of VALOR were unable to identify a meaningful difference between the skills learned with their recurrent model and those learned by DIAYN.

c) **On-/Off-Policy samples**
In principle, VOD methods require on-policy samples. This is because the skill model is trained on (τ, z) pairs, and defines the reward function for RL training. Assuming that skill model and policy are still improving, we do not want to update them with outdated training data. The authors of DADS show how to extend it to use off-policy experience when training discriminator and policy by introducing an importance sampling ratio [36]. However, other works [22, 8, 37] are unable to verify that importance sampling actually confers any benefit, and suggest that off-policy samples may be used without any correction.

3. **RL algorithm**
In general, these approaches specify "any RL algorithm", and then choose a popular instance like Proximal Policy Optimization (PPO) [38] or SAC in their implementation. There are a few traits of RL algorithms that may inform this choice, such as whether they are on- or off-policy, or whether they presume a discrete or continuous action space. The authors of DIAYN and DADS emphasise the use of entropy regularisation, which may play a special role in skill discovery, as described in section 2.4.3. The integration of universal value function approximators [12] allowed the learning of multi-modal policies, and therefore the step from intrinsic motivation to skill discovery.

3.1.1. Explore, Discover, Learn

In VOD approaches, there is a circular dependence between the different components of the algorithm: the skill model requires a good policy to collect trajectory-skill pairs for supervised learning, and the policy requires a good skill model to provide a learnable reward function. DIAYN describes this as a *cooperative game*, in that the optimal solution requires the skill model and policy learner to push each other up. In practice, this approach leads to unstable optimisation targets which make learning more difficult. The authors of EDL argue that this leads to poor coverage of the state space.

In Explore, Discover, Learn [33], the authors separate the skill discovery process into 3 distinct stages: Exploration, *Skill discovery*, and Skill learning. VOD repeats each of these stages in every iteration, whereas each is executed to convergence once in EDL. The difference is illustrated in figure 3.1. For simplicity, like in the paper we consider only the DIAYN objective $\mathcal{I}(s; z)$ rather than the more general $\mathcal{I}(f(\tau); z)$ (as introduced in section 2.4).

To accomplish this, EDL uses an exploration method (SMM [39]) to learn the distribution of states to be encoded in skills, but we could use an oracle or expert trajectories instead. The skill model is learned as the decoder of a VAE, thus we are performing unsupervised learning rather than supervised learning, as in VOD. The final stage of learning a skill-conditioned policy remains the same, although in EDL this stage may be thought of as *embedding* the

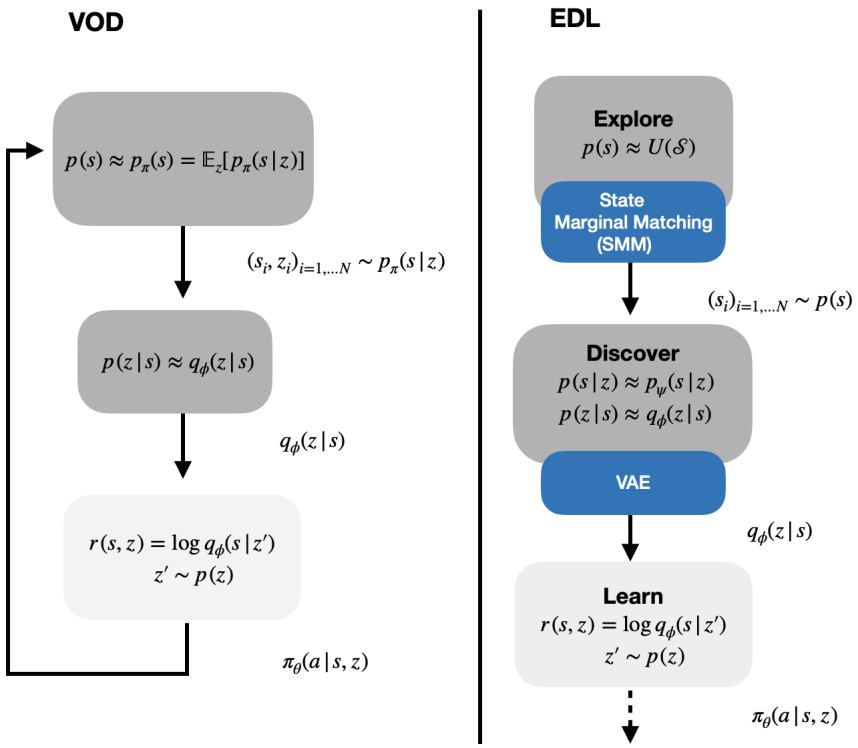


Figure 3.1.: EDL reduces skill discovery to three distinct stages

learned skill model in the MDP.

Thus, EDL modularises skill discovery. Each of the three stages may be evaluated effectively independent of the other stages, and new research in exploration and generative modelling may easily be integrated. The algorithm is overall more complex, in that we replace exploration with random rollouts from a given policy with a targeted exploration mechanism, and simple supervised learning of the decoder net with fully unsupervised learning. On the other hand, the iterative nature of VOD methods can make them very difficult to debug. We may also view EDL as an off-policy skill discovery method, whereas VOD methods are in principle on-policy. The approach taken in EDL allows us to perform each stage multiple times without having to return to the previous stage to do so, which should make for more sample-efficient and transferrable learning.

3.2. HRL with learned options

Since skills are analogous with options, they go hand in hand with hierarchy. VOD methods learn a single layer of the hierarchy, given the reward-free environment, which might then be used to optimise a task reward. In this section we describe how options can be learned within the context of hRL. We discuss two related works in more detail because they are particularly relevant to this thesis.

3.2.1. Latent space policies for hRL (SAC-LSP)

SAC-LSP [40] offers a general view of how *latent space policies* can be incorporated into a hierarchy. Here, we sequentially learn K layers of a hierarchy. For each layer, we learn a latent-conditioned policy $\pi_\theta(a|s,z)$ using a maximum entropy RL algorithm (like SAC) that maximises the layer-specific reward $\mathcal{R}_i$. $\mathcal{R}_i$ could be some low-level heuristic reward (for example encoding a navigation prior in locomotion environments), but also an VOD intrinsic reward. During training, latent variables $z \in \mathcal{Z}$ sampled from the latent prior $p(z)$ are augmented with the state. This allows us to use $\mathcal{Z}$ as the action space of the next layer: the agent selects latents z, and then samples an action from the given policy π_θ. Therefore, we embed π_θ in the MDP transition dynamics, exposing a new MDP within which the next task $\mathcal{R}_{i+1}$ should be easier to solve.

The authors describe two mechanisms for learning such a policy: layerwise, by training a single layer, freezing its weights, and then training the next layer on top of this, and end-to-end, learning all layers simultaneously. The authors show how to accomplish the latter using invertible transformations to embed latent policies in the environment, meaning that each layer can undo actions chosen by lower layers if they are detrimental to the current task reward.

SAC-LSP makes no assumptions about how latent variables z are sampled during training, i.e. about when an option can be initiated or when it terminates. Many of the VOD-approaches presented in section 3.1 test their algorithm in a hierarchical context by instantiating SAC-LSP: they train skills, then freeze the resulting skill-conditioned policy, and then use these skills to optimise some task reward.

3.2.2. HRL by discovering Intrinsic Options (HIDIO)

In HIDIO [8], the authors integrate VOD into a worker-scheduler hierarchy, instantiating an end-to-end latent space policy. In this, the higher-layer policy optimises some task reward, while the lower-layer policy learns skills in an unsupervised manner. However, in VOD skills

are learned without any knowledge of task reward; in HIDIO, skills are learned preferentially that achieve the task reward. The authors compare the bottom-up (VOD) and end-to-end approaches, and find that skills learned with VOD are often not useful in later achieving the task reward. Thus, they deem end-to-end training necessary. Although the interplay between worker and scheduler technically necessitates an on-policy RL algorithm, the authors also use SAC to allow reuse of training data. Because they introduce an external reward into the skill discovery process, HIDIO does not aim to perform unsupervised hRL, but rather uses VOD to enhance previous approaches to hRL.

Unlike works in VOD, HIDIO emphasises the importance of resampling options throughout the episode. This has the benefit that options don't radiate out from the starting state distribution $p(s_0)$, and that options must be learned such that they can effectively be sequenced. Without loss of generality, they assume an option is resampled every T steps. They also define a *meta-MDP* when a skill is initiated, in which each state stores the trajectory from the initation state to itself, which allows them to easily compare different mutual information based objectives.

3.2.3. Option Hierarchies

As demonstrated in SAC-LSP, a hierarchy can also be learned with more than 2 layers. Two recent works [25, 41] build option hierarchies in which the degree of temporal abstraction increases for every layer. They learn K policies, setting the option length T^l for policy π^l to be an integer multiple of the length for π^{l-1}. Thus, the option length in terms of steps in the environment is an exponential function of the form $T^i = c * T^{i-1}$. DEHRL [41] performs multi-layer unsupervised option discovery, as studied in this thesis, but uses a different technique to implement the diversity assumption.

Stochastic Neural Networks for HRL [7] builds a 2-layer hierarchy using stochastic neural networks, and uses a mutual-information objective $\mathcal{I}(z; c)$ relating skills z and the agents center of mass c. Rather than training both neural networks end to end, the authors find that training the top layer on a frozen lower one already lead to good results. In a later work [32] the authors show that introducing top-down feedback further improves performance.

Hierarchical Actor Critic [24] gives further insight into how all the layers in such a multi-layer hierarchy could be trained simultaneously and independently. Much of the paper describes the techniques used to enable stable learning in spite of the non-stationary transition dynamics at each layer, induced by the changing lower layers. They argue that if all lower layers $0, ..., i-1$ were already optimal, we should be able to train layer i on top of these, and the authors show how we can simulate this before the lower layers have been trained. The authors find that each additional layer improves performance.

4. Problem Statement

In this thesis, we aim to synthesise information-theoretic skill discovery and hierarchical reinforcement learning, to more effectively learn skills in an unsupervised manner. This is motivated by the intuition that skills reflect only a very abstract notion of a temporally extended behaviour, and that we may consider skills to be composed of other skills.

The general problem statement for skill discovery is as follows: given the reward-free MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, p(s_0) \rangle$, learn a skill-space $(\pi(\cdot|\cdot, z), \mathcal{Z})$. In VOD, a skill is executed by choosing the skill variable $\mathbf{z} \in \mathcal{Z}$, and rolling out the corresponding skill-conditioned policy $\pi(a|s, \mathbf{z})$ for a given number of time-steps $T > 1$. Thus, a skill discovery method is a function ψ:

$$\psi : (\mathcal{M}, T) \rightarrow (\pi(a|s, z), \mathcal{Z}) \tag{4.1}$$

In this definition, we emphasise the significance of skill length T. Fundamentally, for large T, the diversity assumption becomes easier to fulfil. We argue that this is undesirable, because the diversity assumption is the underlying mechanism by which ψ can learn skills. If skills are not forced to diversify, they continue to behave randomly. On the other hand, if T is smaller, using skills instead of primitive actions can only confer a smaller benefit.

We can extend the skill discovery problem statement to learning a hierarchy of skills. Thus, given the reward-free MDP $\mathcal{M}$, we want to learn a sequence of skill-conditioned policies $(\pi^i(a|s, z))_{i=1,\dots,K}$ and corresponding latent spaces $\mathcal{Z}_i$, where $z \in \mathcal{Z}_i$, $a \in \mathcal{Z}_{i-1}$, and $\mathcal{Z}_0 = \mathcal{A}$, and K is the number of layers in the hierarchy. To this end, we can alter the definition of ψ to return a modified MDP, in which we replace the given action space with the skill space: $\hat{\psi} : (\mathcal{M}, T) \rightarrow \langle \mathcal{S}, \mathcal{Z}, P_{\mathcal{Z}}, p(s_0) \rangle$. Here, $P_{\mathcal{Z}}$ describes the modified environment transition dynamics after embedding the learned skills. We may then apply a given skill discovery method ψ recursively to retrieve the hierarchical skill discovery method Ψ:

$$\Psi : (\mathcal{M}, \psi, (T_i)_{i=1,..,K}) \rightarrow (\pi^i(a|s, z), \mathcal{Z}_i)_{i=1,\dots,K} = \hat{\psi}(\hat{\psi}(...(\hat{\psi}(\mathcal{M}, T_1), ...), T_{K-1}), T_K) \tag{4.2}$$

Here, T_i corresponds to the skill length at the i-th layer of the hierarchy, counted in the number of policy decisions made (rather than the actual number of environment steps). Thus, given a skill discovery problem $(\mathcal{M}, T)$ which we could solve directly with ψ, we can specify K and a corresponding skill-length sequence $(T_i)_{i=1,\dots,K}$, ideally with $T = \prod_{i=1}^{K} T_i$. This way we can use the hierarchical method Ψ instead of ψ.

