Hierarchical Skill Discovery via Reinforcement Learning

Bachelor Thesis in Computer Science

Maximilian Fest

Chair for Robotics and Artificial Intelligence
Technical University of Munich

Abstract

This is an abstract.

Contents

1 Introduction

inductive biases, machines vs. humans, no free lunch theorem, ... -fundamental challenges

- reward shaping
- exploration
- generalisation
- continual learning

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. As such, all interaction between agent and environment reduces to the following simple loop (visualisation). Hence, the RL problem is: 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:

$$\pi^* = arg\max_{\pi} \mathbb{E}_{\tau\sim\pi(s_0)} \sum_{t=0}^{T} r_t \tag{1}$$

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

$$\begin{aligned} V^\pi(s) =& \mathbb{E}_\pi\left\{ \sum_{k=1}^{\infty} \gamma^k r_{t+k} | s_t = s \right\} \\ =& \mathbb{E}_\pi\left\{ r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s \right\} \end{aligned} \tag{2}$$

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

$$Q^\pi(s, a) = \mathbb{E}_\pi\big\{ r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s, a_t = a \big\} \tag{3}$$

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. 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)

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 (DRL)

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 DRL algorithm to show significant promise: 'Deep Q-Networks (DQN)' [**?**] learned to play Atari games from raw, unstructured pixel data to learn a parameterised approximation of (3) via stochastic gradient descent with the loss:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(\mathcal{D})}\left\{\left(r + \gamma \max_{a'} Q_{\theta_i^-}(s', a') - Q_{\theta_i}(s, a)\right)^2\right\} \quad (4)$$

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. It should be noted that the main contribution of [**?**] was in the formulation of a few simple techniques to allow for more stable training of deep function approximators in the RL setting. Most notably, 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 parameterized by θ_i^- (updated to equal θ_i every few iterations).

2.1.2 Multi-task-, Meta- 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 and Meta-RL generalise this by introducing the notion of tasks $\mathcal{T}_i \in \mathcal{T}$:

$$\mathcal{T}_i = \langle \mathcal{S}_i, \mathcal{A}_i, p_i(s_0), p_i(s'|s, a), r_i(s, a)\rangle$$

It follows that 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 framework was introduced as Universal Value Function Approximation (UVFA) in [**?**]. 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 $\pi(a|s,g)$.

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 too sparse (i.e. most transitions (s,a,r,s') have $r = 0$), then these transitions offer the agent no guidance on how to change its behaviour. If the reward is dense, it may introduce local optima that the agent must somehow overcome to reach the intended global optimum. As a result, *reward shaping* requires a domain expert and significant effort, and produces a reward function that is applicable only to the domain it was developed for.

Work in Intrinsic Motivation (IM) seeks to develop task-agnostic reward functions, based on general principles like information gain [], empowerment [], or surprisal []. 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 build a model of the world or to learn new skills. 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:

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

IM can therefore be used to turn a sparse external reward signal into a dense one, simplifying the task for RL agents. As such, IM can be considered an exploration mechanism.

2.1.4 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, i.e. the larger the exploration space. Standard RL algorithms generally perform poorly in such 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 [**?**]: 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. Alternatively, we can build an *Option hierarchy*: 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.

2.1.5 Model-based RL

Overall, goal is pretraining to reduce time to solve at test time... especially if environment interaction is expensive... this is similar to the goal of skill discovery, and implemented directly in DADS, but we choose not to learn a dynamics model. At any rate, it is interesting to put model-free RL into perspective, as it also informs our choice against...

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 [**?**], Claude Shannon demonstrated the unity of information media and showed that virtually any form of information could be encoded in bits.

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.

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:

$$\mathcal{I}(X;Y) = \underbrace{\mathcal{H}(X) - \mathcal{H}(X|Y)}_{forward\ MI} = \underbrace{\mathcal{H}(Y) - \mathcal{H}(Y|X)}_{reverse\ MI} \tag{5}$$

Because MI measures the mutual dependence between two variables, it is symmetric ($\mathcal{I}(X;Y) = \mathcal{I}(Y;X)$). In practice, working with the forward MI will involve different probability distributions than the reverse MI. 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*:

$$\mathcal{I}(X;Y) = D_{KL}(p(X,Y)||p(X)p(Y)) \tag{6}$$

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) [**?**] are a type of deep generative model. To understand them, we look first to their predecessor: the Autoencoder.

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*:

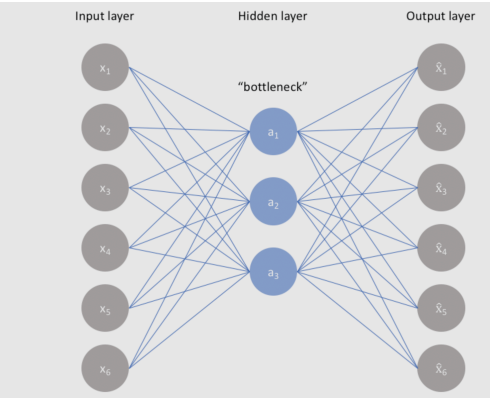

Figure 1: A simple autoencoder

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.

Extending this, we may assume that the training data $(x_i)_{i=0...n}$ is sampled from some underlying generative process $p(x|z)$. In the handwritten

digit example, aside from the digit itself, we might expect $p(z)$ to model 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:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} \tag{7}$$

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:

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \left\{ \underbrace{\mathbb{E}_{p_\theta(z|x)}\big[\log p_\phi(x|z)\big]}_{reconstruction\ loss} - \beta \underbrace{D_{KL}\big(q_\theta(z|x) \,||\, p(z)\big)}_{regulariser} \right\} \tag{8}$$

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 (8) we adopt the more general β-VAE 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. The goal of such approaches is to enable the agent to discover skills purely through 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.

We concentrate on a number of recent approaches motivated by information-theoretic objectives that define what may constitute a skill. For example, in Diversity is all you need (DIAYN) [**?**], Eysenbach et. al argue that skills

must primarily be discriminable, and should therefore effectively partition the state space. Therefore, they maximise the MI between states and skills $\mathcal{I}(s; z)$. Similarly, in Variational Intrinsic Control (VIC) Gregor et. al [**?**] maximise MI between the final state s_f of a skill and the skill itself, conditioned on the starting state s_0: $\mathcal{I}(s_f; z|s_0)$. DADS [**?**] maximises MI between the next state s' and the skill z, conditioned on the current state s: $\mathcal{I}(s'; z|s)$.

As in the classical Options framework, a skill is simply some temporally extended behaviour. Here, this is achieved by defining a latent skill space $\mathcal{Z}$, and conditioning a policy $\pi(\cdot|\cdot, z)$ on skill variables $z \in \mathcal{Z}$ for a certain number of time-steps. In the more common reverse form of the MI (discussed in the following), we train a probabilistic discriminator model $q_\phi(z|\tau)$ to map trajectories τ to skills z based on some information-theoretic measure. In practice, this is done with a Multi-layer-perceptron (MLP) or a Recurrent Neural Network (RNN). Finally, we use this model to compute intrinsic rewards $r_z(\tau)$ with which we train the *skill-conditioned* policy. A common choice for the skill-rewards is $r_z(\tau) = -\log q_\phi(z|\tau)$. This process is repeated until the discriminator and policy converge.

