Ph.D. DISSERTATION

Generalizable Agents with Improved Abstractions and Transfer

향상된 추상화와 전이를 이용한 일반화 가능한 에이전트

August 2023

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Jaekyeom Kim

Generalizable Agents with Improved Abstractions and Transfer

향상된 추상화와 전이를 이용한 일반화 가능한 에이전트

지도교수 김 건 희

이 논문을 공학박사학위논문으로 제출함

2023 년 05 월

서울대학교 대학원

컴퓨터공학부

김 재 겸

김 재 겸의 공학박사 학위논문을 인준함

2023 년 06 월

위 원 장	유 승 주	(인)
부위원장	김 건 희	(인)
위 원	송 현 오	(인)
위 원	이 준 석	(인)
위 원	이 기 민	(인)

Abstract

Many researchers in the field of deep learning have been trying to build agents that perform a wide range of tasks. Since training on all the possible tasks is often not viable, improving the generalization of agents to novel tasks based on what they learn from training tasks has been one of the important challenges in deep learning. For effective generalization, both learning abstractions that can be used under different conditions and the exploitation of the abstractions on new tasks are crucial. In this thesis, we explore the challenge of generalization mainly in those two aspects, abstraction and transfer.

First, we study how to abstract input data and learn features that are robust to noise. As task-irrelevant information during inference can hugely impact the performance of learned models and agents, establishing robustness to such noise is an important problem in generalization. To tackle the problem, we propose a discrete information bottleneck method named *Drop-Bottleneck*, which learns to discretely drop features that are irrelevant to the target variable and distill features of interest. It enjoys not only a simple information compression objective but also provides deterministic compressed representations, which are useful for inference with complete consistency and improved efficiency due to the reduced number of features.

We then investigate how the agent can discover inherent behaviors in the environment without supervision and abstract them into skills in a more reusable form. Unsupervised skill discovery aims at finding and learning a set of useful behaviors by interacting within the environment but with no external rewards.

It is one of the key challenges of temporal abstraction in reinforcement learning as it allows the agent to reuse the knowledge of the learned skills and solve new tasks more efficiently and effectively. To the goal, we suggest an unsupervised skill discovery method named *Information Bottleneck Option Learning (IBOL)*. It seeks extensive behaviors in the state space by linearizing the environment and abstract those behaviors with disentanglement encouraged with the imposition of information bottleneck for improved reusability of the skills.

Lastly, we probe a way to leverage the knowledge learned from source tasks to improve the performance on target tasks without further training. For zero-shot transfer in reinforcement learning where the reward function varies between different tasks, the successor features framework is one of the popular approaches. Our goal is to enhance the transfer of the learned value approximators with successor features to new tasks by bounding the errors on the new target tasks. Given a set of source tasks with their successor features, we present lower and upper bounds on the optimal values for novel task vectors that are expressible as linear combinations of source task vectors. We then propose *constrained GPI* as a simple test-time approach that can improve the transfer by constraining value approximations on new target tasks.

Keywords: Deep Learning, Deep Reinforcement Learning, Skill Discovery, Temporal Abstraction, Transfer Learning

Student Number: 2018-28413

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In deep learning, building more generalizable agents is one of the primary challenges. While trained agents can perform well on the tasks on which they were trained, their generalization to novel tasks may not be as straightforward in many cases. The issue could become severer as the test tasks deviate more from the training tasks. As training agents for every possible task is infeasible, it is an important research direction to improve the generalization of agents to novel tasks. In this thesis, we consider two perspectives for generalization mainly in deep reinforcement learning (RL): abstraction and transfer.

To pursue generalization across tasks, it is prevalent to learn to abstract information from the training tasks so that it can be exploited by the agents. While one common strategy is to learn an abstract method for given inputs for its transfer to different tasks (Chapter 2), we also discuss problems and approaches where the agents find and extract information, such as behaviors, for the transfer of its abstraction to novel tasks (Chapters 3 and 4).

Depending on what the agents desire to transfer to new tasks, different approaches could be taken. Learned abstraction methods can be used as-is or leveraged with consistency and improved efficiency for inference (Chapter 2). Acquired behavior abstractions might become building blocks for higher-level, meta-controllers (Chapter 3) or ingredients for improving their behaviors on novel tasks (Chapter 4).

On the other hand, *generalization* is a very high-level goal, and many different problems can be grouped together under this direction at the highest level. In this thesis, we primarily focus on problems in deep RL or those that could help solve challenges within deep RL. Given the views above, we describe the specific challenges we tackle in this thesis as follows.

Fragility to Task-Irrelevant Information. While deep learning models are known to be effective in various types of tasks, it is not uncommon to observe trained models or agents get easily distracted by the noise in inputs and degraded in their performance. For instance, in the field of RL, there is an observed phenomenon that curiosity-seeking agents might get stuck in situations with noisy but novel observations [24]. Thus, it is crucial to study how to overcome such vulnerabilities to noisy or task-irrelevant information. One popular approach is to introduce the information bottleneck [106] to the learning. The information bottleneck is a framework for, given an input variable X, deriving a new variable Z from X while preserving only information about the target variable Y, which is achieved by minimizing the mutual information (MI) between Z and X while maximizing the MI between Z and Y. While there have been information bottleneck methods applicable to deep learning such as Variational Information Bottleneck (VIB) [11] or Information Dropout [4], their stochastic compressed representations may not be suitable for tasks that require consis-

tency in the representations. Also, despite the reduced amount of information in the compressed representations, the compression does not lead to practical benefits in terms of efficiency. This thesis addresses the above issues by proposing a new information bottleneck method that discretely drops unnecessary features (Chapter 2).

Less Reusable Abstraction of Behaviors. In RL, the behavior spaces of agents are often huge due to a large number of possible temporal combinations of actions. Therefore, it is important to identify and reuse behaviors that could be shared between different tasks for efficient and effective learning across tasks. Unsupervised skill discovery tackles this problem without extrinsic rewards by discovering such behaviors and learning their temporal abstractions. In this domain, *empowerment*, which is often formulated by the maximization of the mutual information (MI) $I(Z;S)$ where Z is the latent skill variable and S is the resulting state variable, is a popular direction as it desires that changes in Z induce corresponding changes in the resulting state S. However, such MI maximization is limited mainly in the following two aspects. First, it does not pursue extensive behaviors in the state space. While agents need to learn to reach various states in order to prepare for tasks in the given environment, due to MI's invariance to scaling, it does not encourage exploration in continuous state spaces, which makes it difficult to discover behaviors for diverse situations and goals. Second, simpler abstractions are not promoted. As MI is invariant to any one-to-one transformation, the learning of the mapping between behaviors in the state space and the latent skills with empowerment does not necessarily lead to simple abstractions. We deal with the limitations by linearizing the environment dynamics for intrinsic motivation to seek extensive behaviors and introducing the information bottleneck for learning disentangled abstractions (Chapter 3).