To ensure that ψ is usable in this hierarchy, the skills it learns must fulfil certain constraints:

1. $\hat{\psi}$ must be recursively applicable, i.e. $\hat{\psi}$ must return an environment to which ψ is applicable. Prior works that impose a discrete skill space on a continuous action space do not immediately fulfil this assumption, because RL-algorithms are usually only applicable to one of the two.

2. We should be able to sequence skills within an episode. This is equivalent to requiring that skills should retain their meaning outside the given starting state distribution $p(s_0)$.

As in SAC-LSP, we could also require that the function $f : f(\psi) = \hat{\psi}$ that embeds skills learned by ψ in the environment be invertible, so that each layer can undo actions of previous layers. We leave this to future work, and consider only the simpler layerwise training procedure, by which the weights of policies $\pi^1, ..., \pi^{i-1}$ are frozen when training the i-th layer policy π^i. If ψ does not learn skills that sufficiently cover the space of possible behaviours, then we may cripple an agent using these skills, an effect we can expect to be compounded with more layers. Layerwise training with weight-freezing may be thought of as the strictly bottom-up approach to learning a skill hierarchy.

4.1. Motivation

There are a number of reasons to introduce such a hierarchy into skill discovery. We present our intuition here, based on the notion of *skill discovery as a compression in the space of action-sequences of length T*. For the sake of argumentation, we consider the problem $(\mathcal{M}, T)$ with discrete and finite $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{Z}$. Consider the space τ^T of all trajectories of length T. Not all of these trajectories will be worth distinguishing depending on the environment, and we may group those that are *similar enough* into a skill. The objectives in VOD then define how we determine similarity: which trajectories can we group?

Under this intuition, the definition of a skill changes from being a useful behaviour, to merely being any temporally extended behaviour. The usefulness of a skill, like that of an action, is determined after it is already learned: when it is used. Many skills will turn out to be generally useless, but the policy using them should be able to learn which skill is useful, and where.

This also allows us to reason about the size of the space we are compressing. In the most general case, every possible trajectory is abstracted into some skill (we are performing lossless compression). We must sample these trajectories from the environment, therefore we argue that the problem of effectively encoding τ^T is in some way proportional to the size of this space, which is exponential in T. For example, if we want to learn skills of length 100 over

a 2-dimensional discrete action and state space, without making any further assumptions we have to consider 2^{100} sequences. If we decompose this problem into a 2-layer hierarchy with $T = (10, 10)$, we consider 2^{10} sequences in the first layer, and $|\mathcal{Z}|^{10}$ in the second. The cardinality of $\mathcal{Z}$ defines the compression. For example, for 100-dimensional $\mathcal{Z}$, this equates to a reduction in the total number of sequences considered of $\frac{2^{100}}{2^{10} + 100^{10}} \approx 10^{11}$. For $|\mathcal{Z}| = 2^{10}$, every possible action-sequence is assigned its own skill, and there is no reduction in complexity. In this way, the introduction of hierarchy carries with it the potential to exponentially reduce the size of the search space, and each layer should simplify the problem for the following layer.

As a result, the discriminability objective is satisfied at different scales. If it is satisfied for layer i, i.e. we have performed a sufficiently lossless compression, we assume that we can repeat this process for layer $i + 1$, using the skills of layer i. This gives us more flexible control over the problem-size (skill length) T_i of each layer. Moreover, we should see improved exploration when using temporally extended behaviours [42], and therefore better coverage of the state space. Again, this is because the compressed search space should result in a simpler problem for the agent.

4.2. Success Indicators

There are no clear metrics (like a task-specific reward) to evaluate the success of our final algorithm, thus we will propose some indicators of success and otherwise restrict evaluation to qualitative judgements. Because skills are learned in a task-agnostic fashion, the only way to evaluate them effectively is to determine how pretraining with them can improve learning in different problem settings (like imitation learning, multi-task learning, or sparse-reward tasks). We use DIAYN as a baseline, because it is simple to implement and representative of VOD methods.

1. Discriminator confidence $\mathbb{E}_{\tau \sim D}[\log q_\phi(z|\tau)]$: the discriminator should be able to map trajectories to skills confidently.

2. Cumulative intrinsic reward achieved by policy: if the policy effectively *generates* the skills learned by the discriminator, it achieves higher intrinsic reward.

3. State space coverage: the agent should learn skills that sufficiently cover the state space. This measures the exploration behaviour of the algorithm, and will have to be evaluated qualitatively in most environments.

4. Sample efficiency: we measure sample efficiency mostly by the number of environment

steps taken by the agent, as this is the usually the most time-consuming part of learning.

5. Stability: the random seed and stochasticity of the algorithm should not significantly affect success according to the described indicators.

6. Performance on a set of evaluation tasks: we may compare a regular agent with one that received pretraining via unsupervised skill discovery, and compare asymptotic performance and sample efficiency over a series of test-tasks. If skills are truly task-agnostic, we would expect them to lead to performance gains on most/all tasks.

Since we can solve the same problem with ($K > 1$) and without ($K = 1$) a hierarchy, the success indicators are the same for the hierarchical as for the flat skill discovery method.

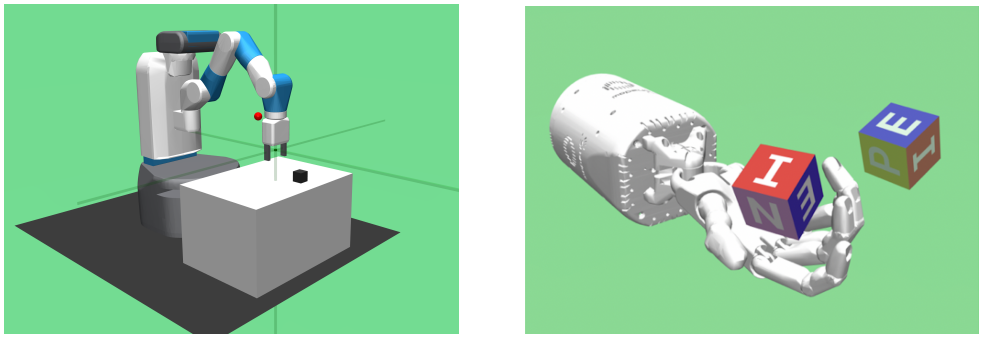
4.3. Environments

We will being by thoroughly evaluating our method in a simple 2d-navigation environment (2dNav), because we can immediately inspect learning with a single image. We can reason more easily about what we would like our skills to look like if our agent learned optimally. This allows us to discuss the impact of the various hyperparameters, and hopefully provide us with some intuitions for less easily inspected environments. This environment is also easily extended into higher dimensions (e.g. 3dNav). Actions directly manipulate the position of the agent. Thus, for d-dimensional navigation, $dim(\mathcal{S}) = dim(\mathcal{A}) = d$.

However we will also run the agent in the OpenAI Gym robotics environments, built on the MuJoCo physics engine. These environments are significantly more complex, but also allow for much more interesting and semantically diverse skills, and better reflect the target environments of this algorithm. Here we have two environment-classes: Fetch and Hand (figure 4.1)[1].

Both of these come with different tasks, or goals, defined with sparse rewards (rather than the shaped rewards used in previous locomotion environments like the OpenAI Gym Halfcheetah, Ant or Humanoid). In the Fetch environments, we manipulate a robotic arm to reach a goal position or move around a puck with the gripper. The state of the Fetch arm is 25-dimensional and goals are 3-dimensional, specifying a target position for the block or gripper. The Hand environment features a robotic hand, with similar position- and object-manipulation based goals. This makes them particularly interesting for skill discovery because we can use the different goals for meta-testing. For example in the Hand environments, we may learn skills that manipulate only the hand, and then finetune these in the reach or object manipulation tasks. In the goal-free Hand environment without any objects, $dim(\mathcal{S}) = 63$

[1] https://openai.com/blog/ingredients-for-robotics-research/



(a) FetchPickAndPlace-v0 (b) HandmanipulateBlock-v0

Figure 4.1.: OpenAI Gym Robotics environments

and $dim(\mathcal{A}) = 20$. Hand environments that manipulate objects expose 7-dimensional goals, and in the HandReach environment a 15-dimensional goal specifies the target end positions of all fingertips.

5. Hierarchical Skill Discovery

In this section, we describe our algorithm for hierarchical skill discovery (hVOD). The approach closely mirrors prior related works in implementation details, with the fundamental goal being to learn composable skills via an algorithm that is recursively applicable to the same MDP. In this thesis, we attempt to develop the simplest instantiation of the class of methods introduced in the problem statement. To this end, we begin by describing our hierarchical method Ψ, and then the underlying skill discovery method ψ, which we call cDIAYN to highlight its similarity with the prior work. A summary of the approach is given in figure 5.1.

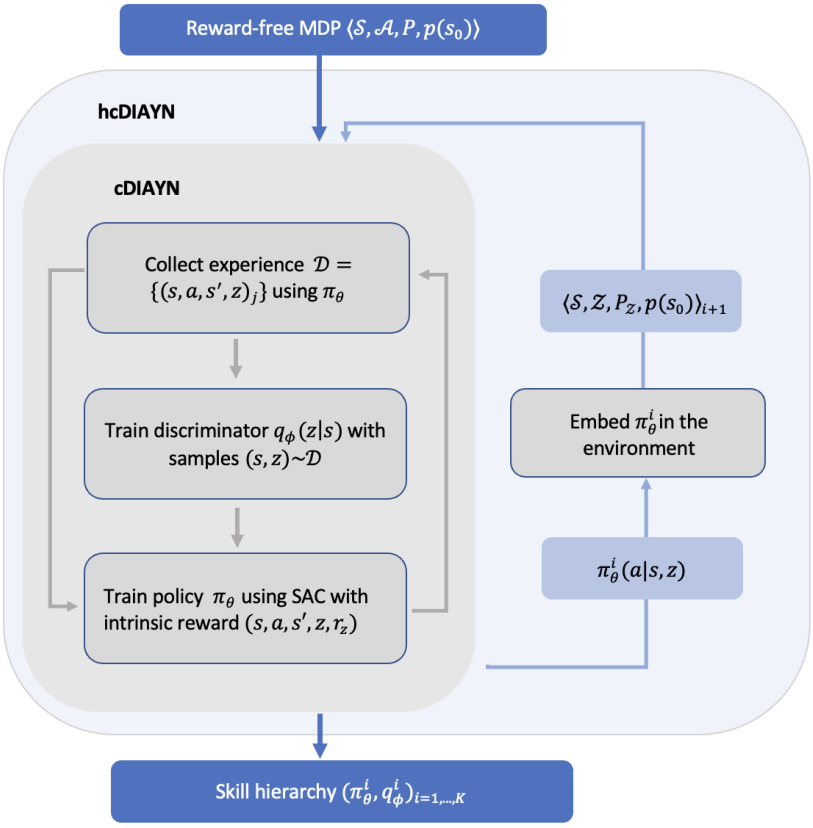


Figure 5.1.: Schematic overview of hcDIAYN, as instance of layerwise approach of building a hierarchy of skills via VOD

5.1. Deep Skill Hierarchy

In this thesis, we choose to train our hierarchy in a layerwise manner, without feedback from higher layers being passed down. We consider this a reasonable assumption in the skill discovery domain, which is fundamentally bottom-up, although it may limit the expressiveness of higher layers, especially if the skill discovery method ψ is only partially successful. The pseudocode for this training procedure is given in algorithm 2.