Algorithm 1: Information-theoretic Skill Discovery

Result: Policy $\pi_\theta(a|s, z)$
Given: skill prior $p(z)$;
Initialize policy $\pi_\theta(a|s, z)$, discriminator $q_\phi(z|\tau)$;
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 $\mathbb{E}[\log q_\phi(z|\tau)]$ using $\mathcal{D}$;
  compute intrinsic reward $r_z(\tau)$ using q_ϕ;
  update π_θ using any entropy-regularised RL algorithm;

In practice, many algorithms compute skills on a per time-step basis, rather than on complete trajectories. Learning skills on a trajectory basis requires recurrent models for the discriminator and policy, and creates a sparser reward signal. Inferring skills and rewards on a per time-step basis makes for simpler models and denser rewards, trading off conceptual validity for algorithmic simplicity.

DIAYN completely omits the notion of temporal abstraction in its definition of skills, while VIC maintains some form of it by inferring skills from starting and final states of a trajectory.

2.4.1 An analogy with VAE

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

$$\pi^*, q_\phi^* = \arg\max_{\pi, q_\phi} \left\{ \mathop{\mathbb{E}}_{c\sim p(c)} [\mathop{\mathbb{E}}_{\tau\sim\pi(\cdot|\cdot,c)} [\log q_\phi(c|\tau)] + \beta\mathcal{H}(\pi|c)] \right\} \tag{9}$$

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 ISD objectives require variational inference to be made tractable, resulting in these similarities.

This provides a useful framework for thinking about ISD. 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, we would expect this to be the other way around, to align more closely with the information bottleneck principle the VAE is based on. However, the iterative nature of these algorithms might make this more of a semantic difference, as illustrated below.

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, whereas in the VAE framework we aim to discover it.

The β-coefficient in the VAE framework corresponds to the α coefficient in entropy-regularised RL. This provides another useful intuition. Like β, α

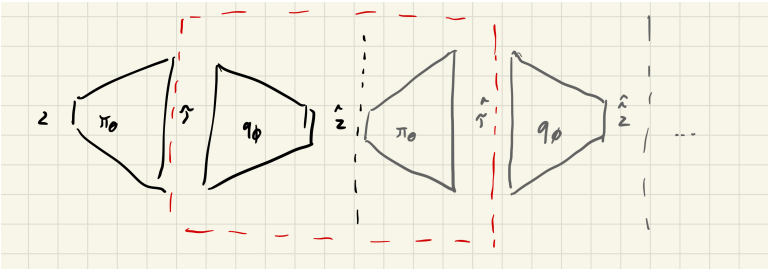

Figure 2: VAE - ISD equivalence

determines the level of *disentanglement* between skills, and thus increasing α should lead to a smoother (latent) trajectory 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, 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.

Regardless, this analogy justifies trying to leverage advances made in the training of VAEs for ISD, and provides a new angle from which to interpret these methods. For example, it has been observed that training a standard VAE (β-VAE with $\beta = 1$) is inferior to different weights for the regularisation term (e.g. $\beta = 0.1$ [**?**]), and that annealing β throughout training generally leads to improved performance. Fu et. al [**?**] describe a cyclical annealing schedule, increasing and then resetting β several times throughout training.

2.4.2 Applications

First and foremost, skills may serve as options in hierarchical reinforcement learning. However, 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.

1. **Transfer Learning & Finetuning**: Many of the goals we set an agent presuppose certain instrumental knowledge. For example, if we want an

agent to move to some goal-location, it should 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. For example, if we train a humanoid with 111 joints, there are a large number of possible gaits and movement patterns, but if we learn to move in the context of achieving a goal location as quickly as possible, we most likely learn only one. We would like to learn different gaits and movement patterns prior to optimising goals that build on these skills. This is true especially in the locomotion domain, where the agent is faced with the colossal challenge of learning to coordinate all its joints with only very little guidance.
We can imagine two ways to use skill discovery in this. First, we may elevate the agents action space by replacing (or augmenting) it with learned skills. Because skills are learned in a task-agnostic fashion, we would expect agents to effectively be solving a simpler MDP. Second, we can initialise an agents 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 can still improve sample efficiency.

2. **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 $z_* = \max_{z \in \mathcal{Z}}[q_\phi(z|\tau_e)]$.

3. **Continual Learning**: Since skill discovery provides a framework for learning from unsupervised interaction with the environment, it also provides a basis for continual learning. In fact, continual and multi-task learning are often used interchangeably in the RL literature, and as discussed previously, skill discovery can be seen as a bottom-up variant of multi-task RL. In single-task RL, the agent converges to a policy that solves this problem to the best of its ability, thus its ultimate capabilities are bounded by the given reward function. In skill discovery, the agent constantly proposes new tasks to solve itself, learning new reward functions, and hopefully new behaviours. The ultimate goal of skill discovery should be lifelong learning, though current approaches are at best a step in this direction.

4. **Adding supervision**: Skill discovery also provides a straightforward avenue for introducing human supervision into the learning process. For example, we might learn skills $q : f(\mathcal{S}) \rightarrow \mathcal{Z}$ from some function f of the states/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 short sequence of temporally extended behaviours, rather than a large number of actions that are difficult to assess individually.

3 Related Works

We give a brief overview of the state-of-the-art in information-theoretic skill discovery, to provide an intuition for the variability in these approaches.

actually explain the variability, instead of listing the approaches and letting the reader infer it themselves...

- objectives, forward vs reverse MI
- recurrent vs flat, sparse vs dense rewards
- skill prior: continuous vs discrete, learned vs fixed
- RL algorithm
- common failure modes, solution proposed in EDL

1. Variational Intrinsic Control (**VIC**) [**?**] maximises the mutual information between the set of options and option termination states $\mathcal{I}(s_f; z|s_0)$. In this, they choose to learn the prior of options $p(z|s_0)$. To justify this, consider three options z_1, z_2, z_3, where z_1 always leads to state s_1, whereas z_2 and z_3 always lead to s_2. We do not want to distinguish z_2 and z_3, and we would like the option prior to reflect this.
   The intrinsic reward computed for this objective is $r_z = \log q(z|s_0, s_f) - \log p(z|s_0)$, and this is optimised using any RL algorithm (like policy gradients or q-learning).

2. Diversity is all you need (**DIAYN**) [**?**] learns skills that partition the state space. The full objective is $\mathcal{I}(s; z) + \mathcal{H}(a|s) - \mathcal{I}(a; z|s)$, which rearranges to $\mathcal{H}(z) - \mathcal{H}(z|s) + \mathcal{H}(a|s, z)$. The first term implies that we maximise the entropy of the skill prior, the second that skills should be easy to infer from states, and the third that we want to learn maximum entropy policies for each skill.
   Two more key changes are made with VIC. First, we fix the option prior $p(z)$ to a uniform distribution (ensuring maximal entropy), rather than learning it. They find that this prevents the algorithm from collapsing to a small set of skills, as in VIC. Second, we use an entropy-regularised RL algorithm like Soft Actor-Critic (SAC) [**?**] to maximise intrinsic rewards, to encourage exploration and to ensure that each skill policy acts as

randomly as possible within the bounds of discriminability. This can be said to improve robustness of the policy.