Insufficient Exploitation of Source Information for Transfer. A common objective of transfer learning in RL is to leverage information learned from source tasks to improve the performance on target tasks where the tasks differ in their reward functions. One of the representative approaches to this is transfer with the combination of the successor features (SFs) framework and generalized policy improvement (GPI) [14]. During the training stage, it aims at learning the optimal behaviors on source tasks and abstracting them as SFs, which are representations of value functions linearly decoupled from reward functions. For inference on target tasks, it combines the SFs on the source tasks with GPI, a generalization of policy improvement for multiple policies. There are also variants of SFs that exploit the smoothness of optimal value functions with respect to tasks by employing task-aware function approximators [21, 73]. Although the SFs framework and its variants have shown promising results in transfer learning, such function approximators might produce highly erroneous outputs for distant target tasks. In this thesis, we propose to use approximations for the source tasks, which are more reliable, to bound the errors on the target tasks and improve the resulting performance (Chapter 4).

1.1 Contributions

The main contributions of this thesis are summarized as follows.

- **Robust and Efficient Feature Abstraction with Discrete Information Bottleneck (Chapter 2).** To establish feature abstractions that are robust to noise and task-irrelevant information, we propose a discrete information bottleneck method named *Drop-Bottleneck*. Within the information bottleneck framework, it learns a compressed representation of

the input by discretely dropping each feature with a learned drop probability, which can be jointly trained with the feature extractor. Each drop probability is trained to take the corresponding feature dimension's importance in solving the tasks into account. Furthermore, we also define its *deterministic* compressed representations, which provide complete consistency and improved efficiency for inference while maintaining the majority of the ability to distill task-relevant information. Drop-Bottleneck can be adopted with its simple objective and empirically shows its greatly improved robustness to adversarial examples compared to Variational Information Bottleneck (VIB) [11].

This work is published in:

[61] **Jaekyeom Kim**, Minjung Kim, Dongyeon Woo, and Gunhee Kim. *Drop-Bottleneck: Learning Discrete Compressed Representation for Noise-Robust Exploration.* **ICLR 2021**.

- **Disentangled Temporal Abstraction for Reusable Skills (Chapter 3).** We combat the aforementioned issues with empowerment-based skill discovery as described below. To encourage extensive behaviors in the state space, we first suggest treating the challenge of making transitions in the state space separately from skill discovery. We simplify the environment dynamics by equipping it with a *linearizer*, a lowest-level policy that is responsible for mapping desired goal directions to the corresponding changes in the state space, and the linearizer benefits not only ours but also other skill discovery methods in the state space coverage. On top of the linearizer, we perform skill discovery by learning to abstract behaviors in the state space into the latent skill space with the information bottleneck to seek simpler abstractions due to the encouragement of

disentanglement. Having simpler temporal abstractions is beneficial for not only meta-controllers that learn to pick and use skills for downstream tasks but also for interpretability. We name our skill discovery method, which combines the two above *Information Bottleneck Option Learning (IBOL)*.

This work is published in:

[62] **Jaekyeom Kim***, Seohong Park*, and Gunhee Kim. *Unsupervised Skill Discovery with Bottleneck Option Learning*. **ICML 2021** (*: equal contribution).

- **Test-Time Improvement with Source Approximation (Chapter 4).** We take the following approach to better exploit the source information for the transfer of SFs between tasks with different reward functions. First of all, we present a theorem on lower- and upper-bounding optimal values for tasks that are expressible as linear combinations of source tasks, where the bounds are computed with approximations for the source tasks. We then introduce *constrained GPI*, which could improve the performance on new tasks by bounding optimal value approximations on the tasks solely at the inference time, based on our theorem. We empirically examine constrained GPI with different environments and settings and show that the bounding can improve the performance on target tasks.

  This work is published in:

  [63] **Jaekyeom Kim**, Seohong Park, and Gunhee Kim. *Constrained GPI for Zero-Shot Transfer in Reinforcement Learning*. **NeurIPS 2022**.

1.2 Thesis Organization

The organization of this thesis is outlined as follows. In Chapters 2–4, we discuss three main topics of this thesis with their challenges and our approaches: robust and efficient feature abstraction with discrete information bottleneck (Chapter 2), disentangled temporal abstraction for reusable skills (Chapter 3) and test-time improvement with source approximation (Chapter 4).

Chapter 2

Robust and Efficient Feature Abstraction with Discrete Information Bottleneck

2.1 Overview

Data with noise or task-irrelevant information easily harm the training of a model; for instance, the noisy-TV problem [23] is one of well-known such phenomena in reinforcement learning. If observations from the environment are modified to contain a TV screen, which changes its channel randomly based on the agent's actions, the performance of curiosity-based exploration methods dramatically degrades [23, 24, 64, 92].

The information bottleneck (IB) theory [105, 106] provides a framework for dealing with such task-irrelevant information, and has been actively adopted to exploration in reinforcement learning [53, 64]. For an input variable X and a target variable Y, the IB theory introduces another variable Z, which is a

compressed representation of X. The IB objective trains Z to contain less information about X but more information about Y as possible, where the two are quantified by mutual information terms of $I(Z;X)$ and $I(Z;Y)$, respectively. IB methods such as Variational Information Bottleneck (VIB) [11, 27] and Information Dropout [4] show that the compression of the input variable X can be done by neural networks.

In this work, we propose a novel information bottleneck method named *Drop-Bottleneck* that compresses the input variable by discretely dropping a subset of its input features that are irrelevant to the target variable. Drop-Bottleneck provides some nice properties as follows:

- The compression term of Drop-Bottleneck's objective is simple and is optimized as a tractable solution.
- Drop-Bottleneck provides a *deterministic* compressed representation that still maintains majority of the learned indistinguishability *i.e.* compression. It is useful for inference tasks that require the input representation to be consistent and stable.
- Drop-Bottleneck jointly trains a feature extractor and performs feature selection, as it learns the feature-wise drop probability taking into account each feature dimension's relevance to the target task. Hence, unlike the compression provided by most neural network-based IB methods, our deterministic representation *reduces* the feature dimensionality, which makes the following inference better efficient with less amount of data.
- Compared to VIB, both of Drop-Bottleneck's original (stochastic) and deterministic compressed representations can greatly improve the robustness to adversarial examples.

Based on the newly proposed Drop-Bottleneck, we design an exploration method that is robust against noisy observations in very sparse reward environments for reinforcement learning. Our exploration maintains an episodic memory and generates intrinsic rewards based on the predictability of new observations from the compressed representations of the ones in the memory. As a result, our method achieves state-of-the-art performance on multiple environments of VizDoom [59] and DMLab [18]. We also show that combining our exploration method with VIB instead of Drop-Bottleneck degrades the performance by meaningful margins.