Algorithm 2: Learn skill hierarchy

Result: Deep Policy Hierarchy $(\pi^1(a|s,z_1), \pi^2(z_1|s,z_2), ..., \pi^K(z_{K-1}|s,z_K))$
Given: base-MDP $\mathcal{M}_0 = \langle \mathcal{S}, \mathcal{A}, p, p(s_0) \rangle$, number of layers K, skill-length function T_i, Skill discovery method $\psi : \mathcal{M}, T \rightarrow \pi_\theta(a|s,z), \mathcal{Z}$;
Initialise ;
for $i \leftarrow 1$ **to** N **do**
  $\pi^i(z_{i-1}|s,z_i), \mathcal{Z}_i = \psi(\mathcal{M}_{i-1}, T_i)$;
  Embed π^i in $\mathcal{M}_{i-1}$ to get skill-MDP $\mathcal{M}_i = \langle \mathcal{S}, \mathcal{Z}_i, p_i, p(s_0) \rangle$;

In this, we have also made the assumption that the action space for each layer is only the skill space of the previous layer. On the other hand, we could formulate the minimally restrictive action space $\mathcal{Z}_i = \mathcal{A} \cup \mathcal{Z}_1 \cup ... \cup \mathcal{Z}_{i-1}$, . In this case, we are using skill discovery merely to augment the action space of the agent, and we are effectively learning a worker-scheduler hierarchy, rather than a deep skill hierarchy. We observe a tradeoff between restricting the agents action space (and therefore search space), and ensuring that the behaviours the agent can learn are not limited.

This leaves two choices, in the number of layers K, and the layer-skill-length T_i, under the constraint that $T \approx \prod_{i=1}^{K} T_i$. A reasonable choice for the skill layer length is a simple exponential function $T_i = c * T_{i-1}$, where $c \in \mathbb{N}$ and $T_0 = 1$, for example $c = \lfloor \log_K T \rfloor$. Thus, in layer j, the agent takes $\prod_{i=1}^{j} T_i$ steps in the environment, but only makes T_j policy decisions. As described in [41], we may think of more appropriate sequences with information about the environment.

In this hierarchy, we have implicitly assumed that every skill may be initiated in every state, and that every skill terminates after exactly the layer-specific number of steps. In the options framework, we have assumed that for every option $\omega = (I_\omega, \pi_\omega, \beta_\omega)$ of layer i, the initiation set is $I_\omega = \mathcal{S}$ and the termination probability is $\beta_\omega(t) = 1 \;\; if \;\; t == T \;\; else \;\; 0$. The first assumption is reasonable, because we can let the agent using the learned skills determine this in a task-specific manner (even if that task is skill discovery). A prior work [8] argues that a constant skill length can be assumed without loss of generality, although we are not sure how

to reproduce this argument. However, one would expect that if the constant skill length is in some way limiting, it is less so for smaller T.

This allows us to visualise each skill as a tree, with degree T_{K-i} on level i. Figure 5.2 shows a hierarchy with 3 layers and with skill lengths $T_i = 2$, which we compare with the approach tested in prior works in VOD in figure 5.3.

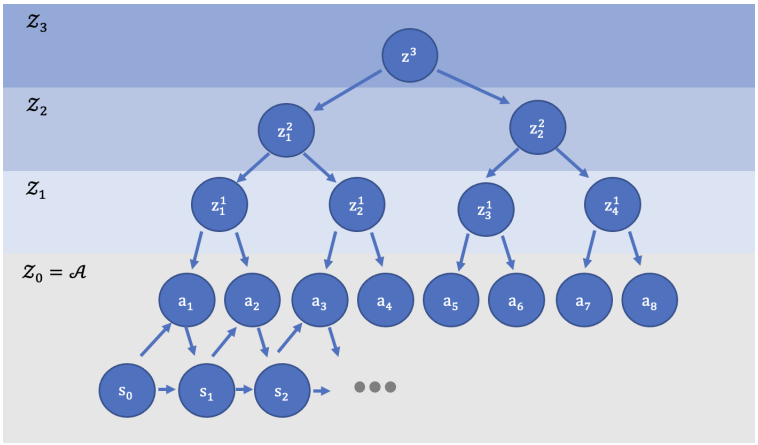


Figure 5.2.: Deep skill hierarchy with $K = 3$ and $T_i = (2, 2, 2)$

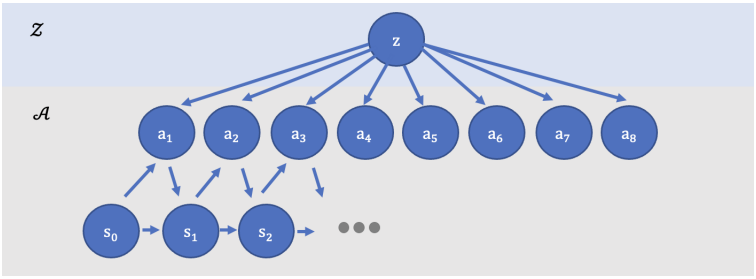


Figure 5.3.: Prior work in VOD, as a special case with $K = 1$ and $T = 8$

This interprets the approach taken in prior work in VOD as the single-layer case of the proposed hierarchical method. Figure 5.2 also provides an appealing visualisation of the bottom-up nature of deep skill hierarchies.

5.2. cDIAYN

In this section, we develop a skill discovery algorithm that satisfies the constraints put forward in section 4. To this end, we extend DIAYN to a continuous skill space $\mathcal{Z}$, and propose a simple state normalisation mechanism to to force the agent to infer skills from

state-differences rather than states. The pseudocode for our continuous, composable version of DIAYN (cDIAYN) is given in algorithm 3.

Algorithm 3: cDIAYN

Result: Policy $\pi_\theta(a|s,z)$, Discriminator $q_\phi(z|s)$
Given: skill prior $p(z) \sim \mathcal{Z}$;
Initialize policy π_θ and discriminator q_ϕ;
while *not converged* **do**
  Collect M transitions (s,a,s',z) from the environment, $a \sim \pi_\theta(\cdot|s,z), s' \sim p(s,a)$, resampling $z \sim p(z)$ with state-normalisation every T time-steps and resetting the environment after H time-steps;
  Update discriminator via supervised learning to maximise $\mathbb{E}[\log q_\phi(z|s)]$ using collected transitions;
  Relabel collected transitions with intrinsic reward calculated according to equation 5.3;
  Update π_θ using Soft Actor-Critic;

5.2.1. Objective

As described in section 3, there are many possible choices for the information-theoretic objective. Broadly, we want to learn skills that are clearly discriminable from one another, while each individual skill is in some form predictable. We use the same objective proposed in DIAYN, because it is conceptually the simplest.

Therefore, we choose to maximise the mutual information between states and skills $\mathcal{I}(\mathcal{S};\mathcal{Z})$. As in prior works, we further employ entropy-regularisation of the skill-conditioned policy $\pi_\theta(a|s,z)$, which encourages exploration and enhances robustness. This gives us the following complete objective:

$$\begin{aligned} \mathcal{F}(\theta) &= \mathcal{I}(\mathcal{S};\mathcal{Z}) + \alpha\mathcal{H}(\mathcal{A}|\mathcal{S},\mathcal{Z}) \\ &= \mathcal{H}(\mathcal{Z}) - \mathcal{H}(\mathcal{Z}|\mathcal{S}) + \alpha\mathcal{H}(\mathcal{A}|\mathcal{S},\mathcal{Z}) \end{aligned} \tag{5.1}$$

In this, we have chosen to optimise the reverse form of the mutual information.

5.2.2. Continuous Skill Prior

Unlike DIAYN, we adopt a continuous uniform distribution for $p(z)$ and choose $\mathcal{Z} = [-1, 1]^D$, where D represents the dimensionality of the skill space. We think that imposing a discrete prior on the skill space limits the expressiveness of skill variables $z \in \mathcal{Z}$, thus less effectively guiding agent behaviour. DIAYN learns skills that induce a uniform distribution over their respective partition of the state-space [4]. Unless you choose a large number of skills, it is likely that this will result in a very coarse guiding signal, at best moving through the partition in a coordinated way. We also believe that a continuous skill space can produce lower-variance (and therefore more predictable) behaviours (as in DADS) which are more useful in a hierarchy.

This continuous latent space allows for a more natural interpolation between different skills. Although prior works like DIAYN and VALOR show that we can interpolate between two discrete skills, much of the interpolating space should effectively be meaningless since it is not learned. By the interpolating space, we mean all vectors $[0, 1]^N$ where N is the number of skills, of which the space of one-hot vectors of length N is a subspace. In the continuous version, the entire latent space is utilised and meaningful. This also makes the latent space more compact. When using a discrete skill space, skills are usually encoded as one-hot vectors. The more skills we wish to learn, the larger the skill vectors.

Moreover, prior works build on Soft Actor-Critic (SAC), a state-of-the-art entropy-regularised RL algorithm, which assumes a continuous action space. Thus, learning a discrete skill space would preclude use of the same RL algorithm on the learned skill space in the hierarchy describe in section 5.1. This, along with the environments we intend to test the method in, is the main reason we chose a continuous skill prior. However, we should also be able to replace SAC with a discrete RL algorithm like DQN and learn a discrete skill space, or map actions from continuous to discrete (like DIAYN) and vice versa.

5.2.3. Optimising the Objective

The choice of a continuous skill space forces us to optimise the objective slightly differently. The authors of DADS showed how to learn skills with a continuous skill space, and we follow their approach. For completeness, the derivation is included in the following.

We may approximate $\mathcal{I}(\mathcal{S}; \mathcal{Z})$ as follows:

$$\begin{aligned}\mathcal{I}(\mathcal{S};\mathcal{Z}) &= \mathbb{E}_{z\sim p(z),s\sim\pi(z)}\left[\log\frac{p(z|s)}{p(z)}\right] \\ &= \mathbb{E}_{z\sim p(z),s\sim\pi(z)}\left[\log\frac{q_\phi(z|s)}{p(z)}\right] + \mathbb{E}_{s\sim p(s)}\left[\mathcal{D}_{KL}(p(z|s)\;||\;q_\phi(z|s)\right] \\ &\geq \mathbb{E}_{z\sim p(z),s\sim\pi(z)}\left[\log\frac{q_\phi(z|s)}{p(z)}\right]\end{aligned}$$

Here we have used the non-negativity of the KL-divergence to variationally lower bound the objective using a learned discriminator model $q_\phi(z|s)$ to approximate the intractable distribution $p(z|s)$. We can improve our approximation by increasing $\mathbb{E}\big[\log q_\phi(z|s) - \log p(z)\big]$ (*maximising the approximate lower bound*), and by minimising $\mathbb{E}\big[\mathcal{D}_{KL}(p(z|s)\;||\;q_\phi(z|s)\big]$ (*tightening the variational lower bound*).

The latter amounts to maximising the likelihood of samples from p (collected with rollouts in the environment) under q_ϕ. As in DADS, we can write the gradient for our skill model as follows:

$$\begin{aligned}\nabla_\phi\mathbb{E}_{s,z}\big[\mathcal{D}_{KL}(p(z|s)\;||\;q_\phi(z|s))\big] &= \nabla_\phi\mathbb{E}_{s,z}\left[\log\frac{p(z|s)}{q_\phi(z|s)}\right] \\ &= -\nabla_\phi\mathbb{E}_{s,z}\big[\log q_\phi(z|s)\big] \qquad (5.2)\end{aligned}$$

We can maximise $\mathbb{E}\big[\log q_\phi(z|s) - \log p(z)\big]$ under our policy π via reinforcement learning, by maximising with the following *intrinsic* reward:

$$r_z(s,a) = \log\frac{q_\phi(z|s)}{\frac{1}{2L}\sum_{i=1}^{L} q_\phi(z_i|s)}, \quad z_i \sim p(z) \qquad (5.3)$$

Here we approximate $\frac{1}{|\mathcal{Z}|}\int_{\mathcal{Z}} q_\phi(z|s)dz \approx \frac{1}{L}\sum_{i=1}^{L} q_\phi(z_i|s)$. In the discrete version of DIAYN, the one-hot categorical skill mapping incorporates the notion of mutual exclusivity of skills, i.e. that each state should only map to one skill. This is lost when using a continuous mapping, in that the discriminator is not immediately penalised for learning a high probability $q_\phi(z|s)$ for many z, given the same s. We compensate for this by the above *softmax*-like output normalisation. Thus, π is encouraged to produce transitions that are discriminable, i.e. that assign a high probability to only one skill. Equation 5.3 is maximised when the probability $q_\phi(z|s)$ is large, but $q_\phi(z_i|s)$ is small. Where, in the discrete case we can marginalise over the entire space $\mathcal{Z}$, in the continuous case we sample from $\mathcal{Z}$.