3. Variational Autoencoding Learning of Options **VALOR** [**?**] extends these approaches to complete trajectories by using recurrent discriminator and policy architectures. In doing so, they maximise the mutual information between skills and trajectories $\mathcal{I}(z;\tau)$. They argue that this enables the decoder to correctly distinguish between trajectories that share some states. VALOR assumes a fixed skill prior and also uses entropy-regularised RL.
   The authors also propose a *curriculum approach*, whereby they increase the number of skills the agent learns as the confidence of the discriminator increases. They find that this improves stability and enables them to learn a larger number of skills.

4. Dynamics Aware Discovery of Skills (**DADS**) [**?**] takes a different approach. Unlike the other approaches, DADS learns the forward form of the mutual information. They maximise the mutual information $\mathcal{I}(s';z|s)$. This can be likened to a one-step version of VIC, though DADS does not learn a discriminator. Rather, they learn a *skill-dynamics* model $q(\Delta s|s,z),\ \Delta s = s' - s$, which predicts the next state s' from the current state s and skill z. Thus, DADS learns skills under which transitions are predictable.
   This leads to a slightly different intrinsic reward:

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

   In this, we approximate the intractable function $p(s'|s) \approx \sum_{i=1}^{L} q_\phi(s'|s,z_i)$ with L samples from the skill prior.
   There are a few advantages to this formulation. First, it enables model-based planning in the latent space, using the skill dynamics. Second, it provides a natural way of integrating continuous skill priors. Because discrete priors are usually represented with one-hot encodings, this allows for a more compact latent space, as well as smooth interpolation between skills. Third, it leads to lower-variance skills, that are therefore more amenable to hierarchical control. Like DIAYN and VALOR, DADS uses entropy-regularised RL to maximise intrinsic rewards.

There are a number of other algorithms, not described in detail here, but for completeness listed with their objectives.

1. Discriminative Embedding Reward Networks (**DISCERN**) [**?**]: $\mathcal{I}(s_g; s_T)$ the MI between goal states and actual achieved final states. This approach uses the discriminator to compute the intrinsic rewards, but otherwise resorts to goal-conditioned RL.

2. **Skew-Fit** [**?**]: $\mathcal{I}(s, g)$ - similarly, the mutual information is used to augment goal-conditioned RL, in this case to maximise the entropy over goals g.

3. Variational Information Maximising Exploration (**VIME**) [**?**]

→ Link to some reviews on the topic (IMGCRL, maybe I can find another one)

3.1 Explore, Discover, Learn

ISD algorithms that follow the scheme described in section 2.4 suffer from some common failure modes. These are induced by a *vicious cycle*: the discriminator q_ϕ is limited by the skill-conditioned policy π_θ, and vice versa. This may lead to pathological training dynamics, wherein the policy starts to practice only skills which it has already learned well, and thus may not cover parts of the state space. Following the analogy in section 4.1, we are performing lossy compression, and crippling an agent that uses these skills instead of actions. This is considered in *Explore, Discover, Learn* (EDL) [**?**], which breaks this circular dependency by dividing skill discovery into three distinct stages.

In this approach, we collect experience with some exploration policy $\pi(a|s)$ (in EDL: State Marginal Matching [**?**]), and then train a VAE on this *ground truth* experience. Learning a skill-conditioned policy can then be likened to *embedding* the VAE decoder in the MDP: we know how we would like skills to be decoded into trajectories, but we still have to train a policy that does this in the environment. Compared to ISD, this entire process need only be executed for one iteration (which does not imply that it terminates more quickly).

This framework is highly modular, and can therefore benefit from advances relevant to each stage (Explore exploration, Discover generative modelling, Learn - RL). Moreover, each stage can more easily be evaluated

independently of the others. However, the focus of EDL was on the discovery of state-covering skills, something they argue the earlier ISD methods do not. This is primarily because the agent receives larger rewards for visiting known states than previously unknown states, covering a distribution of states $p(s)$ induced by the skill-conditioned policy itself $p(s) = p_\pi(s) = \mathbb{E}_z[p_\pi(s|z)]$.

Perhaps the strongest argument for this approach is its understandability, and resulting comparability. Each stage has a clear purpose, therefore we can develop metrics to evaluate success. As pointed out in [**?**], no such metrics exist for ISD methods, and it is not easy to see what one could look like. So, to compare the success of different methods, we are restricted mostly to qualitative judgements.

4 Intuition

In the following section we will describe the intuition that underlies the general framework for hierarchical skill discovery described in section 6.

4.1 Skill Discovery as Action Space Compression

One way of approaching skill discovery, that follows from the previous section, is as compression of the agents action space. For now, we assume $\mathcal{S}, \mathcal{A}$ and $\mathcal{Z}$ are finite. Consider the space of all action-sequences of length T $\mathcal{A}^T$ the agent can take, and the number of skills N we wish to learn. Not all of these action-sequences will be worth distinguishing depending on the environment, and we may group those that are *similar enough* into a *skill*. The objectives in ISD then define how we determine similarity: which action-sequences can we group?

For example, if we simply map states to skills with the objective $\mathcal{I}(s; z)$ (as in [**?**, **?**]), we introduce the inductive bias that we want to partition $\mathcal{S}^T$ into N classes. So we are reducing $\mathcal{A}^T$ based on the states visited when following any given action sequence. The smaller N we choose, the larger the compression, and therefore the larger the information loss caused by it. For larger N, the information loss is smaller, but the gain from introducing an $(N + 1)$-th skill is also smaller. It is clear that an agent trained only on skills instead of primitive actions is solving a compressed (and therefore simpler) version of the original MDP, an advantage gained from unsupervised interaction with the environment. In other words, during skill discovery, the agent is exploring the realm of possibilities in the MDP and encoding this knowledge in skills.

We may also intentionally perform a lossy compression. For example, in a stochastic environment, we may choose to keep only the action-sequences that are *predictable* (e.g. [**?**]). We might define predictability in terms of how consistently the action-sequence produces the same sequence of states, or just the same final state. In this case, action-sequences that lead to very unpredictable dynamics are simply lost in skill discovery.

Moreover, in most environments not every trajectory in $\mathcal{S}^T$ is actually feasible (the exception being an MDP with $\forall s, s' \in \mathcal{S}\ \exists a \in \mathcal{A} : p(s'|s, a) >$

0, i.e. every state is reachable from every other state in one time-step). In environments where the transition dynamics are more sparse, there is more to gain from unsupervised interaction, by developing an implicit model (encoded in the skills) of what trajectories are actually feasible.

This compression induces a natural tradeoff: for N too small, an agent acting only on the learned skills may be crippled, i.e. it may become incapable of a useful behaviour if there is no corresponding skill. For N too large, the value of skill discovery is diminished, because the dimensionality of the trajectory space is not significantly reduced.

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 expecting anything better is, in our eyes, unreasonable. This formulation places the skill discovery problem in a (to the best of our knowledge) new context, and simplifies it somewhat.

To summarise: skill discovery constitutes a compression over the space of action-sequences for some skill length T, and we use unsupervised interaction with the environment to inform this compression.

How do we extend this argument to continuous (infinite) spaces? VIME may give a clue!

4.2 Integrating Hierarchy

Using learned skills in hierarchical reinforcement learning usually takes a secondary role in the papers discussed in section **??**. In the following, we discuss how we might perform skill discovery with a clearer focus on subsequent hierarchical composition.