Additionally, we empirically compare with VIB to show Drop-Bottleneck's superior robustness to adversarial examples and ability to reduce feature dimensionality for inference with ImageNet dataset [91]. We also demonstrate that Drop-Bottleneck's deterministic representation can be a reasonable replacement for its original representation in terms of the learned indistinguishability, with Occluded CIFAR dataset [4].

2.2 Related Work

Information bottleneck methods. There have been a number of IB methods that are approximations or special forms of the original IB objective. Variational Information Bottleneck (VIB) [11] approximates the original IB objective by establishing variational bounds on the compression and prediction terms. Chalk et al. [27] propose the same variational bound on the IB objective in the context of sparse coding tasks. Conditional Entropy Bottleneck (CEB) and Variational Conditional Entropy Bottleneck (VCEB) [38, 39] use an alternative form of the original IB objective derived under the Minimum Necessary Information (MNI) criterion to preserve only a necessary amount of information. The IB theory

[106] has been used for various problems that require restriction of information or dealing with task-irrelevant information. Information Dropout [4] relates the IB principle to multiple practices in deep learning, including Dropout, disentanglement and variational autoencoding. Moyer et al. [78] obtain representations invariant to specific factors under the variational autoencoder (VAE) [66] and VIB frameworks. Amjad and Geiger [13] discuss the use of IB theory for classification tasks from a theoretical point of view. Dai et al. [30] employ IB theory for compressing neural networks by pruning neurons in networks. Schulz et al. [96] propose an attribution method that determines each input feature's importance by enforcing compression of the input variable via the IB framework.

Similar to our goal, some previous research has proposed variants of the original IB objective. Deterministic information bottleneck (DIB) [102] replaces the compression term with an entropy term and solves the new objective using a deterministic encoder. Nonlinear information bottleneck (NIB) [67] modifies the IB objective by squaring the compression term and uses a non-parametric upper bound on the compression term. While DIB is always in the deterministic form, we can flexibly choose the stochastic one for training and the deterministic one for test. Compared to NIB, which is more computationally demanding than VIB due to its non-parametric upper bound, our method is faster.

Reinforcement learning with information bottleneck methods. The IB theory has been applied to several reinforcement learning (RL) tasks. Variational discriminator bottleneck [88] regulates the discriminator's accuracy using the IB objective to improve adversarial training, and use it for imitation learning. Information Bottleneck Actor Critic [53] employs VIB to make the features generalize better and encourage the compression of states as input to the actor-critic algorithm. Curiosity-Bottleneck [64] employs the VIB framework to train

a compressor that compresses the representation of states, which is still informative about the value function, and uses the compressiveness as exploration signals. InfoBot [43] proposes a conditional version of VIB to improve the transferability of a goal-conditioned policy by minimizing the policy's dependence on the goal. Variational bandwidth bottleneck [44] uses a modified, conditional version of VIB, and solves RL tasks with privileged inputs (*i.e.* valuable information that comes with a cost). Our exploration method differs from these methods in two aspects. First, we propose a new information bottleneck method that is not limited to exploration in RL but generally applicable to the problems for which the IB theory is used. Second, our method generates exploration signals based on *the noise-robust predictability i.e.* the predictability between noise-robust representations of observations.

2.3 Feature Abstraction with Drop-Bottleneck

2.3.1 Preliminaries of Information Bottleneck

Given an input random variable X, the information bottleneck (IB) framework [105, 106] formalizes a problem of obtaining X's compressed representation Z, which still and only preserves information relevant to the target variable Y. Its objective function is

$$\text{minimize} -I(Z;Y) + \beta I(Z;X), \tag{2.1}$$

where β is a Lagrangian multiplier. The first and second terms are prediction and compression terms, respectively. The prediction term $I(Z;Y)$ encourages Z to preserve task-relevant information while the compression term $I(Z;X)$ compresses the input information as much as possible.

As reviewed in the previous section, there have been several IB methods

[4, 11, 27, 67, 102], among which the ones derived using variational inference such as Variational Information Bottleneck (VIB) [11] have become dominant due to its applicability to general problems.

2.3.2 Drop-Bottleneck

We propose a novel information bottleneck method called *Drop-Bottleneck* (DB), where the input information is compressed by *discretely* dropping a subset of input features. Thus, its compression objective is simple and easy to optimize. Moreover, its representation is easily convertible to a deterministic version for inference tasks (Section 2.3.3), and it allows joint training with a feature extractor (Section 2.3.4). While discrete dropping of features has been explored by prior works including Dropout [101], DB differs in that its goal is to assign different drop probabilities to feature variables based on their relevance to the target variable.

For an input variable $X = [X_1, \ldots, X_d]$ and a drop probability $\boldsymbol{p} = [p_1, \ldots, p_d] \in [0, 1]^d$, we define its compressed representation as $Z = C_{\boldsymbol{p}}(X) = [c(X_1, p_1), \ldots, c(X_d, p_d)]$ such that

$$c(X_i, p_i) = b \cdot \text{Bernoulli}(1 - p_i) \cdot X_i, \quad \text{where} \quad b = \frac{d}{d - \sum_k p_k}, \tag{2.2}$$

for $i = 1, \ldots, d$. That is, the compression procedure *drops* the i-th input feature (*i.e.* replaced by zero) with probability p_i. Since the drop probability is to be learned, we use a scaling factor b to keep the scale of Z constant. We use a single scaling factor for all feature dimensions in order to preserve the relative scales between the features.

With the definition in Equation (2.2), we derive the compression term of

DB to minimize as

$$I(Z;X) = \sum_{i=1}^{d} I(Z_i; X_1, \ldots, X_d | Z_1, \ldots, Z_{i-1}) \tag{2.3}$$

$$= \sum_{i=1}^{d} [I(Z_i; X_i | Z_1, \ldots, Z_{i-1}) + I(Z_i; X_1, \ldots, X_d \setminus X_i | Z_1, \ldots, Z_{i-1}, X_i)]$$

$$= \sum_{i=1}^{d} I(Z_i; X_i | Z_1, \ldots, Z_{i-1}) \leq \sum_{i=1}^{d} I(Z_i; X_i) = \hat{I}(Z;X) \tag{2.4}$$

using the properties that $Z_i \perp\!\!\!\perp X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_d | Z_1, \ldots, Z_{i-1}, X_i$ and $Z_i \perp\!\!\!\perp Z_1, \ldots, Z_{i-1} | X_i$. Note that $\hat{I}(Z;X) - I(Z;X) = \left(\sum_{i=1}^{d} H(Z_i)\right) - H(Z_1, \ldots, Z_d) = TC(Z)$ where $TC(Z)$ is the total correlation of Z and $H(\cdot)$ denotes the entropy, and $\hat{I}(Z;X) = I(Z;X)$ if $X_1, \ldots, X_d$ are independent. To provide a rough view on the gap, due to the joint optimization with the compression term $\hat{I}(Z;X)$ and the prediction term $I(Z;Y)$, Z becomes likely to preserve less redundant and less correlated features, and $TC(Z)$ could decrease as the optimization progresses.