Unlike discrete DIAYN, it is not immediately clear how we could apply this softmax normalisation during discriminator training. However, this is where the cyclical nature of the described skill discovery methods arguably adds flexibility to the framework. Because the reward encourages the policy to produce trajectories that are discriminable, the discriminator receives more discriminable samples.

Although the previous derivation is analogous to the one performed in DADS, the justification is slightly different. In DADS, the authors approximate $p(s'|s) = \int_{\mathcal{Z}} p(s'|s,z)dz \approx \sum_{i=1}^{L} q_\phi(s'|s,z)$. This difference is just a consequence of our choice of the reverse MI. If we chose the forward MI, we would perform a similar marginalisation to approximate the intractable $p(s)$ using the policy induced distribution $\mathbb{E}_{z\in\mathcal{Z}}[p(s|z)]$. However, we choose the reverse MI to mirror DIAYN.

5.2.4. State Normalisation

Skills learned by prior methods are not composable, because during learning they sample a skill at the beginning of every episode and keep it constant throughout. Thus, skills radiate out from $p(s_0)$, and are therefore less meaningful when initiated outside this distribution. However, for skills to be composable, they should retain their meaning for broader starting state distributions.

As discussed in section 4.1, we think of skill discovery as a compression in the space of trajectories. As a result, and especially in this thesis, the skill length is an important hyperparameter, and longer sequences should increase the difficulty of the skill discovery problem. This is alleviated in prior works by formulating time-step based objectives, like $\mathcal{I}(s;z)$ and $\mathcal{I}(s';z|s)$, arguing that we can differentiate whole trajectories from them. The states of the skill-trajectory encoded in z are necessarily correlated, because we assume that z is sampled and fixed for a given number of time-steps T. Thus, if we assume skills are always sampled in some fixed state s_0, subsequent states s retain information about where in the trajectory they are (relative to the fixed s_0). When we resample z for different s_0, it becomes unclear what behaviour we wish the policy to fulfil: should they move towards the same target state? This is not necessarily possible in T time-steps, and we would expect it to be very different depending on the sampled s_0.

Behaviours learned by DIAYN and DADS are strongly tied to the states on which they are learned. This is why resampling skills during training breaks learning: the agent has no control over which skill is sampled. It also implies that in prior work in VOD, we can not use learned skills to reach out of distribution states, limiting the potential of learned skills for aiding exploration.

To resolve this, we propose to normalise states within a skill-MDP with the starting state.

Thus we allow policy and dynamics models access only to normalised states $\bar{s_i} = s_i - s_0$, where s_i is the state of the agent on the i-th time-step after initiating the current skill. As a result, $\bar{s_0} = \mathbf{0}$ for all skills, and each normalised state $\bar{s_i}$ contains trajectory information, because it relates s_i to the starting state. Thus, the skill variable z imposes a consistent skill-MDP around $\bar{s_0}$, and both models now learn to achieve repeatable **state-differences**. In the above examples, using a skill z is equivalent from any starting point, in that it should move through roughly the same normalised states $(\mathbf{0}, \bar{s_1}, \bar{s_2}, ...\bar{s_T})$. Thus, with state-norm our objective is effectively $\mathcal{I}(\Delta s; z)$, defined by the trajectory preprocessing function $f(\tau) = s_0, s_1 - s_0, ..., s_t - s_0$. This also strengthens our understanding of skills as dynamic behaviours.

By resampling skills with state-norm throughout the episode, we can approximate a much broader starting state distribution $p(s_0)$ than given in the base-MDP. Hopefully, because of this resampling mechanism, the skills learn to move between different *stable states*, i.e. states from which many skills are applicable. Thus, we are not actually trying to learn $p(s_0) = \mathcal{U}(\mathcal{S})$, but this more constrained distribution of stable states. How often we resample a skill during learning should therefore be reflective of how much we expect transition dynamics to vary around different states. If (like in unbounded 2dNav), they do not, we do not need to resample. We can simply learn skills without state-norm and then apply it during skill use.

On the other hand, in a maze the transition dynamics are different near the walls, through which the agent can't move (a step that ends outside a wall is projected to the closest point in the maze). This makes learning while resampling skills throughout the episode difficult, but it highlights the semantic difference between skills learned with and without state-norm. Without it, we assume skills must cover the entire space of possible behaviours, whereas with state-norm we learn skills that should make covering this space easier when using them. If skills are not composable, then we can only solve a given task if we have already learned a specific skill that does so.

To offer a more interesting example, we may consider a simulated humanoid environment. Here a stable state might be standing upright, and we might have skills for taking a step with the left or the right leg, both returning to this stable step. By performing the two alternately, we can form the more temporally abstracted behaviour of walking. We believe that state-norm introduces an inductive bias that could encourage learning of such skills.

5.2.5. Temperature annealing

Although many prior works use entropy-regularised RL to formulate objectives of the form $\mathcal{I}(f(\tau); z) + \alpha\mathcal{H}(a|s, z)$, they usually simply fix the temperature $\alpha = 0.1$. Section 2.4.3 provides a motivation for considering this hyperparameter more closely.

In maximum-entropy RL, entropy-regularisation is used to encourage exploration. In skill

discovery, this entropy prolongs skill learning, because it makes it more difficult to match the sampled trajectories with the corresponding skill variable. This corresponds with a smoothing of the trajectory space. However, we ultimately want to learn low-variance skills, therefore with a low entropy, and more clearly disentangled from one another. This suggests an intermediate approach, in which we begin training with high α to encourage exploration, and end training with low α.

It has been observed that training a β-VAE with $\beta = 1$ is inferior to different weights for the regularisation term (e.g. $\beta = 0.1$ [30]), and that annealing β throughout training generally leads to improved performance. [43] describes a cyclical annealing schedule, increasing and then resetting β several times throughout training (5.4).

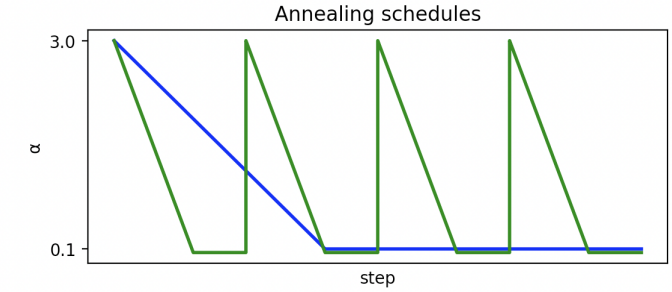


Figure 5.4.: Annealing schedules for the temperature in VOD

The authors of SAC also proposed a version (EC-SAC) of their algorithm in which α is learned, arguing that the policy should be allowed to behave more deterministically in areas of the state space in which it can act confidently (and more randomly in others). Gradients for α are computed with equation 5.4, where $\bar{\mathcal{H}}$ is the target entropy.

$$J(\alpha) = \mathbb{E}_{a\sim\pi}\Big[-\alpha \log \pi(a|s) - \alpha\bar{\mathcal{H}}\Big] \tag{5.4}$$

6. Experiments

In the following, we will evaluate the success of our method in a simple navigation environment, and then in a complex robotics task. In doing so, we hope to answer the following research questions:

1. Does cDIAYN learn discriminable skills?

2. What is the effect of skill length T on skills learned by cDIAYN?

3. Can we learn deep option hierarchies by repeatedly applying cDIAYN?

We use the navigation environments because they allow for simpler inspection of results, to gain a better understanding of the changes proposed in the previous section. We test our method in the robotics environments to evaluate whether cDIAYN can scale to high-dimensional state- and action- spaces.

6.1. Simple navigation environments

We visualise skills spread evenly over the 2-dimensional skill space $[-1, 1]^2$ (figure 6.1).

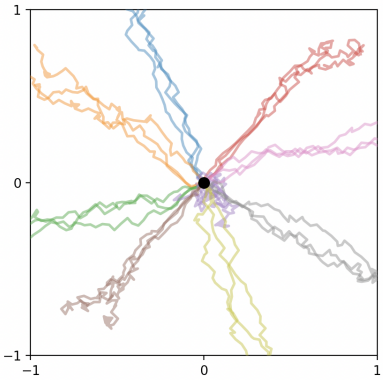


Figure 6.1.: cDIAYN learns diverse skills with a continuous latent space.

We find that cDIAYN learns lower-variance behaviours than the discrete version, leading us to believe that the low variance of skills cited in DADS [5] may be a consequence of the continuous skill space, rather than the modified objective $\mathcal{I}(s';z|s)$. The learned skills cover the reachable portion of the state space well, with each skill ending in a small distribution of states. This means that skills learned by cDIAYN converge to goal-reaching behaviours.

We can also effectively showcase the semantic difference in skills learned with and without state-norm. Figure 6.2 shows cDIAYN when intialised from a state outside the training distribution. Many of the skills could not be executed within T timesteps in this case, making them less useful in our proposed hierarchy.

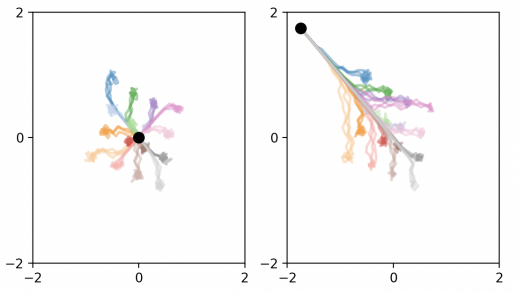


Figure 6.2.: Out of training distribution use of skills in cDIAYN. Here, we learned 8 skills, with each colour representing one one skill.

Moreover, without state-norm there is little value in sequencing skills at all, since the agent can never leave the distribution of states induced by the skill-conditioned policy. The difference is shown in figure 6.3.

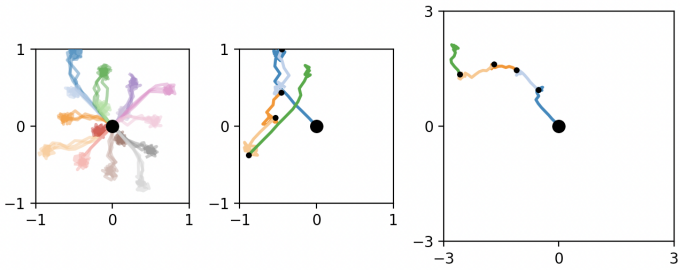
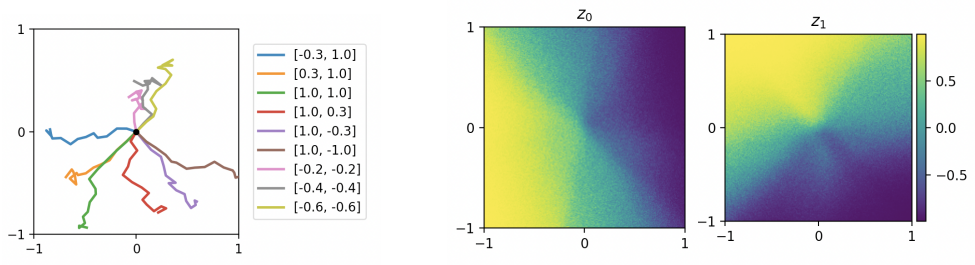


Figure 6.3.: With state-norm we can use skills to reach out-of-training-distribution states. Left: learned skills. Middle: skill use without state-norm. Right: skill use with state-norm.



(a) Skill interpolation

(b) Discriminator visualisation

Figure 6.4.: In (a) we choose different $z \in \mathcal{Z}$ and rollout the corresponding skill. (b) visualises $q_\phi(z|s)$ for $s \in [-1, 1]^2$, $z = (z_0, z_1)$.

6.1.1. Smoothness

It is possible to interpolate between skills in DIAYN. However, much of the interpolating space is meaningless, since we never use it during training. By introducing a continuous skill space, we learn skills with interpolation in mind. We show this in figure 6.4.

Crucially, the skill discriminator learns a smooth skill space, despite the diversity assumption not explicitly being backpropagated through it.

6.1.2. Skill Dimensionality

In cDIAYN, the dimensionality of the skill space determines the size of the bottleneck, replacing the number of skills in DIAYN. We would like to get a better understanding of its importance in simple navigation environments. We compare cDIAYN in 2dNav with $|\mathcal{Z}| \in 1, 2, 3$ and show the results in 6.5. We see that for a one-dimensional skill space, cDIAYN learns a function with only linear variability for $q_\phi(z|s)$, with smooth interpolation along a single axis. We observe the same effect when we test $|\mathcal{Z}| = 2$ in a 3-dimensional point environment, in that skills learned cover states along a 2-dimensional hyperplane in the 3-d space. We view this as a property of the discriminator function we are learning: a d-dimensional latent space can express variability along d dimensions. Thus, although in 2dNav a 2d skill space seems reasonable, by using higher dimensionality we may be able to capture properties of trajectories, rather than states.