Recall that skill discovery leaves us with a trained skill-conditioned policy $\pi_\theta(a|s,z)$, and a corresponding skill space $\mathcal{Z}$. The simplest way to use these skills is to 'freeze' π_θ, and to train another policy π^h that chooses a skill $z \in \mathcal{Z}$ every T time-steps.

We can easily extend this to more than one level, by performing skill

discovery over the modified MDP $\langle \mathcal{S}, \mathcal{Z}, p_l \rangle$, where p_l denotes the new transition dynamics induced by replacing primitive actions in $\mathcal{A}$ with skills. Of course, we can do this as many times as we like. This idea is formalised and generalised in Latent Space Policies for HRL (SAC-LSP) [**?**], though not immediately in the context of skill discovery. In it, we are given an MDP, as well as an ordered sequence $\{\mathcal{R}_0, \mathcal{R}_1, ..., \mathcal{R}_{K-1}\}$ of K reward functions, where K is the number of layers in the hierarchy. We then learn a sequence of latent-conditioned policies $\pi_i(\cdot|\cdot, z)$ as well as a latent prior $p(z)$, which solve $\mathcal{R}_i$. After each iteration we embed the new dynamics learned in the previous one into the environment. In the skill discovery case, this would effectively mean replacing the action space of the current level with the learned skill space. In this, the authors explore two architectures, one in which we freeze the current level and then learn the next (K-layer), and another in which higher layers can alter earlier levels (an "invertible" layer structure), effectively producing a single end-to-end policy. They find that the simpler stagewise training performs as well or better than the invertible approach. The two-layer invertible approach is similar to common HRL approaches, in which controller (high-level) and worker (low-level) policies are trained jointly. In the skill discovery setting, this approach is adopted by [**?**, **?**], whereas [**?**, **?**] discover deeper option hierarchies.

The options framework presupposes an initiation set I_ω and a termination probability β_ω for each option $(I_\omega, \pi_\omega, \beta_\omega)$. One simple way of defining these is given by the 'freeze-then-use' idea, with a close precedent in Diversity-driven Extensible HRL (DEHRL) [**?**]. Here, skills have a defined length of T time-steps. Thus, $I_\omega = \mathcal{S}$, that is all skills can be started from any state, and $\beta_\omega(t) = (1 \text{ if } t = T \text{ else } 0)$, where t is the number of time-steps the skill has already been carried out. Once again, we can not expect every skill to be useful from every state, but we can leave this to be determined by the policy using these skills. In a multi-level hierarchy, we define the skill-length T_l as a function of the level, where T_l is an integer multiple of T_{l-1}. A simple candidate is the exponential $T_l = c^l * T_0,\ c \in \mathbb{N}$.

The assumption $I_\omega = \mathcal{S}$ is also important. Methods like DIAYN and VALOR sample a skill at the beginning of every episode. Thus, as pointed out in [**?**], skills tend to radiate out from the starting state distribution $p(s_0)$. This makes skills less amenable to sequential composition, which is crucial for hierarchy. Thus, during skill discovery we should resample skills throughout the episode, forcing the agent to learn skills that can follow and be followed by other skills. This makes the overall skill discovery problem

more difficult, but if solved successfully will produce more flexible skills.

This deep hierarchical approach has a simple and appealing justification, following from the arguments made in section **??**. Again, assume finite $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{Z}$. Assume we would like to discover skills with a length of 1000 steps, therefore over the space of trajectories $\mathcal{A}^{1000}$. Instead of doing this directly, we might wish to adopt a 3-level hierarchy, learning skills of length 10 at the first level, of length 100 at the second, and length 1000 at the third. However, this means that at each level a skill only constitutes 10 policy decisions, since it is using the skills from the previous level. Thus, we have reduced the problem of skill discovery over trajectories of length 1000, to three skill discovery problems of length 10. Because the size $|\mathcal{A}|^T$ of the trajectory space we are encoding is exponential in T, we suspect that this significantly simplifies the overall problem. We can imagine a 'solvable' size for the skill discovery problem, to which we may then reduce any larger ones.

canonic layerwise skill discovery algorithm, also add visualisations

5 Problem Statement

In this thesis, we aim to synthesise information-theoretic skill discovery and hierarchical reinforcement learning, to learn composable skills in a bottom-up, unsupervised manner. Overall, we want to design a skill discovery method with the goal of layerwise hierarchical assembly in mind, i.e. this skill discovery method should be recursively applicable. Fundamentally, we want to elevate the action space of the agent, such that it is better prepared for any task

The primary component of this algorithm is skill discovery: given an MDP $\langle \mathcal{S}, \mathcal{A}, P, p(s_0) \rangle$, we want to learn skills Ω such that we can train an agent in the modified MDP $\langle \mathcal{S}, \Omega, P_\Omega, p(s_0) \rangle$ where P_Ω is derived from P by extending transition dynamics over multiple time-steps. These arise implicitly from learned skills, and do not actually need to be computed. In this we define a skill $\omega \in \Omega$ as the behaviour achieved when conditioning the agent policy $\pi_\theta(a|s,z)$ on some $z \in \mathcal{Z}$ for T time-steps, creating a direct mapping from $\mathcal{Z} \to \Omega$. This definition deviates from the one used in ISD literature only by defining all skills to be a specific number of time-steps, though prior methods necessarily do this too when using skills for HRL, and during discovery by fixing the episode length.

$\to$ canonic skill discovery as described in section 2.4

Extending this to multi-level hierarchies: we learn a sequence of skill-conditioned policies $\pi^i(a|s,z)$ where $z \in \mathcal{Z}_i$, $a \in \Omega_{i-1}$, $\Omega_0 = \mathcal{A}$, $s \in \mathcal{S}$. In this, we generally define and fix $\mathcal{Z}_i$ such that an intuitive choice is $\mathcal{Z}_i = \mathcal{Z}$ for some appropriate $\mathcal{Z}$. From this, we see that we can apply the same skill discovery method at every level of the hierarchy by abstracting the notion of time-steps into the number of policy decisions, implicitly defining a simple exponential function for the length in time-steps T^i of skills at the i-th level. However, this is a hyperparameter that is simple to tune, and we may think of more appropriate functions.

$\to$ canonic layerwise hierarchy as described in section 4.2

Thus, the main aim of this thesis is to develop an algorithm that learns skills that are amenable to hierarchical composition. Such skills should fulfil two criteria, generally only weakly fulfilled by prior ISD methods:

1. Skills are applicable in states outside the starting state distribution, i.e. we can sequence skills.
2. Skills learned over an action-sequence space $\mathcal{A}^T$ should sufficiently cover the space of possible trajectories $\mathcal{S}^T$, i.e. after skill discovery completes we can replace $\mathcal{A}$ with the learned skill space Ω without crippling the agent.

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. We use DIAYN as a baseline, because it is simple to implement and one of the seminal papers in skill discovery.

Since the multi-level hierarchy may be seen as the central focus of this thesis, we formulate two research questions to help us evaluate its potential:

1. Are shorter skills easier to learn than longer ones?
This question is fundamental to this thesis, although it concerns only the base skill discovery method. If the gain in simplicity from learning shorter skills is minimal, we should not expect a hierarchy to incur much benefit.

2. What is the impact of increasing the number of layers?
By the intuition described in section 4.2 we would aim to maximise the number of layers within the constraint of skills having an integer length. However, this depends heavily on the success of the skill discovery method. In practice, we hope to see gains from introducing an additional layer, with diminishing marginal returns.