Finally, DB's new compression term, $\hat{I}(Z;X)$, is computed as

$$\hat{I}(Z;X) = \sum_{i=1}^{d} I(Z_i; X_i) = \sum_{i=1}^{d} \big(H(X_i) - H(X_i|Z_i)\big) \tag{2.5}$$

$$= \sum_{i=1}^{d} \big(H(X_i) - p_i \cdot H(X_i|Z_i = 0) - (1 - p_i) \cdot H(X_i|Z_i = bX_i)\big) \tag{2.6}$$

$$\approx \sum_{i=1}^{d} \big(H(X_i) - p_i \cdot H(X_i) - (1 - p_i) \cdot 0\big) = \sum_{i=1}^{d} H(X_i)(1 - p_i). \tag{2.7}$$

From Equation (2.6) to Equation (2.7), we use the two ideas: (i) $H(X_i|Z_i = 0) = H(X_i)$ because $Z_i = 0$ means it contains no information about X_i, and (ii) $H(X_i|Z_i = bX_i) = 0$ because $Z_i = bX_i$ means Z_i preserves the feature (*i.e.* Z_i fully identifies X_i) and thus their conditional entropy becomes zero. Importantly, DB's compression term is computed as the simple tractable expression in

Equation (2.7). As the goal of the compression term is to penalize $I(Z;X)$ not $H(X)$, the drop probability $\boldsymbol{p}$ is the only parameter optimized with our compression term. Thus, each $H(X_i)$ can be computed with any entropy estimation method such as the binning-based estimation, which involves quantization for continuous X_i, since the computation has no need to be differentiable.

However, one issue of Equation (2.7) is that Z is not differentiable with respect to $\boldsymbol{p}$ as Bernoulli distributions are not differentiable. We thus use the Concrete relaxation [55, 74] of the Bernoulli distribution to update $\boldsymbol{p}$ via gradients from Z:

$$\text{Bernoulli}(p) \approx \sigma\left(\frac{1}{\lambda}\left(\log p - \log(1-p) + \log u - \log(1-u)\right)\right), \tag{2.8}$$

where $u \sim \text{Uniform}(0,1)$ and λ is a temperature for the Concrete distribution. Intuitively, $\boldsymbol{p}$ is trained to assign a high drop probability to the feature that is irrelevant to or redundant for predicting the target variable Y.

2.3.3 Deterministic Compressed Representation

With Drop-Bottleneck, one can simply obtain the deterministic version of the compressed representation as $\bar{Z} = \bar{C}_{\boldsymbol{p}}(X) = [\bar{c}(X_1, p_1), \ldots, \bar{c}(X_d, p_d)]$ for

$$\bar{c}(X_i, p_i) = \bar{b} \cdot \mathbb{1}(p_i < 0.5) \cdot X_i, \quad \text{where} \quad \bar{b} = \frac{d}{\sum_k \mathbb{1}(p_k < 0.5)}. \tag{2.9}$$

$\bar{b}$ is defined similarly to b with a minor exception that the scaling is done based on the actual, deterministic number of the dropped features. The deterministic compressed representation $\bar{Z}$ is useful for inference tasks that require stability or consistency of the representation as well as reducing the feature dimensionality for inference, as we demonstrate in Section 2.5.4.

2.3.4 Training with Drop-Bottleneck

We present how Drop-Bottleneck (DB) is trained with the full IB objective allowing joint training with a feature extractor. Since DB proposes only a new compression term, any existing method for maximizing the prediction term $I(Z;Y)$ can be adopted. We below discuss an example with Deep Infomax [50] since our exploration method uses this framework (Section 2.4). Deep Infomax, instead of $I(Z;Y)$, maximizes its Jensen-Shannon mutual information estimator

$$I_{\psi}^{\mathrm{JSD}}(Z;Y) = \frac{1}{2}\left(\mathbb{E}_{\mathbb{P}_{ZY}}[-\zeta(-T_{\psi}(Z,Y))] - \mathbb{E}_{\mathbb{P}_{Z}\otimes\mathbb{P}_{Y}}[\zeta(T_{\psi}(Z,\tilde{Y}))] + \log 4\right),$$

where T_{ψ} is a discriminator network with parameter ψ and $\zeta(\cdot)$ is the softplus function.

Finally, the IB objective with Drop-Bottleneck becomes

$$\text{minimize} -I_{\psi}^{\mathrm{JSD}}(Z;Y) + \beta \sum_{i=1}^{d} H(X_i)(1-p_i), \tag{2.10}$$

which can be optimized via gradient descent. To make $\boldsymbol{p}$ more freely trainable, we suggest simple element-wise parameterization of $\boldsymbol{p}$ as $p_i = \sigma(p_i')$ and initializing $p_i' \sim \mathrm{Uniform}(a,b)$. We obtain the input variable X from a feature extractor, whose parameters are trained via the prediction term, jointly with $\boldsymbol{p}$ and ψ. In next section, we will discuss its application to hard exploration problems for reinforcement learning.

2.4 Robust Exploration with Drop-Bottleneck

Based on DB, we propose an exploration method that is robust against highly noisy observations in a very sparse reward environment for reinforcement learning tasks. We first define a parametric feature extractor f_{ϕ} that maps a state to

X. For transitions (S, A, S') where S, A and S' are current states, actions and next states, respectively, we use the DB objective with

$$X = f_\phi(S'), \quad Z = C_{\boldsymbol{p}}(X), \quad Y = C_{\boldsymbol{p}}(f_\phi(S)). \tag{2.11}$$

For every transition (S, A, S'), the compression term minimizes $I(Z; X) = I(C_{\boldsymbol{p}}(f_\phi(S')); f_\phi(S'))$ and encourages $C_{\boldsymbol{p}}$ to drop unnecessary features of the next state embedding $f_\phi(S')$ as possible. The prediction term maximizes $I(Z; Y) = I(C_{\boldsymbol{p}}(f_\phi(S')); C_{\boldsymbol{p}}(f_\phi(S)))$ and makes the compressed representations of the current and next state embeddings, $C_{\boldsymbol{p}}(f_\phi(S))$ and $C_{\boldsymbol{p}}(f_\phi(S'))$, informative about each other.