Interestingly, in figure 6.5 we attain a qualitatively better solution when we choose a 3-dimensional skill space to encode the 2-d state, in spite of attaining a lower discriminator accuracy and significantly lower intrinsic reward. We think that this is a consequence of high variance in the discriminator, since the diversity assumption is not included in the

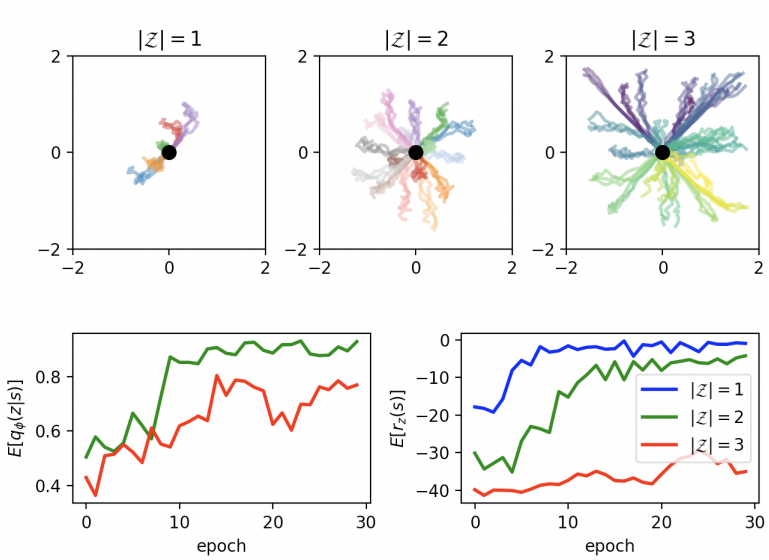


Figure 6.5.: cDIAYN in 2dNav for varying dimensionality of the latent space

discriminator loss. Thus, although $\mathbb{E}[q_\phi(z|s)]$ increases, the intrinsic reward does not, because the discriminator may assign high probability $q_\phi(z|s)$ to many skills z for each s.

6.1.3. Escaping Randomness

Before any training, and therefore during initial experience collection, the skill-conditioned policy is just a random policy. This random policy induces a distribution $p^0_\pi(s) = \mathbb{E}_z[p(s|z)]$ over states $s \in \mathcal{S}$, over which we then perform skill learning with our discriminator model $q_\phi(z|s)$. For maximal generality of learned skills z, we would hope that $p_\pi(s)$ approximates something like a uniform distribution over all states. The authors of DIAYN show that this is in fact the optimal solution for their method, with each skill $\mathbf{z}$ inducing a uniform distribution $p(s|\mathbf{z})$ over its partition of the state space and skills being uniform in $\mathcal{Z}$.

However, the initial distribution $p^0_\pi(s)$ is likely not uniformly distributed over the state space. Therefore, throughout training the state distribution of the skill-conditioned policy must expand. This is what we consider the fundamental value of the diversity assumption: it is beneficial for the policy to reach new states, which it can effectively assign to one skill. However, the starting distribution for skill discovery is always induced by a random policy over the action space.

As we see in figure 6.6, the state space coverage of a random policy in 2dNav degenerates as we increase the length of rollouts. Although in each case, the policy could reach every part of the state space ($\delta * T = 1$), in the case $\delta = 0.01, T = 100$ the policy would have to take 100

steps in the same (any) direction to reach the edge of the state space.

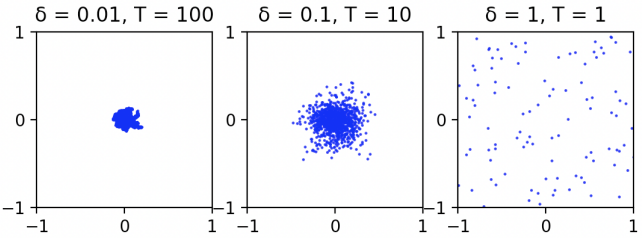


Figure 6.6.: Visualising $p_\pi^0(s)$ for different 2dNav environments. We collect 100 rollouts with step size δ and rollout length T.

This raises the question: how does the initial distribution of states p_π^0 affect the distribution of states p_π covered by the final skill-conditioned policy? In the ideal case, skill discovery learns skills that cover the entirety of the reachable state space, regardless of p_π^0. Again, we can test this in 2dnav, by performing skill discovery with varying step-size and skill-length. To perform a fair comparison, we keep all parameters constant between cases (including the number of skill and policy model train steps), only scaling the total number of environment steps taken and the size of the replay buffer proportionally with the skill length. Figure 6.7 shows that skill discovery converges to a good state-coverage much more quickly for shorter trajectories. State-coverage increases throughout training as skills diversify, but for larger T skills must diversify further, from a more restrictive $p_\pi^0(s)$.

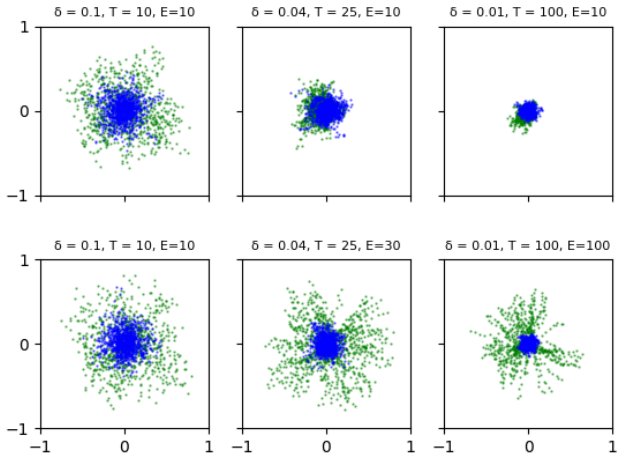


Figure 6.7.: Comparing $p_\pi(s)$ after E epochs (green) and $p_\pi^0(s)$ (blue) for different 2dNav environments.

This provides a justification for our integration of hierarchy into VOD. We can use learned skills to improve exploration in the beginning of skill discovery, and thus more quickly learn

skills for larger T.

6.1.4. Goal-attaining Behaviours

We also see an undesirable training dynamic, both in the continuous and discrete versions of DIAYN. When we let training continue for longer skills begin collapsing to a smaller distribution of states around s_0, as visualised in figure 6.8. p_π expands initially because skills diversify, i.e. the agent reaches new states which are more easily discriminable. However, eventually the agent learns to discriminate states closer to s_0 and reaches them more quickly, making them more desirable. We think this is a consequence of choosing an objective that encodes states, rather than trajectories. Every skill ultimately becomes strongly associated with one small portion of the state space, rather than all of the states in the trajectory it was supposed to encode. Skills become goal-reaching behaviours, and skill discovery becomes a mechanism for goal-setting and learning the corresponding reward-function for GCRL. There is nothing inherently wrong with this (we can reasonably define a skill as reaching some goal within T time-steps), but it deviates from our desired interpretation of skills. Moreover, many of the T time-steps may be devoted to maintaining the achieved goal state, further removing skills from dynamic behaviours. It also means that the problem difficulty is not directly tied to the skill length, possibly diminishing the benefit of the hierarchy.

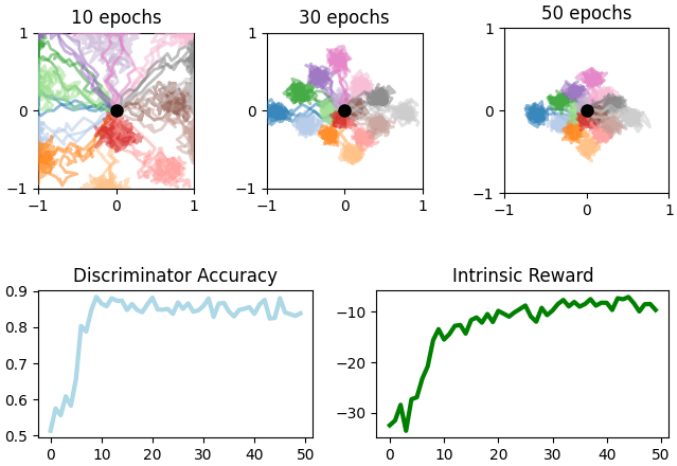


Figure 6.8.: $p_\pi(s)$ collapses to states near s_0 if we let training carry on for too long. We see that learned skills correspond to simple goal-reaching behaviours. Although skills are learned with $\delta = 0.1, T = 20$, we set $T = 100$ here to emphasise the described effect.

The hierarchy may help to counteract this, in that we can stop skill discovery once discriminator accuracy stops increasing, and then learn the next level of the hierarchy, benefiting from

improved exploration by using learned skills.

6.1.5. α-annealing

We find that the entropy-regularisation term has a significant impact on the skills ultimately learned by the algorithms. This is clearer in the continuous version of DIAYN, because it eliminates the choice of the number of skills, which fulfils a similar role. Choosing small alpha impedes learning, because the policy takes a long time to achieve a good coverage of the state space (figure 6.10), and can only converge when this coverage is achieved. For large α, the algorithm converges more quickly, but again covering a smaller area around s_0 (figure 6.9).

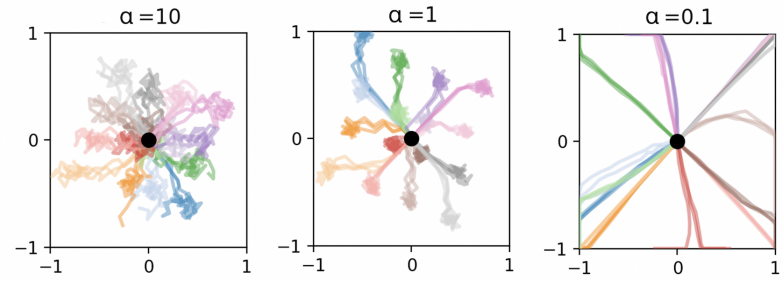


Figure 6.9.: cDIAYN after 100 epochs for different fixed values of α

localoptima.png

Figure 6.10.: Training with low temperature can get stuck in local optima. Comparing $\alpha = 0.1$ (top), and $\alpha = 1.0$ (bottom) after 10, 30, and 60 epochs.

We find that annealing α leads to more stable training and better state space coverage, primarily because it helps to avoid the local optima in figure 6.10. The cyclical annealing schedule did not lead to noticeably different results, and we were unable to learn with EC-SAC, because α diverges and the policy regresses to a random policy, though we are not sure why.

6.1.6. Hierarchy

We now decompose the problem of learning skills of length 100 in 2dNav into learning a hierarchy with $T = (10, 10)$, and compare the results in figure 6.11.

We keep the number of environment steps per epoch constant (at 1000 steps), therefore

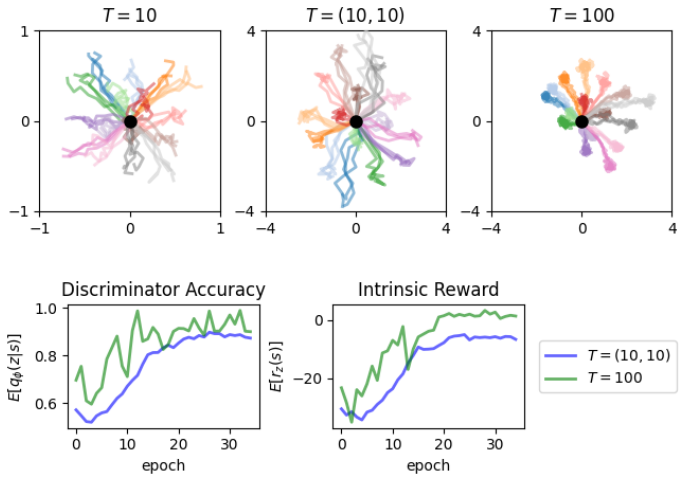


Figure 6.11.: Skills learned in 2dNav with hierarchical and flat skill discovery.

the skill model and policy of the second layer ($T = (10, 10)$) see only every 10th environment step. Both the layered and the flat hierarchy are trained for the same total number of epochs, however skills learned by the hierarchical policy cover a slightly larger portion of the state space. We find that as we reduce T the attain-target behaviour becomes less pronounced, suggesting that skills can not diversify merely based on the final state. This makes it more difficult for the agent to assign a small distribution of states to a skill, which may be the reason for the lower accuracy and reward.