5.1 Success Indicators

- Discriminator confidence $\mathbb{E}_{\tau\sim D}[\log q_\phi(z|\tau)]$: the discriminator should be able to map trajectories to skills confidently.
- Cumulative intrinsic reward achieved by policy: if the policy effectively *generates* the skills learned by the discriminator, it achieves higher intrinsic reward.
- State space coverage: as in success criterion (2), 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.

- Sample efficiency: we measure sample efficiency mostly by the number of environment steps taken by the agent, as this is the usually the most computationally expensive part of learning.
- Stability: the random seed and stochasticity of the algorithm should not significantly affect success according to the described indicators.
- Performance on a set of evaluation tasks: we may compare an agent operating on the given MDP with one operating on the skill-MDP, 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.

5.2 Environments

For the most part, we will be testing our skill discovery agent in different 2-d navigation environments, as these allow for simpler analysis of results. However we will also run the agent in the locomotion domain, in environments built on the MuJoCo physics engine, like the Cheetah, Ant and Humanoid. These environments are significantly more complex, but also allow for much more interesting and semantically diverse skills, and better reflect the target environment of this algorithm (skill discovery offers little benefit in environments easily solvable with flat RL). **show the environments and explain them in a bit more detail**

6 Hierarchical Skill Discovery

In this section, we describe our algorithm for hierarchical skill discovery (hSD). 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. The choices made will reflect the intuition that the latent skill variable $z \in \mathcal{Z}$ defines a skill-MDP $\langle \mathcal{S}, \mathcal{A}, p, p_z(s_0), r_z \rangle$. In the discrete case, we could imagine defining a separate policy π^z for each skill variable. We want the agent to be able to act confidently within this mini-MDP, such that conditioning the policy $\pi_\theta(a|s,z)$ for T time-steps leads to predictable and repeatable behaviours, which are diverse over the latent skill space $\mathcal{Z}$.

First, we develop a skill discovery algorithm with the ultimate goal of hierarchy and composition in mind. Then we show how to use this algorithm to learn a deep option hierarchy in a layer-wise fashion. In the following, we describe how we extend DIAYN to a continuous skill space $\mathcal{Z}$, and propose a simple state normalisation mechanism to learn composable skills, by forcing the agent to infer skills from *state-differences* rather than states. Figure **??** summarises the approach.

6.1 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. In other words, we maximise $\mathcal{H}(\mathcal{A}|\mathcal{S},\mathcal{Z})$, weighted with coefficient α. This gives us the following complete objective:

$$\begin{aligned}\mathcal{F}(\theta) &= \mathcal{I}(\mathcal{S};\mathcal{Z}) + \alpha\mathcal{H}(\mathcal{A}|\mathcal{S},\mathcal{Z}) \\ &= \mathcal{H}(\mathcal{Z}) - \mathcal{H}(\mathcal{Z}|\mathcal{S}) + \alpha\mathcal{H}(\mathcal{A}|\mathcal{S},\mathcal{Z})\end{aligned} \tag{10}$$

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

Contrary to DIAYN, we adopt a continuous uniform distribution for $p(\mathcal{Z})$ and choose $\mathcal{Z} \in [-1, 1]^D$, where D represents the dimensionality of the skill space. We find 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. 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. However, we are learning a simpler and arguably more general objective than the one chosen in DADS.

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 $[0, 1]^{num\_skills}$, of which the space of one-hot vectors of length num_skills is a subspace. In the continuous version, the entire latent space is utilised and meaningful.

Moreover, prior works build on Soft Actor-Critic, 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. Less importantly, a continuous space can be encoded significantly more compactly. 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.

This choice 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:

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

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:

$$
\begin{aligned}
\nabla_\phi\mathbb{E}_{s,z}\big[\mathcal{D}_{KL}(p(z|s)\;||\;q_\phi(z|s))\big] &= \nabla_\phi\mathbb{E}_{s,z}\left[\log\frac{p(z|s)}{q_\phi(z|s)}\right] \\
&= -\nabla_\phi\mathbb{E}_{s,z}\big[\log q_\phi(z|s)\big] \qquad (11)
\end{aligned}
$$

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:

$$
r_z(s,a) = \log\frac{q_\phi(z|s)}{\frac{1}{2L}\sum_{i=1}^{L} q_\phi(z_i|s)}, \quad z_i\sim p(z) \qquad (12)
$$

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.

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 $p(s'|s) = \int_{\mathcal{Z}} p(s'|s,z)dz \approx \sum_{i=1}^{L} q_\phi(s'|s,z)$, thus the sum corresponds to a marginalisation over the skill variable z. This difference is just a consequence of our choice of the reverse MI. If (like DADS) we chose the forward MI, we would perform a similar marginalisation to approximate the intractable $p(s)$. We consider this a somewhat arbitrary choice, since both forms can be learned simultaneously in a straightforward manner. However, we choose the reverse MI to mirror DIAYN, and because we consider imitation learning a more interesting application than model-based RL.

6.2 State Normalisation

Skills learned by prior methods are not composable, since we sample a skill at the beginning of every episode and keep it constant throughout. Thus, skills radiate out from $p(s_0)$, and lose their meaning when initiated outside this distribution. In DADS, the authors point out that resampling skills throughout the episode increases the hardness of the problem. We explain why, and then propose a simple state normalisation mechanism to remedy this.

For maximally general skills, the starting state distribution approximates a uniform distribution over all states $p_z(s_0) \approx \mathcal{U}(\mathcal{S})$. In the Options framework, this is equivalent to $\mathcal{I}_\omega \approx \mathcal{S}$, as discussed in section 4.2. In this case, a skill can only be learned if it is applicable from many states. More restrictive $p(s_0)$, for example fixed to a single state, simplify this problem but limit the generality of learned skills. This is easy to see, as shown in Figure 3 in the 2-d navigation environment.

adjust this to show the problem with DIAYN instead of DADS

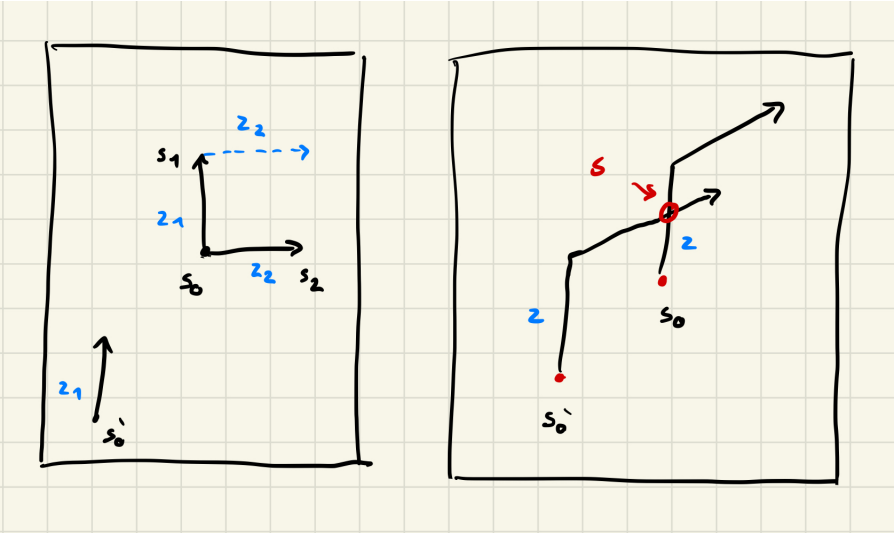

Figure 3: DADS failure modes

clarify the role of the skill-variable in informing behaviour of the policy, which is conditioned on s and z, we want the combination of the two to be maximally informative of the action the agent should choose...