Applying Equation (2.11) to Equation (2.10), the Drop-Bottleneck objective for exploration becomes

$$\text{minimize} -I_\psi^{\text{JSD}}(C_{\boldsymbol{p}}(f_\phi(S')); C_{\boldsymbol{p}}(f_\phi(S))) + \beta \sum_{i=1}^{d} H((f_\phi(S'))_i)(1 - p_i). \tag{2.12}$$

While f_ϕ, $\boldsymbol{p}$, and T_ψ are being trained online, we use them for the agent's exploration with the help of episodic memory inspired by Savinov et al. [92]. Starting from an empty episodic memory M, we add the learned feature of the observation at each step. For example, at time step t, the episodic memory is $M = \{\bar{C}_{\boldsymbol{p}}(f_\phi(s_1)), \bar{C}_{\boldsymbol{p}}(f_\phi(s_2)), \ldots, \bar{C}_{\boldsymbol{p}}(f_\phi(s_{t-1}))\}$ where $s_1, \ldots, s_{t-1}$ are the earlier observations from that episode. We then quantify the degree of novelty of a new observation as an intrinsic reward. Specifically, the intrinsic reward for s_t is computed utilizing the Deep Infomax discriminator T_ψ, which is trained to output a large value for joint (or likely) input and a small value for marginal (or arbitrary) input:

$$r^{\text{i}}_{M,t}(s_t) = \frac{1}{t-1} \sum_{j=1}^{t-1} [g(s_t, s_j) + g(s_j, s_t)] \tag{2.13}$$

$$\text{for } g(x, y) = \zeta(-T_\psi(\bar{C}_{\boldsymbol{p}}(f_\phi(x)), \bar{C}_{\boldsymbol{p}}(f_\phi(y))),$$

where $g(s_t, s_j)$ and $g(s_j, s_t)$ denote the unlikeliness of s_t being the next and the previous state of s_j, respectively. Thus, intuitively, for s_t that is close to a region covered by the earlier observations in the state space, $r^{\mathrm{i}}_{M,t}(s_t)$ becomes low, and vice versa. It results in a solid exploration method capable of handling noisy environments with very sparse rewards. For computing the intrinsic reward, we use the DB's deterministic compressed representation of states to provide stable exploration signals to the policy optimization.

2.5 Experiments

We carry out various types of experiments to evaluate Drop-Bottleneck (DB) in multiple aspects. First, we apply DB exploration to multiple VizDoom [59] and DMLab [18] environments with three hard noise settings, where we compare with state-of-the-art methods as well as its VIB variant (Sections 2.5.2 and 2.5.3). Second, we empirically show that DB is superior to VIB and the mutual information-based feature selection for adversarial robustness and feature dimensionality reduction in ImageNet classification [91] (Section 2.5.4), and we juxtapose DB with VCEB, which employs a different form of the IB object, for the same adversarial robustness test (Section 2.5.5). Third, we make another comparison with VIB in terms of the removal of task-irrelevant information and the validity of the deterministic compressed representation (Section 2.5.6) and provide the visualization of the task-irrelevant information removal on the same task (Section 2.5.7). Finally, we perform an ablation study on the effect of DB on the exploration performance (Section 2.5.8). For all the experiments with DB, each entropy $H(X_i)$ is computed with the binning-based estimation using 32 bins, and the drop probability $\boldsymbol{p}$ is initialized with $p_i = \sigma(p'_i)$ and $p'_i \sim \mathrm{Uniform}(a, b)$ for $a = -2, b = 1$.

2.5.1 Experimental Setup for Exploration Tasks

Basic setup and baselines. For the exploration tasks on VizDoom [59] and DMLab [18], we use the proximal policy optimization (PPO) algorithm [95] as the base RL method. We employ ICM from Pathak et al. [87], and EC and ECO from Savinov et al. [92] as baseline exploration methods. ICM learns a dynamics model of the environment and uses the prediction errors as exploration signals for the agent. EC and ECO are curiosity methods that use episodic memory to produce novelty bonuses according to the reachability of new observations, and show the state-of-the-art exploration performance on VizDoom and DMLab navigation tasks. In summary, we compare with four baseline methods: PPO, PPO + ICM, PPO + EC, and PPO + ECO. Additionally, we report the performance of our method combined with VIB instead of DB.

We conduct experiments with three versions of noisy-TV settings named "Image Action", "Noise" and "Noise Action", as proposed in Savinov et al. [92]. Following Savinov et al. [92], we resize observations as 160×120 only for the episodic curiosity module while as 84×84 for the PPO policy and exploration methods. We use the official source code[1] of Savinov et al. [92] to implement and configure the baselines (ICM, EC and ECO) and the three noise settings.

For the feature extractor f_ϕ, we use the same CNN with the policy network of PPO from Mnih et al. [77]. The only modification is to use $d = 128$ *i.e.* 128-dimensional features instead of 512 to make features lightweight enough to be stored in the episodic memory. The Deep Infomax discriminator T_ψ consists of three FC hidden layers with $64, 32, 16$ units each, followed by a final FC layer for prediction. We initialize the drop probability $\boldsymbol{p}$ with $p_i = \sigma(p'_i)$ and $p'_i \sim \text{Uniform}(a, b)$ where $a = -2, b = 1$. We collect samples and update

[1] `https://github.com/google-research/episodic-curiosity`.

$\boldsymbol{p}, T_\psi, f_\phi$ with Equation (2.12) every 10.8K and 21.6K time steps in VizDoom and DMLab, respectively, with a batch size of 512. We duplicate the compressed representation 50 times with differently sampled drop masks, which help better training of the feature extractor, the drop probability and the discriminator by forming diverse subsets of features.

Implementation details. We describe additional details of the VizDoom [59] and DMLab [18] exploration experiments. We collect training samples in a buffer and update $\boldsymbol{p}, T_\psi, f_\phi$ with Equation (2.12) periodically. We use Adam optimizer [65] with a learning rate of 0.0001 and a batch size of 512. In each optimization epoch, the training samples from the buffer are re-shuffled. For each mini-batch, we optimize the Deep Infomax [50] discriminator T_ψ with 8 extra epochs with the same samples, to make the Jensen-Shannon mutual information bound tighter. This way of training T_ψ only runs forward and backward passes on T_ψ for the fixed output of the feature extractor f_ϕ, and thus can be done with low computational cost. β is the hyperparameter that determines the relative scales of the compression term and the Deep Infomax Jensen-Shannon mutual information estimator. It is tuned to $\beta = 0.001/128$ for DB and $\beta = 0.0005/128$ for VIB. To make experiments simpler, we normalize our intrinsic rewards with the running mean and standard deviation.