We can also learn a 3-layer hierarchy with $T = (5, 5, 5)$, finding that this results in significantly better coverage of the state space (figure 6.12). Interestingly, the layers using skills rather than primitive actions converge more quickly, in spite of the stochasticity of skills. This may be because the learned skills favour states further away from s_0, whereas primitive actions are uniform in $[-0.1, 0.1]$.

Each subsequent layer takes 5 times as many steps per epoch in the environment. This exposes a potential weakness of the method, or perhaps just a property of learning in a hierarchy. As the number of time-steps in the environment between policy decisions increases (for higher layers), the training data for this layer becomes increasingly sparse relative to the actual amount of environment interaction. We could pass the intermediate steps as well, reasonably so with the state-based DIAYN objective. We choose not to do this here, since it weakens our desired understanding of the skill hierarchy. We suspect that the benefit this might have would be reduced when choosing a sequence-based objective.

We observe a fundamental tradeoff in optimising the diversity assumption. The longer the trajectories are that we are encoding, the easier it is to learn discriminable skills (measured by q_ϕ and r_z), because the trajectories we sample from the environment are increasingly sparse

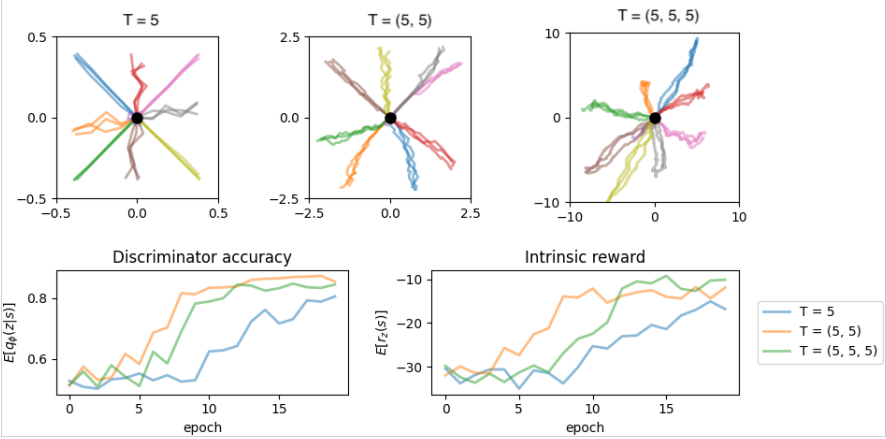


Figure 6.12.: A 3-layer skill hierarchy.

in the space of trajectories of length T. However, skills derive their meaning in diversifying from one another: if this is too easy, we attain a weaker solution. We see that discriminator accuracy and intrinsic reward are not perfect proxies for what constitutes successful skills.

6.2. Robotics Environments

We now examine whether cDIAYN learns discernible and composable skills in more complex robotics environments. Unfortunately, it is difficult to represent trajectories through high-dimensional space on paper. In previous works, the authors validated their methods in various contexts, like task success with a given external reward or imitation learning. We believe that a broad test suite of this kind is the only way to effectively test the learned skills, because individual skills need not necessarily be interpretable to a human to be useful later. Unfortunately we did not have time to conduct such experiments, nor to tune the many different hyperaparameters. Therefore we use this section merely to demonstrate that our method scales to more complex environments.

6.2.1. Fetch

In Fetch environments, we learn skills that move the robotic arm to different gripper positions. cDIAYN converges very quickly (about 100 epochs, comparable with 2dNav) with relatively short $T = 20$, covering the range of possible gripper positions (6.13). As a result, there is little

to be gained from introducing the hierarchy, and we find the two learn the same simple set behaviours. This also showcases a problem with learning skills in the complete absence of task rewards, as much of the state space covered by the arm can never lead to any useful behaviours, and a more significant source of novelty (the object) is completely ignored.

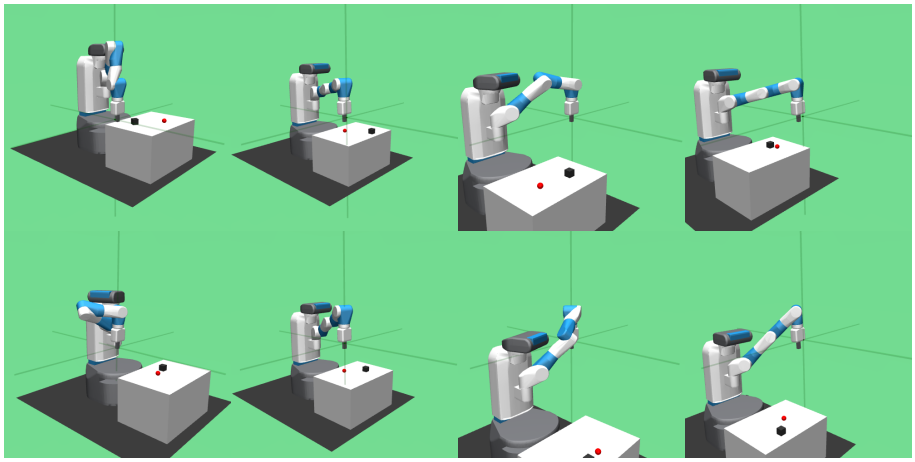


Figure 6.13.: Skills learned by cDIAYN in the FetchPush environment.

Figure 6.14 can interpolate along a single axis to see a smooth change in the final position of the gripper. In environments with objects, the skills largely ignore the objects, likely because these interactions are relatively sparse in the space of possible behaviours. There are skills that open and/or close the gripper in some consistent way, but not specifically to pick up the object.

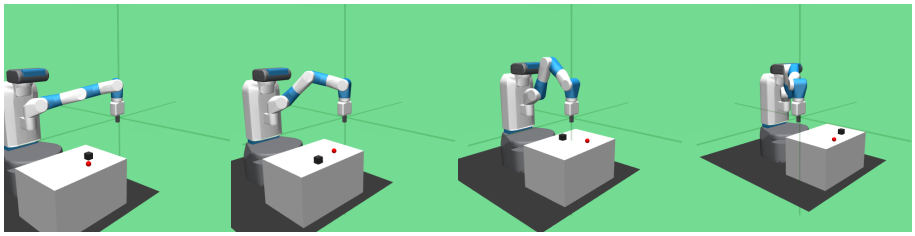


Figure 6.14.: Skill interpolation along a single axis in Fetch.

6.2.2. Hand

We experiment with 3- and 4-dimensional skill spaces, finding that if we increase the dimensionality further, the skills learned by cDIAYN become less diverse. In environments with objects, no interesting manipulation behaviours were learned. Figure 6.15 shows the end-state of 16 skills sampled from $p(z)$ for an average run.

In figure 6.16 we compare single-layer cDIAYN learning for different sequence lengths.

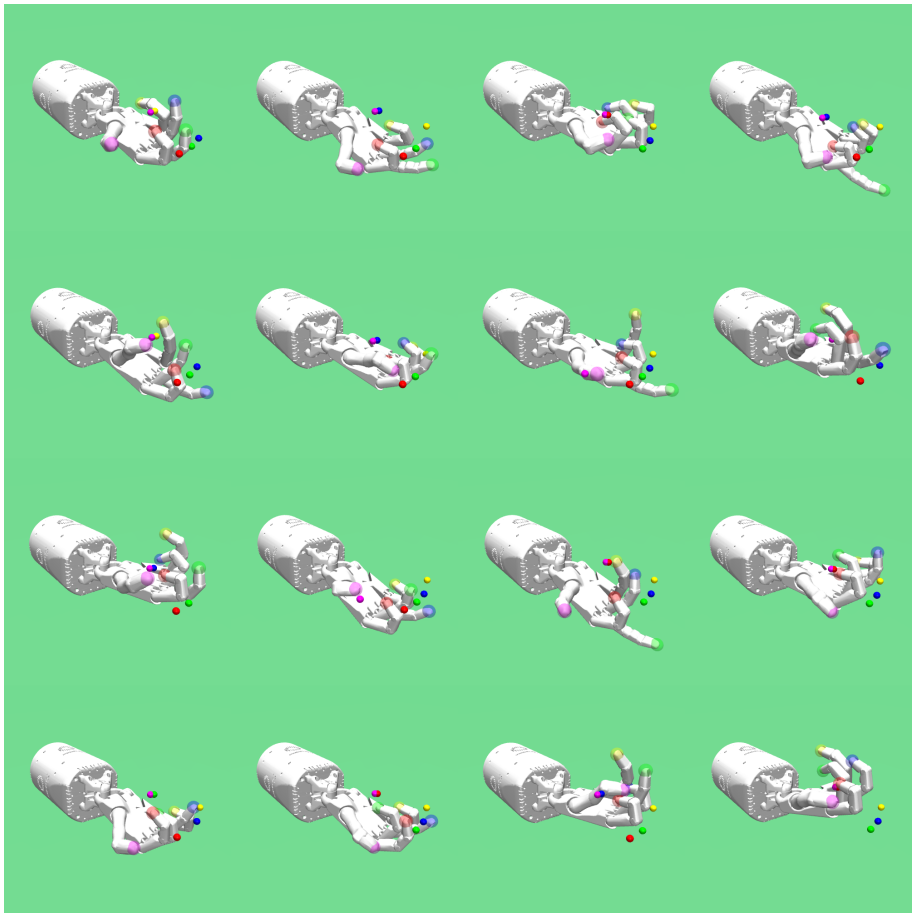


Figure 6.15.: Skills learned by cDIAYN in the HandReach environment.

Again, we see that we achieve higher discriminator accuracy and intrinsic reward for longer sequence lengths, but without attaining qualitatively better skills. we find the number of skills sampled per epoch to be an important hyperparameter. This makes sense: we want to make sure that every model update includes diverse $z \in \mathcal{Z}$. For longer skills, this means longer epochs, and therefore fewer model updates per environment step. Not increasing it leads to unstable training. This may be because the marginalisation in the intrinsic reward requires exponentially more samples for every additional latent dimension. Although we do learn distinct behaviours, it is clear that the learned skills cover only a small subset of the space of possible behaviours.

Figure 6.17 shows that we can again loosely interpolate behaviours, suggesting that the skill space learned by cDIAYN is reasonably smooth. It also showcases how the skills learned by VOD differ from our intuitions. We would like to learn skills that isolate parts of the state space and action space, like bending a single finger. In Fetch we would like to see a skill that corresponds with opening/closing the gripper. However, we don't think the chosen

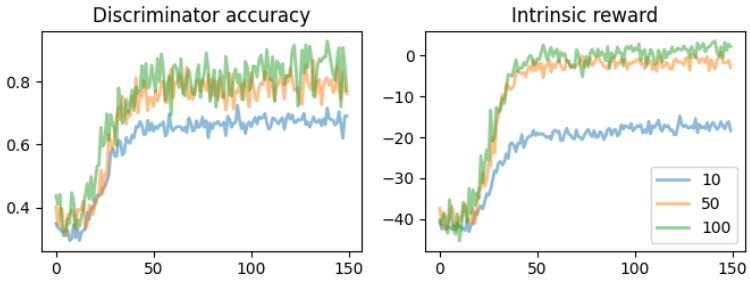


Figure 6.16.: Comparison of convergence behaviour of cDIAYN for $T \in 10, 50, 100$.

objective prioritises such behaviours. Rather, we learn behaviours that move every joint at every time-step, in some repeatable way (in that a skill z initiated from some state s will consistently produce the same behaviour). This may be why prior works [4, 5] restricted the observations seen by the discriminator to only the (x, y) coordinates or the center of mass of the agent, creating spaces in which diversity matches our understanding.

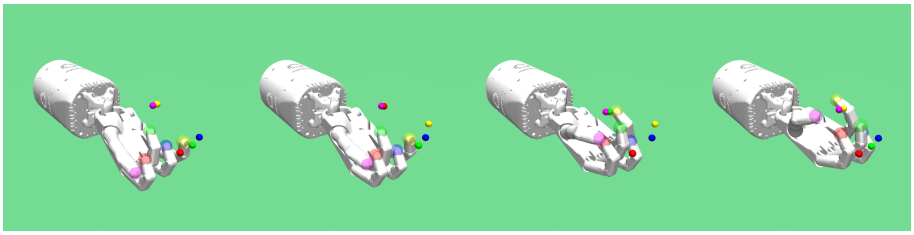


Figure 6.17.: Skill interpolation in Hand.