Now assume skills are learned from a single starting state s_0. Policy $\pi(a|s,z)$ and skill $q(z|s)$ models are both conditioned on a specific state s. This imposes a strict state dependence on both models. More broadly, whenever we choose a skill outside its usual state distribution, it is at first completely in the dark, and must effectively learn to map known states to this unfamiliar distribution. This is an ill-posed problem, because neither model receives any *trajectory* information: to know how to act in some state s, it must know roughly *where in the skill trajectory* s is. This problem is illustrated on the right. The same skill z, initiated from s_0 and s_0' may pass through the same state s, but at different points in the overall trajectory. For the same input (s,z) the models now need to learn two different outputs. Even with probabilistic models, this is possible only in a limited form.

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 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 moves through the same normalised states $(\mathbf{0}, \bar{s_1}, \bar{s_2}, ...\bar{s_T})$. In general, we can say that in environments in which the transition dynamics do not vary around different states, no behaviour-relevant information is lost in this normalisation: the agent does not need to adjust a skill policy for different state distributions.

With state-norm, we can resample skills throughout the episode without really increasing the difficulty of the problem. We trade off ease of learning inflexible skills for the slightly more difficult problem of learning flexible skills, which is undoubtedly worthwhile. By resampling skills with state-norm throughout the episode, we can approximate a much broader starting state distribution $p(s_0)$. Hopefully, because of this resampling mechanism, the skills learn to move between different *stable states*, that is 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 2dNav), they do not, we do not need to resample. In more complicated environments, we would expect there to be more to gain from this.

Give an example of this... In a maze environment, a skill can't be carried out equally predictably from different states... Also, semantically different from skill definition in previous works... less fragile to distribution shift, already introduced mechanism to deal with it...

In more complicated environments, in which the transition dynamics vary around different states, the agent needs to learn skills that are achievable regardless of these varying dynamics. This takes a step towards $\mathcal{I}_\omega \approx \mathcal{S}$, and should lead to skills that can be composed sequentially. For example, in the Humanoid environment, we might imagine a skill z_1 that takes one step with the right leg, and another z_2 with the left leg. If z_1 is learned such that after executing it the humanoid is in a state in which it it can effectively execute z_2 (and vice versa), we can form the more temporally extended behaviour of walking.

Our continuous extension of DIAYN (cDIAYN), along with state normalisation give us the following skill discovery method:

Algorithm 2: 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 12;
  Update π_θ using Soft Actor-Critic;

6.3 Skill hierarchies

As discussed in section 4.2, we aim to replace the action space $\mathcal{A}$ with the skills learned during skill discovery, and thus to elevate it to a higher level of temporal abstraction. The key inductive bias we introduce here, is that skills have a specific, fixed length. As with any inductive bias, this removes some flexibility from the algorithm, and we could imagine scenarios in which it might prevent the agent from learning useful behaviours. However, we think it will simplify the skill discovery problem, and lead to more meaningful skills.

In the previous section, we derived an algorithm that learns skills of length T, encoded in continuous latent skill vectors $z \in \mathcal{Z}$. By replacing the action space with the learned skill space, and rolling out the corresponding skill-conditioned policy $\pi_\theta(\cdot|\cdot,z)$ for T, we create a new MDP, in which the agent makes decisions at a reduced frequency, acting on a learned policy in-between. In this way, we are *embedding* the learned skills in the environment.

Alternatively, we could define an option space with a bijective function

$f : \mathcal{Z} \to \Omega$, where $f(z) = (\mathcal{S}, \pi(\cdot|\cdot, z), \beta)$, $\beta(t) = 1$ if $t = T$ else 0, where t is the number of time-steps the current skill has been rolled out.

Thus, we may construct a skill-MDP $\langle \mathcal{S}, \mathcal{Z}_i, p'_i, p(s_0) \rangle$ from another MDP $\langle \mathcal{S}, \mathcal{Z}_{i-1}, p'_{i-1}, p(s_0) \rangle$ given a skill-conditioned policy $\pi(a|s, z)$ which chooses actions $a \in \mathcal{Z}_{i-1}$. We get modified transition dynamics p_i by *embedding* π in the previous layers environment. This gives us a sequence of policies, in which each policy makes sense only in the corresponding skill-environment. As a consequence, to (for example) execute a single layer 3 policy step, we take T_2 layer 2 steps and $T_1 * T_2$ layer 1 steps, where the cumulative number of layer 1 steps always corresponds to the number of steps taken in the base environment. It follows that each layer i policy depends on all policies from layers $(1, 2, ..., i-1)$. It should also be noted that the state-space remains the same in every layer-MDP.

We should also point out that adding layers in training strictly adds to the number of environment steps taken. Thus, adding layers may be regarded as a form of pretraining in solving the final skill discovery problem for skills of length T. Even though a higher-layer policy acts in a modified environment with a reduced decision frequency, in order to carry out the chosen option, we must ultimately take as many steps in the base-environment as we would in a single-layer hierarchy.In section 4.2, we propose reducing the problem of skill-discovery to smaller, more solvable but analogous problems, from which we may construct the overall solution. Still, we hope that this will lead to more sample-efficient and stable training.

We may train our skill hierarchy according to the following pseudocode.

Algorithm 3: Learn skill hierarchy

Result: Deep Policy Hierarchy $(\pi_{\theta_1}(a|s, z_1), \pi_{\theta_2}(z_1|s, z_2), ...)$
Given: base-MDP $\langle \mathcal{S}, \mathcal{A}, p, p(s_0) \rangle$, number of layers N, skill-length function T_i, Skill discovery method $SD : \langle \mathcal{S}, \mathcal{A}, p, p(s_0) \rangle, T \to \pi_\theta(a|s, z)$;
Initialise ;
for $i \leftarrow 1$ **to** N **do**
    $\pi_{\theta_i}(z_{i-1}|s, z_i) = SD(\langle \mathcal{S}, \mathcal{Z}_{i-1}, p_{i-1}, p(s_0) \rangle, T_i)$;
    Embed π_{θ_i} in the environment to form layer-$(i+1)$ MDP;

There are many ways we could use the learned skill hierarchy at test time. On the two ends of this spectrum, we could either allow the agent access only to the highest layer skills $\mathcal{Z}_N$, or to all layers ($\mathcal{Z}_0 \cup \mathcal{Z}_1 \cup ... \mathcal{Z}_N$), including primitive actions $\mathcal{Z}_0 = \mathcal{A}$. We encounter a tradeoff between reducing the agents search space and simplifying the problem, and potentially crippling the agent. Because using the skill hierarchy is not the immediate focus of this thesis, the following pseudocode is left deliberately broad. However, to test the success of our option hierarchy, we will evaluate the mentioned extremes (only the top layer vs. all learned skills) against only primitive actions.

6.4 Implementation

show how we can map the full algorithm to the interfaces defined for the implementation. would be cool to have a UML diagram or probably rather a system flowchart or something

tf-agents api, correspondence of interfaces to existing ones...

This is where description of Rollout driver, skill model, policy learner, and environment wrappers come in, as well as a brief overview and discussion of all the hyperparameters.