Table 2.1 and Table 2.2 report the hyperparameters of the methods for VizDoom and DMLab experiments, respectively. We tune the hyperparameters based on the ones provided by Savinov et al. [92]. Unless specified, we use the same hyperparameters with Savinov et al. [92].

Under the three noise settings suggested in Savinov et al. [92], the lower right quadrant of every observation is occupied by a TV screen as follows.

- "Image Action": Every time the agent performs a specific action, it

Table 2.1: Hyperparameters of PPO [95], PPO + ICM [87], PPO + ECO [92], and PPO + Ours for the VizDoom experiments.

	PPO	PPO + ICM	PPO + ECO	PPO + Ours
PPO				
Learning rate	0.00025	0.00025	0.00025	0.00025
Entropy coefficient	0.01	0.01	0.01	0.01
Task reward scale	5	5	5	5
Exploration method				
Training period and sample size	–	3K	120K	10.8K
# of optimization epochs	–	4	10	4
Intrinsic bonus scale	–	0.01	1	0.001

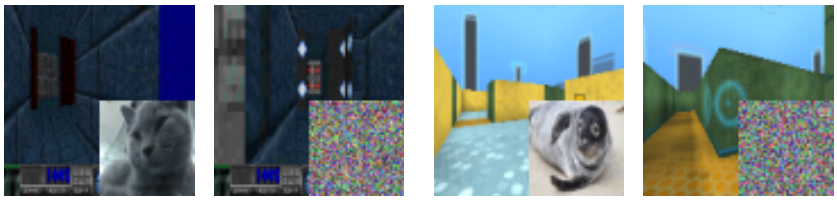
Table 2.2: Hyperparameters of PPO [95], PPO + ICM [87], PPO + EC/ECO [92], and PPO + Ours for the DMLab experiments.

	PPO	PPO + ICM	PPO + EC/ECO	PPO + Ours
PPO				
Learning rate	0.00019	0.00025	0.00025	0.00025
Entropy coefficient	0.0011	0.0042	0.0021	0.0011
Task reward scale	1	1	1	1
Exploration method				
Training period and sample size	–	3K	720K (ECO)	21.6K
# of optimization epochs	–	4	10 (ECO)	2
Intrinsic bonus scale	–	0.55	0.030	0.005

changes the channel of the TV randomly to one of the 30 predefined animal images.

- "Noise": At every observation, a new noise pattern is sampled and shown on the TV screen (independently from the agent's actions).
- "Noise Action": Same as "Noise", but the noise pattern only changes when the agent does a specific action.

Figure 2.1 shows some observation examples from VizDoom and DMLab environments with "Image Action" and "Noise" settings.



(a) VizDoom (b) DMLab

Figure 2.1: Example observations from VizDoom [59] and DMLab [18] environments with "Image Action" (first) and "Noise" (second) settings.

2.5.2 Exploration in Noisy Static Maze Environments

VizDoom [59] provides a static 3D maze environment. We experiment on the *MyWayHome* task with nine different settings by combining three reward conditions ("Dense", "Sparse" and "Very Sparse") in Pathak et al. [87] and three noise settings in the previous section. In the "Dense", "Sparse" and "Very Sparse" cases, the agent is randomly spawned in a near, medium and very distant room, respectively. Thus, "Dense" is a relatively easy task for the agent to reach the goal even with a short random walk, while "Sparse" and "Very Sparse" require the agent to perform a series of directed actions, which make the goal-oriented navigation difficult.

Table 2.3 and Figure 2.2 compare the DB exploration with three baselines, PPO, PPO + ICM, and PPO + ECO on the VizDoom tasks, in terms of the final performance and how quickly they learn. The results suggest that even in the static maze environments, the three noise settings can degrade the performance of the exploration by large margins. On the other hand, our method with DB shows robustness to such noise or task-irrelevant information, and outperforms the baselines in all the nine challenging tasks, whereas our method combined with VIB does not exhibit competitive results.

Table 2.3: Comparison of the average episodic sum of rewards in VizDoom (over 10 runs) and DMLab (over 30 runs) under three noise settings: Image Action (IA), Noise (N) and Noise Action (NA). The values are measured after 6M and 20M (4 action-repeated) steps for VizDoom and DMLab respectively, with no seed tuning done. Baseline results for DMLab are cited from Savinov et al. [92]. Grid Oracle† provides the performance upper bound by the oracle method for DMLab tasks.

Method	VizDoom									DMLab					
	Dense			Sparse			Very Sparse			Sparse			Very Sparse		
	IA	N	NA	IA	N	NA	IA	N	NA	IA	N	NA	IA	N	NA
PPO [95]	**1.00**	**1.00**	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00	8.5	11.6	9.8	6.3	8.7	6.1
PPO + ICM [87]	0.87	**1.00**	**1.00**	0.00	0.50	0.40	0.00	0.73	0.20	6.9	7.7	7.6	4.9	6.0	5.7
PPO + EC [92]	–	–	–	–	–	–	–	–	–	13.1	18.7	14.8	7.4	13.4	11.3
PPO + ECO [92]	0.71	0.81	0.72	0.21	0.70	0.33	0.19	0.79	0.50	18.5	28.2	18.9	16.8	26.0	12.5
PPO + Ours (VIB)	**1.00**	**1.00**	**1.00**	0.21	0.61	0.40	0.37	0.70	0.67	28.2	31.9	27.1	23.5	25.4	22.3
PPO + Ours (DB)	**1.00**	**1.00**	**1.00**	**0.90**	**1.00**	**0.99**	**0.90**	**1.00**	**0.90**	**30.4**	**32.7**	**30.6**	**28.8**	**29.1**	**26.9**
PPO + Grid Oracle†	–	–	–	–	–	–	–	–	–	37.4	38.8	39.3	36.3	35.5	35.4

2.5.3 Exploration in Noisy and Randomly Generated Maze Environments

As a more challenging exploration task, we employ DMLab [92], which are general and randomly generated maze environments where at the beginning of every episode, each maze is procedurally generated with its random goal location. Thanks to the random map generator, each method is evaluated on test mazes that are independent of training mazes. As done in Savinov et al. [92], we test with six settings according to two reward conditions ("Sparse" and "Very Sparse") and the three noise settings. In the "Sparse" scenario, the agent is (re-)spawned at a random location when each episode begins or every time it reaches the goal *i.e.* the sparse reward; the agent should reach the goal as many times as possible within the fixed episode length. The "Very Sparse" is its harder version: the agent does not get (re-)spawned near or in the same room with the goal.