Learning is possible when resampling skills throughout the episode, but against our expectations we find this to be the case both with and without state-norm. However, the behaviours learned by the two are semantically different. Again, we can learn a second layer of skills on top of the first, however the learned skills do not appear qualitatively any better than those learned by the corresponding flat approach, and overall take longer to learn because of the sparsity of training data at the second level. We were not able to learn 3 layers in a stable manner, but we think this is likely possible with enough training time.

7. Discussion

We now perform a more holistic review of the results of our experiments in chapter 6. Overall, we showed that it is possible to learn a multi-layer hierarchy of skills in an unsupervised manner, although we could demonstrate only limited usefulness. In this section, we will discuss the applicability, the advantages and the weaknesses of our approach.

7.1. Applicability

Because cDIAYN operates in the reward-free MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, p(s_0) \rangle$, human domain expertise is required only to set the hyperparameters. However, we would expect a successful configuration to work in different environments, because few assumptions are made before training. Although the VOD family of approaches is applicable to any MDP, cDIAYN and its hierarchical variant can only be applied in environments with continuous state- and action-spaces.

7.2. Advantages

The primary goal of this thesis was to learn a hierarchy of skills in an unsupervised manner. As a result, the diversity assumption underlying skill discovery is fulfilled at different scales, and we are able to encode the inductive bias that skills are composed of other skills, making them more dynamic.

Skill discovery methods explore the environment by diversifying. If this is too simple, the skill variable z can offer only a limited guiding signal for the policy $\pi_\theta(a|s,z)$. In discrete DIAYN, $\pi(\cdot|\cdot,z)$ approximates a uniform distribution over its partition of the state-space. If the partition corresponding to a skill is too large, the policy continues to behave randomly. The same intuition extends to cDIAYN. This is why, although our experiments showed worse results on our chosen metrics (discriminator accuracy and intrinsic reward achieved), we think that skill discovery can be applied more effectively for shorter skills.

Of course the goal is to learn more temporally extended behaviours, and this is enabled by the hierarchy. To this end, we proposed a simple normalisation mechanism for skill discovery, which allows us to learn skills that are applicable outside the distribution of starting states $p(s_0)$, and can be applied to any skill discovery method to learn a semantically different kind of skills. Resampling skills throughout the episode also provides a basis for *continual learning*, since we have removed the need for environment resets. Most of our experiments were conducted this way.

We also showed that the marginalisation over the skill space $\mathcal{Z}$ to compute the intrinsic reward in DADS implements the diversity assumption, in that it is applicable in cDIAYN with similar results but this alternative justification. This reflects the intuition that we can marginalise over all skills for finite $\mathcal{Z}$ but must sample for continuous skill spaces. This suggests that other prior works (e.g. [3, 6]) which use a discrete skill prior can be adapted in the same way.

Similarly, we introduced annealing schedules for α weighting the entropy-regularisation term, inspired by the success of similar approaches in the training of VAE.

7.3. Weaknesses

Although we showed that learning a hierarchy of skills is possible in principle, we were unable to show much benefit from it. This might be a matter of finding the right hyperparameters, but it might also be a problem with the underlying skill discovery method and idea. Overall, the broadness of the skill discovery problem statement makes evaluating success quite difficult, and as such we were forced to rely on qualitative judgements more than we would like. For the same reason, a lot of hyperparameters were kept at some (we think) reasonable default value, without our having much of an intuition for their effect. More testing in a broader range of environments would be required to really assess the merits of our approach, together with more clearly defined goals for skill discovery. We discuss some ideas for this in section 8.1.3.

In retrospect, a VOD method that works on a discrete action space would have allowed us to better evaluate the intuition underlying this thesis, described in section 4.1. However, we wanted to maintain comparability with prior works in VOD, which were applied mostly in continuous environments. We don't expect the intrinsic reward used here (and in DADS) to scale to higher-dimensional latent spaces, again because samples of the skill variable z for marginalisation become increasingly sparse in $\mathcal{Z}$.

It is also clear that the simple layerwise training procedure described in section 5.1 leads to a very rigid hierarchy. It makes the unreasonable assumption that we can completely cover

the space of possible behaviours in the first layer. For example, if we learn skills for the robotic hand without access to the block or pen, and then introduce one of these objects, we can only learn very temporally extended skills to manipulate them. If we wanted to learn high-frequency (low-layer) behaviours to manipulate the object, we would have to retrain the lower layer, which in turn would mean that we have to retrain every subsequent layer. This may be possible, but it is certainly undesirable. While the increase in temporal abstraction as we move up the hierarchy seems reasonable (and is fundamental to this thesis), it necessitates top-down feedback, i.e. the ability of higher layers to modify the behaviour of lower layers. Barring this, it is likely that we cripple the agent with every additional layer, because we may not be able to perfectly solve the skill discovery problem at a given layer before training the next.

We also think that the purely state-based objective is misplaced in skill discovery because skills are only loosely related to trajectories. It encourages the attain-target-state behaviours described in section 6.1.4. We think encoding trajectories might ease learning for shorter skill-lengths. However, for shorter trajectories, the information loss induced by state-norm is also stronger. We think this can be circumvented by allowing the policy to see the full state s rather than the normalised one $\bar{s}$, and only using the latter in training the discriminator. Moreover, although we use an off-policy RL algorithm, we use only on-policy data.

We also don't backpropagate the diversity assumption through the skill-mapping, meaning that it can learn a function that assigns high probability $q_\phi(z|s)$ for diverse z without immediately being penalised for it. The iterative nature of skill discovery algorithms compensates for this, but it can still lead to an inconsistent reward function for the RL-agent.

Overall, our experiments in the robotics domain call into question the usefulness of completely unsupervised skill discovery. In the Fetch environment, most of the learned skills have no potential to be useful in any way later. We had hoped that the hierarchy would encourage more abstract behaviours by manipulating the object, but this was not the case.

8. Conclusion

In this thesis, we present cDIAYN, a method for learning composable skills that is recursively applicable to the same environment, allowing us to learn a hierarchy of skills in an unsupervised manner. We successfully replaced the categorical skill prior in DIAYN with a continuous one, and introduce α-annealing as already used in the training of VAE. We analyse this method in a simple navigation environment, and provide preliminary results in high-dimensional state- and action-spaces suggesting its limitations. Moreover, we provide a simple implementation based on the recent TF-Agents API. Overall, we hope that this thesis provides a solid foundation for further work in what is an exciting research topic. We summarise some possible directions for future work in the following, organised by the aspect of the algorithm they address.

8.1. Future Work

8.1.1. Skill Discovery

In this thesis, we described the family of skill discovery approaches we refer to as Variational Option Discovery algorithms, and provided a single instantiation that fulfils the criteria that make it usable in the proposed hierarchy. However, we can imagine many such instantiations (like the previously mentioned discrete-discrete mapping). DADS would be immediately applicable within our skill hierarchy. We outline some of the possible sources of variability within VOD, and give directions for further research.

Objective

Perhaps the most obvious parameter to investigate further is the information-theoretic objective $\mathcal{I}(f(\tau); z)$, more specifically the preprocessing function f. This implies different model architectures of the skill mapping and rl-agent (although the latter may not be necessary [44]). In particular, the use of recurrent models that don't decompose a trajectory into constituent time-steps would align more closely with the intuition presented in section 4.1, and we think

the hierarchy can have a clearer impact here. State Marginal Matching [39] extends the skill discovery objective with terms that explicitly encourage improving state coverage $p_\pi(s)$, and approximating some given distribution over states $p^*(s)$.

We can introduce more complicated functions for f, for example ones that hide part of the state. We could use this to formulate an agent *curriculum* at a very holistic level. For example, in the hand environment we could learn skills for each finger individually, communicating to the agent that each finger represents a separate functional component. After learning skills that manipulate the hand, we could learn skills that manipulate only the block. A similar idea was put forward in a recent paper [**MUSIC**], in which the authors define an MI-based intrinsic reward $\mathcal{I}(\mathcal{S}_a, \mathcal{S}_s)$ by which the agent (whose state is given by $\mathcal{S}_a$) learns to manipulate its surroundings $\mathcal{S}_s$. Although we do not immediately see a way to compose different objectives in a single layer (aside from simply adding them up), this is easily done between different layers.

It would also be interesting to study how $|f(\tau)|$ affects learning. f_DIAYN generates lots of training data, because a (τ, z) is turned into (s_i, z) pairs, whereas for $f(\tau) = \tau$ we generate a single sample. Since neural networks require large amounts of training data to learn, perhaps the apparent success of DIAYN in comparison with similar prior works was to some extent because of this.

Reusing off-policy samples

[36] shows how we can extend our method to reuse off-policy samples, which has the potential to significantly improve sample efficiency. A simple way to do this would be to train the skill-mapping in an on-policy manner, while training the rl-agent off-policy. This is possible by relabelling old experience (s, a, s', z, r_{old}) using the reward-function r_z defined by the up-to-date skill-mapping. In principle, this does not even require an importance sampling ratio, since transitions (s, a, s', z) that are unlikely under the current skill-mapping simply receive a low reward, and prior works have found limited usefulness of such ratios [**revistingER**]. Building on this, we could perform much more aggressive data augmentation, by relabelling transitions (s, a, s') with skill variables z that were not used to generate the transition. This can be thought of as a kind of Hindsight Experience Replay [45], a technique that has displayed significant benefits in sparse-reward setting. In the skill discovery context, this reflects the intuition that we use the intrinsic reward to *embed* the skill-mapping in the environment via RL. In particular, this could help to counteract the increasingly sparse training data received from the environment by higher layers. [46] provides motivation for exploring data augmentation in RL further.

In a similar vein, the authors of HIDIO build a meta-MDP, in which states store the skill-trajectory leading up to them. This allows them to test multiple objectives against

each other simultaneously. A similar construction would be interesting for skill discovery, although we would have to collect on-policy data corresponding to each objective to update the discriminator.

Diversity assumption

We use a very crude instantiation of the diversity assumption in this thesis. To ensure that $q_\phi(z|s)$ is different from $q_\phi(z'|s)$ for $z \neq z'$, we estimate $\mathbb{E}_{\mathcal{Z}}[q_\phi(z'|s)]$ and assign high reward when $q_\phi(z|s)$ is larger than this expected value, and low reward when it is lower. Perhaps there is a way to encode a stronger notion of smoothness here, for instance regularising with a KL divergence term. Intuitively, we would like to encode the bias that parts of the skill-space $\mathcal{Z}$ that are "further apart" are also more different. Although it appears as though this behaviour is achieved in cDIAYN (because of the smooth skill space), we think that this could be encoded more effectively.

Building on the VAE analogy

Prior works have shown that with a sufficiently engineered feature space, RL may be overkill to optimise an external reward [47]. Particularly in skill discovery, the reasons for using reinforcement learning become somewhat blurred. Since we control the skill length, we can ensure that long-term credit assignment is not necessary, and the skill-mapping produces a dense reward signal. It would be interesting to compare the performance of SAC with that of a simple feed-forward neural network or linear controller for the policy $\pi_\theta(a|s,z)$, in the context of skill discovery. It would be straightforward to introduce entropy-regularisation, so we should not expect this to hinder exploration, reducing the role of RL to performing reward backup.

Self-consistent Trajectory Autoencoders [48] build directly on the notion of skill discovery as trajectory compression. EDL (see section 3.1.1) offers another alternative approach to skill discovery, that more directly builds on VAE. It would be interesting to compare the performance of EDL in our hierarchical framework.

Leveraging Meta-RL

Meta-learning and skill discovery may be viewed as complementary to one another: skill discovery learns to propose and solve tasks within the defined distribution $p(z)$, whereas meta-learning provides a framework for learning to learn within a given task distribution. This synergy is taken advantage of in [49]. Broadly, the task of Meta-RL is to adapt quickly

to any task within a given distribution, using experience from samples of this distribution. In the authors words, their method "can be thought of as automatically acquiring an environment-specific learning procedure". The authors use DIAYN in their method, but any task distribution learning method (like VOD) can take this role.