7 Experiments

In the following, we will evaluate the success of our method by answering the following research questions:

1. Does cDIAYN learn discriminable skills?
2. Are temporally shorter skills easier to learn than longer ones?
3. Can we use state normalisation to learn composable skills?
4. Can we learn deep option hierarchies by repeatedly applying cDIAYN?
5. Can hcDIAYN learn useful skills in complex environments?

We will attempt to answer these questions according to the success criteria described in section 5.1. We will answer questions 1-4 using the 2d navigation environment (2dNav) for easier communicability of results, and subsequently perform a more holistic evaluation in more complex environments.

7.1 Does cDIAYN learn discriminable skills?

To test this, we compare cDIAYN with the discrete version in the 2-d navigation environment. Here we have used an entropy-weight of 0.1 for both algorithms, as in DIAYN. In cDIAYN, we visualise skills spread evenly over the skill space $[-1, 1]^2$, choosing latent dimension of 2, and we learn DIAYN with 8 skills. This comparison may be seen in figure 4.

The continuous version takes longer to converge for the chosen α, which we attribute to it learning a more complex function. If we increase the number of skills in the discrete case, learning very quickly becomes even slower. We can also see that cDIAYN learns a smooth latent space.

— skill interpolation and smoothness of discriminator latent space...

We observe an undesirable training dynamic in the discrete case, which we also observe in the high entropy continuous case: skills converge to cover

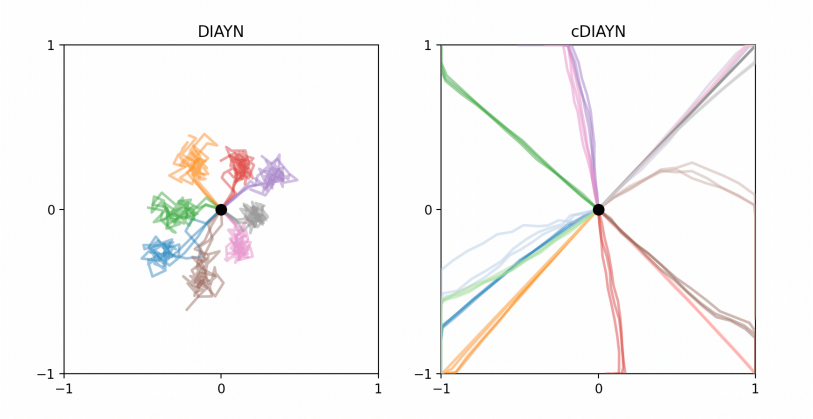

Figure 4: DIAYN vs. cDIAYN, 100 epochs

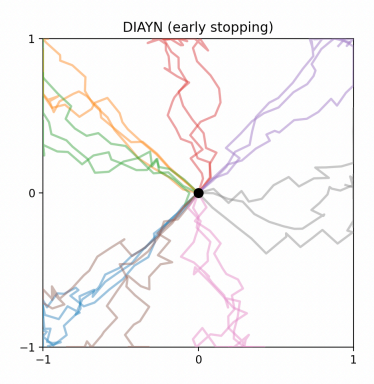

Figure 5: DIAYN after 40 epochs

a small set of discriminable states. If we end training earlier (in this case after 40 epochs), we get results similar to those shown in the paper.

Although this does not reflect our desired notion of skills, it might be a flaw of the objective itself. As described in the paper, the optimal solution in DIAYN evenly partitions the state space among skills, and learns skill-policies that approximate a uniform distribution over states within each partition. We believe this can only lead to poor skills, primarily because the guidance provided to the policy by the latent skill variable z is too coarse, i.e. the bottleneck imposed by the skill variable too strong. In the above case, states closer to s_0 are reached more often, are thus more discriminable,

receive higher rewards, and are therefore preferred by the policy. The agent explores more effectively earlier in training because states around s_0 are still more difficult to discriminate, so the agent is forced to diversify to learn at all. As a result, we find that with more training, skills eventually collapse to the closest discriminable states.

This same training dynamic is exhibited by our method cDIAYN, more strongly when the entropy is higher. Our method cDIAYN can learn a *large* number of skills in a more stable and sample-efficient manner. However, it exhibits this same convergence behaviour, more strongly when the entropy is higher. The reason we can say that this does not correspond with our understanding of skill discovery, is that we want skills to diversify further as learning continues. The optimal solution (especially in 2dNav) should visit every part of the state space reachable to it.

Overall, we can say that continuous latent variables can provide a more informative signal. We think that the low variance of skills learned by DADS may be a consequence of the continuous skill space, rather than the altered objective $\mathcal{I}(s'; z|s)$. It would be interesting to see whether DADS converges to similar pathological solutions, given enough training time.

We find that the entropy-regularisation term has a significant impact on the skills ultimately learned by the algorithms. This is clearer in cDIAYN, because it eliminates the choice of the number of skills, which fulfils a similar role. Recall that entropy-regularisation is weighted with coefficient α in our objective $\mathcal{I}(s; z) + \alpha\mathcal{H}(a|s, z)$. Choosing small alpha impedes learning, because the policy takes a long time to achieve a good coverage of the state space (figure 7), 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).

This suggests that an intermediate approach, in which we start with higher entropy and slowly reduce it, initially encouraging exploration, but ultimately learning low-variance skills, more useful for hierarchical control.

The SAC authors also proposed a version of their algorithm (EC-SAC) 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). We experimented with EC-SAC, but found it resulted in a strange convergence behaviour in both the discrete and

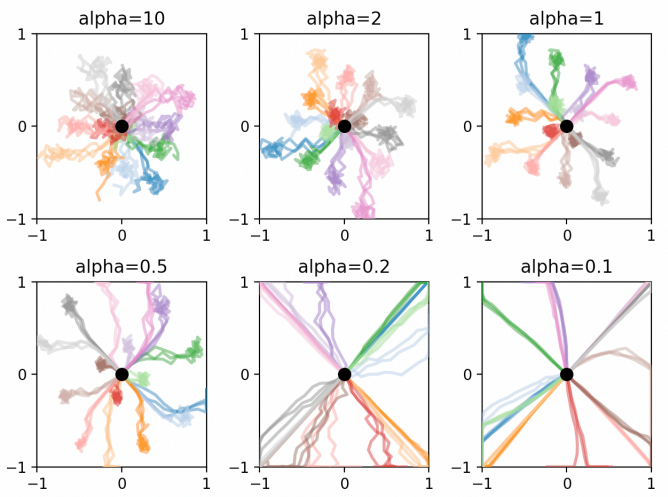

Figure 6: cDIAYN after 100 epochs for different value of α

continuous versions of DIAYN. α continues to increase, and as a result the skills collapse in on themselves, slowly regressing back to a random policy. We are not sure whether this problem is specific to the implementation, or perhaps a consequence of the skill discovery context within which we are using EC-SAC.

7.2 Can we use state normalisation to learn composable skills?

We want to learn skills in such a way that they can be sequenced, according to the simple state normalisation mechanism described in section 6.2. Depending on the hyperparameters, this is possible without state-norm in our 2d navigation environment, but only if the policy $\pi(a|s, z)$ learns to ignore the state s, and simply maps a skill z to a specific action a.