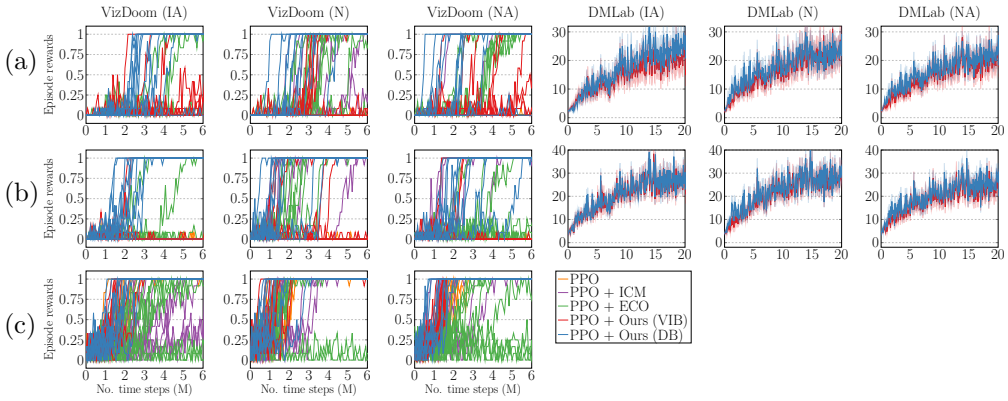


Figure 2.2: Reward trajectories as a function of training step (in millions) for VizDoom (columns 1-3) and DMLab (columns 4-6) with (a) Very Sparse, (b) Sparse and (c) Dense settings. For VizDoom tasks, we show all the 10 runs per method. For DMLab tasks, we show the averaged episode rewards over 30 runs of our exploration with the 95% confidence intervals.

Table 2.3 compares between different exploration methods on the DMLab tasks. The results demonstrate that our DB exploration method achieves state-of-the-art performance with significant margins from the baselines on all the 6 tasks, and performs better than our method equipped with VIB as well. The plots too suggest that our method provides stable exploration signals to the agent under different environmental and noise settings. As mentioned in Section 2.5.1, our method can achieve better performance even using much lower resolution observations of 84×84 than 160×120 of EC and ECO. Also, excluding the policy network, our method maintains 0.5M parameters, which is significantly smaller compared to ECO with 13M parameters. Please refer to Section 2.5.8 for an ablation study,

Figure 2.3 shows evolution examples of the drop probability distribution over training time steps. It overviews the role of drop probability $\boldsymbol{p}$ in DB. As the joint training of the feature extractor f_ϕ with $\boldsymbol{p}$ progresses, $\boldsymbol{p}$ gets separated into

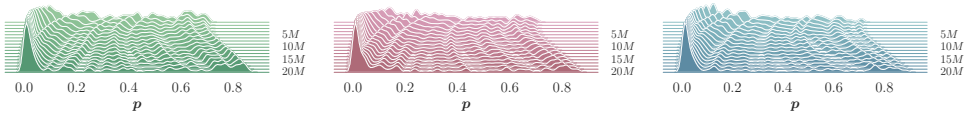


Figure 2.3: Evolution examples of the drop probability distribution $\boldsymbol{p}$ on Very Sparse DMLab environments with (left) Image Action, (middle) Noise and (right) Noise Action settings. Each figure shows a histogram per $\boldsymbol{p}$ value according to training iterations (the more front is the more recent).

high- and low-value groups, where the former drops task-irrelevant or redundant features and the latter preserves task-relevant features. This suggests that in the DMLab environments, the DB objective of Equation (2.12) successfully encourages dropping the features unrelated to transition between observations and also the deterministic compressed representation becomes stable as the training progresses.

2.5.4 Comparison with VIB: Adversarial Robustness & Dimension Reduction

We experiment with image classification on ImageNet [91] to compare Drop-Bottleneck (DB) with Variational Information Bottleneck (VIB) [11], the most widely-used IB framework, regarding the robustness to adversarial attacks and the reduction of feature dimensionality. We follow the experimental setup from Alemi et al. [11] with some exceptions. We use $\beta_1 = 0.9$ and no learning rate decay for DB's Adam optimizer [65]. For prediction, we use one Monte Carlo sample of each stochastic representation. Additionally, we provide a similar comparison with the mutual information-based feature selection method.

Robustness to adversarial attacks. Following Alemi et al. [11], we employ the targeted ℓ_2 and ℓ_∞ adversarial attacks from Carlini and Wagner [26]. For each method, we determine the first 200 validation images on ImageNet

Table 2.4: Results of the adversarial robustness for Drop-Bottleneck (DB) and Variational Information Bottleneck (VIB) [11] with the targeted ℓ_2 and ℓ_∞ attacks [26]. *Succ.* denotes the attack success rate in % (lower is better), and *Dist.* is the average perturbation distance over successful adversarial examples (higher is better).

Attack type	β	VIB		DB		DB (determ.)	
		Succ.	Dist.	Succ.	Dist.	Succ.	Dist.
ℓ_2	0.0001	99.5	0.806	100.0	0.929	99.5	0.923
	0.0003162	99.5	0.751	100.0	0.944	100.0	0.941
	0.001	100.0	0.796	99.5	1.097	100.0	1.134
	0.003162	99.5	0.842	27.0	3.434	23.0	2.565
	0.01	100.0	0.936	18.5	6.847	20.0	6.551
	0.03162	100.0	0.695	41.0	2.160	39.5	1.953
	0.1	99.5	0.874	85.5	2.850	85.5	2.348
ℓ_∞	0.0001	99.5	0.015	91.0	0.013	95.5	0.009
	0.0003162	99.5	0.017	85.0	0.016	91.5	0.009
	0.001	100.0	0.017	62.5	0.020	70.0	0.012
	0.003162	97.5	0.017	1.5	0.009	1.5	0.020
	0.01	87.0	0.019	2.0	0.022	2.0	0.013
	0.03162	25.0	0.121	8.5	0.022	8.0	0.023
	0.1	15.5	0.202	23.0	0.017	23.0	0.019

that are classified correctly, and apply the attacks to them by selecting uniformly random attack target classes. Each input image is sized as $299 \times 299 \times 3$ and each pixel value is ranged $[-1, 1]$. We perform our adversarial robustness experiments based on the official source code[2] of Carlini and Wagner [26]. For the targeted ℓ_2 attack we increase the number of binary search steps to 20 and use a batch size of 25. Other than the two, the default hyperparameters are used.

Table 2.4 shows the results. For the targeted ℓ_2 attacks, choosing the value of β from $[0.003162, 0.1]$ provides the improved robustness of DB with the max-

[2] `https://github.com/carlini/nn_robust_attacks`.