8.1.2. Hierarchy

Training the hierarchy in a strictly layerwise fashion leads to a very rigid structure, and arguably wastes a lot of environment interaction that could be used to improve the model. We could incorporate top-down feedback by continuing to train layers $0, ..., i-1$ while training layer i, since we continue to sample skills from these lower layers. In this way, how the skills are actually used can shape them, and higher layers at least theoretically preserve the ability to enact any behaviour. This is a very simple modification that we think has promise, at the very least to improve sample efficiency. In this setup, although no actual rewards are passed down from higher layers, the lower layers continue to learn in conjunction with higher layers.

On the other hand, prior works like [40, 8, 24] have showed how we could learn multiple layers of the hierarchy simultaneously, as discussed in 3. Particularly SAC-LSP provides interesting insights for ensuring that suboptimal lower layers don't cripple the agent, as we observed our layered approach is likely to do. The authors of HIDIO similarly posit that joint training of both layers in their worker-scheduler hierarchy was necessary for task success.

An important topic that was unfortunately completely omitted from this thesis, is how to actually use the hierarchy. For instance, we would expect that an external task requires the agent to act at different levels of temporal abstraction, perhaps increasing the granularity of behaviour when longer behaviours lead to insufficient task progress. As an example, consider the Fetch environment: moving the gripper to the object is a more temporally extended behaviour that requires lesser precision, but actually picking up the object when the gripper is close enough may require more precision.

8.1.3. Algorithm Evaluation

Reproducibility and comparability are fundamental issues in any research area. Because of their broad goal, skill discovery methods are inherently difficult to evaluate. We demonstrated in this thesis that the two fundamental metrics (discriminator accuracy and intrinsic reward achieved) do not perfectly capture our understanding of successful skills. We could come up with other metrics, like some measure of the state space coverage, but we think the most effective way to test how beneficial learned skills can be, is to use them. The changes made in this thesis (in particular reducing the skill length) arguably make the learned skills even

more difficult to inspect against human intuitions of skills, particularly as we increase the dimensionality of the latent space.

Ideally, we could place an agent in a rich environment, let it learn independently for a while, and then present it with a set of tasks across which we measure the benefit of unsupervised interaction with the environment, similar to meta-testing in Meta-RL.

Another open question is how closely we can expect our intuitions for diverse skills to align with those learned during skill discovery. For instance, in the hand environment we would like to (for example) see skills manipulating individual fingers. Primarily, we would like to see skills that control different isolatable functional components. Is this part of an optimal solution, and our agent is simply not achieving it? Or should we not even expect to see this in an optimal solution? In [6], the authors give a pessimistic view of this.

A. General Addenda

A.1. Implementation

Our implementation[1] based on the Tensorflow Agents API. The implementation is divided broadly into rollout-driver, skill-model and policy learner, reflecting the three stages in each iteration of VOD. The skill discovery base class manages the data flow between them and implements the training loop. The code was run on Ubuntu 20.04 with Nvidia GeForce 2080 GPU, as well as Mac with M1 chip.

The skill model is a simple neural network with two hidden layers and a probabilistic output layer. The output layer is a mixture of independent gaussians, where we usually fix the variance to 0.1, and then squash this distribution to the range $[-1, 1]$. For every experiment we use the same network architecture between discriminator and SAC models, with two hidden layers of size varying between 128 and 300. We keep the number of samples from the skill prior constant across experiments, although this is in all likelihood a parameter you need to tune. Our initial hyperparameter configuration follows that used in DADS. The robotics environments require the recently open-sourced MuJoCo physics engine.

A.2. Hyperparameters

We provide a default hyperparameter configuration for hcDIAYN, training a 2-layer skill hierarchy.

[1] https://github.com/maxf98/thesis

Hyperparameter	Value
Number of layers K	2
Skill lengths T	(10, 10)
Skill prior $p(z)$	$\mathcal{U}([-1,1]^d)$
Skill dimensionality d	2
Number of prior samples for r_z	400
Replay buffer size	1000
Episode length H	1000
Apply state-normalisation	True
Discriminator fully connected layer params	(128, 128)
Fix discriminator variance	True
Initial α	3.0
Target α	0.1
α anneal steps	4000
α anneal period	None
Collect steps per epoch	1000
Train batch size	128
Discriminator train steps	32
SAC train steps	128

Table A.1.: General hyperparameters

Hyperparameter	Value
Fully connected layer params	(128, 128)
Optimizer	Adam [**Adam**]
Learning rate	$3 * 10^{-4}$
Alpha loss weight	0
Target update tau	0.005
Target update period	1
Discount factor γ	1.0

Table A.2.: SAC hyperparameters

List of Figures

List of Tables

Bibliography

[1] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[2] A. G. Barto and S. Mahadevan. "Recent advances in hierarchical reinforcement learning". In: *Discrete event dynamic systems* 13.1 (2003), pp. 41–77.

[3] K. Gregor, D. J. Rezende, and D. Wierstra. "Variational intrinsic control". In: *arXiv preprint arXiv:1611.07507* (2016).

[4] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. "Diversity is all you need: Learning skills without a reward function". In: *arXiv preprint arXiv:1802.06070* (2018).

[5] A. Sharma, S. Gu, S. Levine, V. Kumar, and K. Hausman. "Dynamics-aware unsupervised discovery of skills". In: *arXiv preprint arXiv:1907.01657* (2019).

[6] J. Achiam, H. Edwards, D. Amodei, and P. Abbeel. "Variational option discovery algorithms". In: *arXiv preprint arXiv:1807.10299* (2018).

[7] C. Florensa, Y. Duan, and P. Abbeel. "Stochastic neural networks for hierarchical reinforcement learning". In: *arXiv preprint arXiv:1704.03012* (2017).

[8] J. Zhang, H. Yu, and W. Xu. "Hierarchical Reinforcement Learning By Discovering Intrinsic Options". In: *arXiv preprint arXiv:2101.06521* (2021).

[9] V. R. Konda and J. N. Tsitsiklis. "Actor-critic algorithms". In: *Advances in neural information processing systems*. 2000, pp. 1008–1014.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[11] M. E. Taylor and P. Stone. "Transfer learning for reinforcement learning domains: A survey." In: *Journal of Machine Learning Research* 10.7 (2009).

[12] T. Schaul, D. Horgan, K. Gregor, and D. Silver. "Universal value function approximators". In: *International conference on machine learning*. PMLR. 2015, pp. 1312–1320.

[13] A. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine. "Visual reinforcement learning with imagined goals". In: *arXiv preprint arXiv:1807.04742* (2018).

[14] D. Ghosh, A. Gupta, and S. Levine. "Learning actionable representations with goal-conditioned policies". In: *arXiv preprint arXiv:1811.07819* (2018).

[15] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. "Vime: Variational information maximizing exploration". In: *arXiv preprint arXiv:1605.09674* (2016).

[16] S. Mohamed and D. J. Rezende. "Variational information maximisation for intrinsically motivated reinforcement learning". In: *arXiv preprint arXiv:1509.08731* (2015).

[17] J. Achiam and S. Sastry. "Surprise-based intrinsic motivation for deep reinforcement learning". In: *arXiv preprint arXiv:1703.01732* (2017).

[18] A. Aubret, L. Matignon, and S. Hassas. "A survey on intrinsic motivation in reinforcement learning". In: *arXiv preprint arXiv:1908.06976* (2019).

[19] A. S. Polydoros and L. Nalpantidis. "Survey of model-based reinforcement learning: Applications on robotics". In: *Journal of Intelligent & Robotic Systems* 86.2 (2017), pp. 153–173.

[20] R. S. Sutton, D. Precup, and S. Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial intelligence* 112.1-2 (1999), pp. 181–211.

[21] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum. "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation". In: *Advances in neural information processing systems* 29 (2016), pp. 3675–3683.

[22] O. Nachum, S. Gu, H. Lee, and S. Levine. "Data-efficient hierarchical reinforcement learning". In: *arXiv preprint arXiv:1805.08296* (2018).

[23] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. "Feudal networks for hierarchical reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 3540–3549.

[24] A. Levy, R. Platt, and K. Saenko. "Hierarchical actor-critic". In: *arXiv preprint arXiv:1712.00948* 12 (2017).

[25] C. Chuck, S. Chockchowwat, and S. Niekum. "Hypothesis-Driven Skill Discovery for Hierarchical Deep Reinforcement Learning". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 5572–5579.

[26] C. E. Shannon. "A mathematical theory of communication". In: *The Bell system technical journal* 27.3 (1948), pp. 379–423.

[27] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. "Soft actor-critic algorithms and applications". In: *arXiv preprint arXiv:1812.05905* (2018).

[28] C. Marsh. "Introduction to continuous entropy". In: *Department of Computer Science, Princeton University* (2013).

[29] D. P. Kingma and M. Welling. "An introduction to variational autoencoders". In: *arXiv preprint arXiv:1906.02691* (2019).

[30] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. "beta-vae: Learning basic visual concepts with a constrained variational framework". In: (2016).

[31] C. Colas, T. Karch, O. Sigaud, and P.-Y. Oudeyer. "Intrinsically motivated goal-conditioned reinforcement learning: a short survey". In: *arXiv preprint arXiv:2012.09830* (2020).

[32] A. C. Li, C. Florensa, I. Clavera, and P. Abbeel. "Sub-policy adaptation for hierarchical reinforcement learning". In: *arXiv preprint arXiv:1906.05862* (2019).

[33] V. Campos, A. Trott, C. Xiong, R. Socher, X. Giró-i-Nieto, and J. Torres. "Explore, discover and learn: Unsupervised discovery of state-covering skills". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 1317–1327.

[34] D. Warde-Farley, T. Van de Wiele, T. Kulkarni, C. Ionescu, S. Hansen, and V. Mnih. "Unsupervised control through non-parametric discriminative rewards". In: *arXiv preprint arXiv:1811.11359* (2018).

[35] V. H. Pong, M. Dalal, S. Lin, A. Nair, S. Bahl, and S. Levine. "Skew-fit: State-covering self-supervised reinforcement learning". In: *arXiv preprint arXiv:1903.03698* (2019).

[36] A. Sharma, M. Ahn, S. Levine, V. Kumar, K. Hausman, and S. Gu. "Emergent real-world robotic skills via unsupervised off-policy reinforcement learning". In: *arXiv preprint arXiv:2004.12974* (2020).

[37] W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, and W. Dabney. "Revisiting fundamentals of experience replay". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 3061–3071.

[38] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[39] L. Lee, B. Eysenbach, E. Parisotto, E. Xing, S. Levine, and R. Salakhutdinov. "Efficient exploration via state marginal matching". In: *arXiv preprint arXiv:1906.05274* (2019).

[40] T. Haarnoja, K. Hartikainen, P. Abbeel, and S. Levine. "Latent space policies for hierarchical reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1851–1860.

[41] Y. Song, J. Wang, T. Lukasiewicz, Z. Xu, and M. Xu. "Diversity-driven extensible hierarchical reinforcement learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 33. 01. 2019, pp. 4992–4999.

[42] O. Nachum, H. Tang, X. Lu, S. Gu, H. Lee, and S. Levine. "Why does hierarchy (sometimes) work so well in reinforcement learning?" In: *arXiv preprint arXiv:1909.10618* (2019).

[43] H. Fu, C. Li, X. Liu, J. Gao, A. Celikyilmaz, and L. Carin. "Cyclical annealing schedule: A simple approach to mitigating kl vanishing". In: *arXiv preprint arXiv:1903.10145* (2019).

[44] A. X. Lee, A. Nagabandi, P. Abbeel, and S. Levine. "Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model". In: *arXiv preprint arXiv:1907.00953* (2019).

[45] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba. "Hindsight experience replay". In: *arXiv preprint arXiv:1707.01495* (2017).

[46] I. Kostrikov, D. Yarats, and R. Fergus. "Image augmentation is all you need: Regularizing deep reinforcement learning from pixels". In: *arXiv preprint arXiv:2004.13649* (2020).

[47] D. Ha and J. Schmidhuber. "World models". In: *arXiv preprint arXiv:1803.10122* (2018).

[48] J. Co-Reyes, Y. Liu, A. Gupta, B. Eysenbach, P. Abbeel, and S. Levine. "Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1009–1018.

[49] A. Gupta, B. Eysenbach, C. Finn, and S. Levine. "Unsupervised meta-learning for reinforcement learning". In: *arXiv preprint arXiv:1806.04640* (2018).