However, if we choose a large fixed entropy, the policy learns "achieve target state behaviours" as described above, and we can show how this leads to a failure when initiating a skill outside the usual starting state distribution $p(s_0)$, which in our case is fixed to $s_0 = (0, 0)$. This is because skills deliberately only learn how to behave within their allocated parts of the state space. It also becomes unclear how we would like our policy to behave then, and what we understand as a skill. Should the policy learn

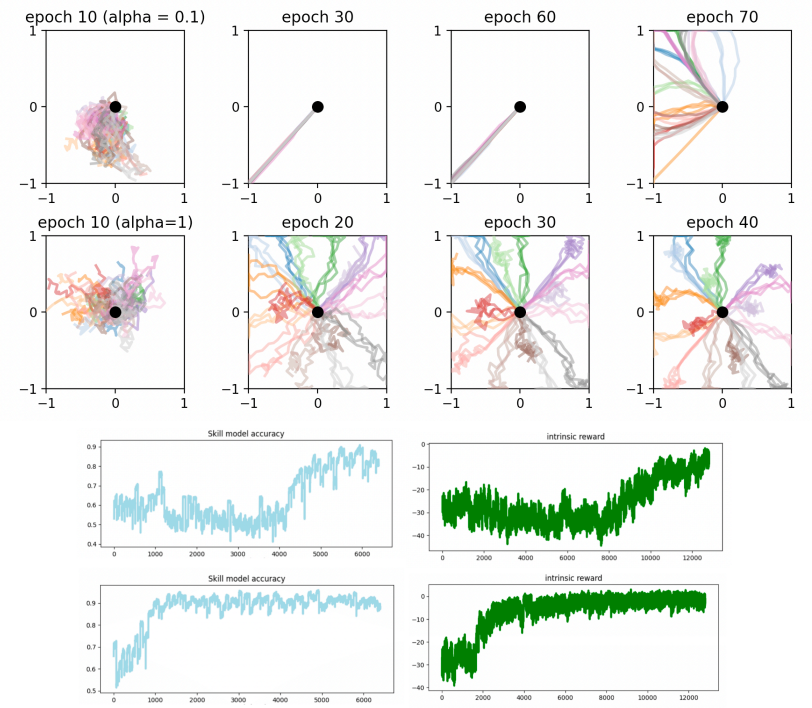

Figure 7: Poor local optima during training with low entropy ($\alpha = 0.1$ - top)

to reach that part of the state space associated with the current skill, from any other part of the state space? We induce a more concrete definition of skills.

— diagram of high entropy failure of outside distribution use - means: visualise skills, and then use a skill starting from $s \neq s_0$ and hopefully it fails? —

For this reason, previous works have assumed that a skill remains fixed throughout an episode. If we resample skills during episodes without applying state-norm, this effectively breaks learning. However, even if we learn skills without resampling and state-norm, applying state-norm on learned skills immediately makes them applicable outside $p(s_0)$. This again traces back to the unvarying transition dynamics in 2dNav.

— show no learning with resampling? yea, it just doesn't converge...

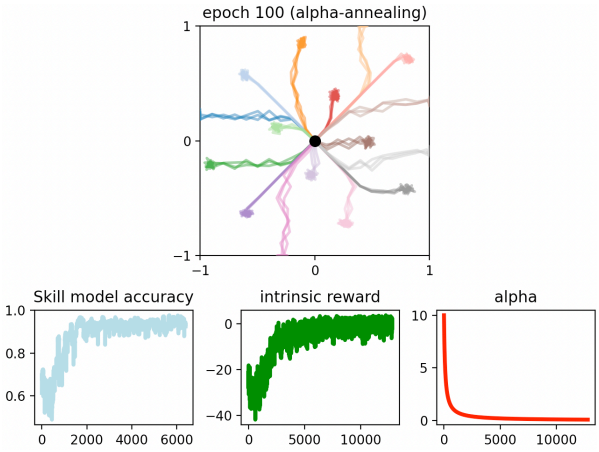

Figure 8: Annealing alpha leads to more stable training and better state space coverage

With state-normalisation, skills become dynamic behaviours. Both the normalised state $\bar{s}$ and skill variables z offer strong information based on past experience to guide the next action a. In 2dNav, state-norm unsurprisingly allows effective learning even when skills are resampled throughout the episode, and used with state-norm, skills can be effectively reused from any state afterwards.

— diagram of skills being sequenced in the way we would expect...

As described in section 6.2, this is not particularly exciting in 2dNav, because the transition dynamics are the same around every state (with exception of those near the edges). Because of this, it does not matter whether we learn skills with or without resampling and state-norm, as long as we apply normalisation during out of distribution use. However, once again, the effects described above would likely only become more pronounced in more complex environments.

7.3 Are temporally shorter skills easier to learn than longer ones?

Although we can intuitively expect this to be the case, this question is worth asking because it is fundamental to this thesis. If shorter skills are

only marginally easier to learn than longer ones, then it is doubtful whether extra layers during training will yield much benefit. In these experiments, we keep all hyperparameters constant, and vary only the skill length and the number of collect steps per episode, increasing the latter proportionally with skill length such that we always sample the same number of skills per epoch.

— here a graph: state-space coverage vs. time-steps, with graphs for different traj lengths - I can just compute this with the saved policies—

We find that even in 2dNav, as skill length is increased, learning becomes increasingly unstable. The agent gets stuck in local optima, which it has to escape before it can converge to the correct solution. We also find that time to convergence increases more than linearly with the skill length T.

— linearly increasing skill lengths vs. convergence behaviour (stability, environment steps, clock time...)—

Moreover, we would expect that this effect becomes more pronounced the larger the dimensionality of the action space, roughly reflecting the worst case problem complexity $|\mathcal{A}|^T$. *We could test this without visualisation in higher-dimensional point environments... worth doing?*

7.4 Can we learn deep option hierarchies by repeatedly applying cDIAYN?

After skill discovery is completed, we can replace the action space of the agent with the learned skill space. If skill discovery was *successful*, then there is no reason why we couldn't repeat this process, as argued in section 6. Here, at least in 2dNav, we can show that this works.

As shown in figure **??**, skill discovery with $T = 100$ is difficult: learning is unstable, and even with much patience we do not see convergence. If we decompose this problem into a 2-layer skill discovery problem with lengths $T_i = 10$, $i = \{1, 2\}$, it becomes solvable.

— diagram with two layer solution... achieving good coverage in same amount of time steps

Of course, training the second layer of the hierarchy takes much longer than the first. In the second layer, although only 10 2nd-layer policy decision are made, we are still taking 100 environment steps, like in the flat ($T = 100$) case. This is why adding more layers will eventually become counter-productive. We show learning in 2dNav with larger numbers of layers.

— 3, 4 and maybe 5-layer hierarchy

As you can see ...

7.5 Can hcDIAYN learn useful skills in complex environments?

This is the most important, and the most interesting question we attempt to answer in this thesis.

8 Discussion

9 Conclusion and Future Work

- importance sampling for intrinsic reward computation - should be more different from points that are further away than closer in latent space
- extension into off policy algorithm (shouldn't be difficult, just use importance sampling)
- α-annealing entropy regularisation as proposed in the thesis
- more evaluation in complex environments
- layer invertibility (as in SAC-LSP)
- development of metrics - on one hand: comparability, on the other: optimisation - this is an important one, and we have some ideas, motivated by the point environment... (mention how we could use oracle for this...)

10 Appendices

References