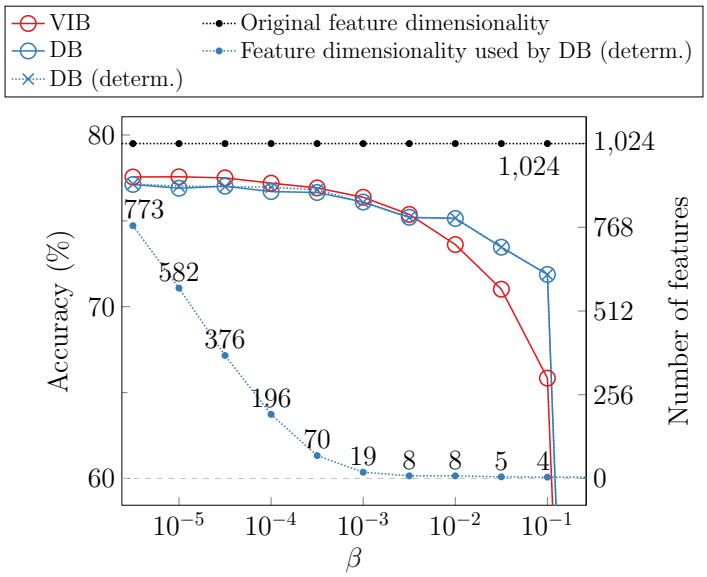


Figure 2.4: Classification accuracy of Inception-ResNet-v2 equipped with VIB [11] and DB on ImageNet validation set [91]. DB (determ.) quickly drops many feature dimensions with increased β, while VIB retains them at 1024 regardless of β.

imum at $\beta = 0.01$. On the other hand, VIB has no improved robustness in all ranges of β. For the targeted ℓ_∞ attacks, DB can reduce the attack success rate even near to 0% (*e.g.* $\beta = 0.003162$ or 0.01). Although VIB decreases the attack success rate to 15.5% at $\beta = 0.1$, VIB already suffers from the performance degradation at $\beta = 0.1$ compared to DB (Figure 2.4), and increasing β accelerates VIB's degradation even further. Note that the validation accuracies of both VIB and DB are close to zero at $\beta = 0.3162$.

This adversarial robustness of DB with the appropriate range of β seems intriguing. While one possible explanation is that small perturbations in the image space might not be easily mapped to enough differences in its feature space to cause changes in its predictions, investigating how exactly DB's adversarial robustness is achieved would be an interesting direction for future study.

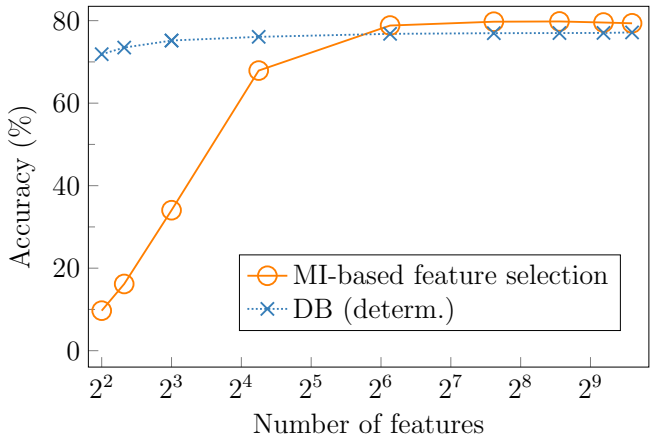


Figure 2.5: Classification accuracy of Inception-ResNet-v2 equipped with the mutual information-based feature selection and DB on ImageNet validation set [91], using the same number of features.

Dimensionality reduction. Figure 2.4 compares the accuracy of DB and VIB by varying β on the ImageNet validation set. Overall, their accuracies develop similarly with respect to β; while VIB is slightly better in the lower range of β, DB produces better accuracy in the higher range of β. Note that *DB (determ.)* shows the almost identical accuracy plot with *DB*. Importantly, *DB (determ.)* still achieves a reasonable validation accuracy ($\geq$ 75%) using only a few feature dimensions (*e.g.* 8) out of the original 1024 dimensions. This suggests that DB's deterministic compressed representation can greatly reduce the feature dimensionality for inference with only a small trade-off with the performance. It is useful for improving the efficiency of the model after the training is complete. On the other hand, VIB has no such capability. Finally, as Figure 2.4 shows, the trade-off between the dimensionality reduction and the performance can be controlled by the value of β.

Comparison with feature selection. As the deterministic representation of DB, *DB (determ.)*, provides the dimensionality reduction, we also empirically

Table 2.5: Results of the adversarial robustness for Drop-Bottleneck (DB) and the mutual information-based feature selection with the targeted ℓ_2 and ℓ_∞ attacks [26], using the same number of features. *Succ.* denotes the attack success rate in % (lower is better), and *Dist.* is the average perturbation distance over successful adversarial examples (higher is better).

Attack type	# of features	MI-based FS		DB (determ.)	
		Succ.	Dist.	Succ.	Dist.
ℓ_2	196	99.5	1.484	99.5	0.923
	70	100.0	1.323	100.0	0.941
	19	99.5	1.161	100.0	1.134
	8	99.5	1.164	20.0	6.551
	5	97.0	1.202	39.5	1.953
	4	97.0	1.127	85.5	2.348
ℓ_∞	196	99.5	0.016	95.5	0.009
	70	100.0	0.014	91.5	0.009
	19	99.5	0.013	70.0	0.012
	8	99.5	0.014	2.0	0.013
	5	97.0	0.016	8.0	0.023
	4	97.0	0.015	23.0	0.019

compare DB with the univariate mutual information-based feature selection method for obtaining the feature space with a reduced dimensionality. In the experiments, the same features provided to DB and VIB are given as input to the feature selection method as well, and for a more straightforward comparison, we let the feature selection method preserve the same number of features as *DB (determ.)*. The input of the feature selection method is the same as DB's and VIB's: the input features are obtained with a pre-trained model of Inception-ResNet-v2 on ImageNet [91]. We randomly pick $100k$ samples out of the total 1281167 training samples of ImageNet, and compute the relevance scores for features using those samples by estimating the mutual information between each feature and its label using the entropy estimation with the k-

nearest neighbors [68, 90]. Given the computed scores, we perform the feature selection by preserving the features with the highest scores.

Figure 2.5 shows the classification accuracy of the two methods for the same numbers of features. The results suggest that while the mutual information-based feature selection method could provide a marginal performance benefit when a larger subset of the pre-trained features is preserved, DB is significantly better at retaining the accuracy with a small number of feature dimensions. For instance, DB achieves the accuracy over 71% even with 4 features, but the accuracy of feature selection method drops from $\approx 68\%$ to $\approx 10\%$ when the number of features is $< 2^6$. Also, we make a comparison of the adversarial robustness; Table 2.5 suggests that the features preserved with the feature selection method show almost no robustness to the targeted ℓ_2 and ℓ_∞ attacks, where every attack success rate is $\geq 97\%$. On the other hand, *DB (determ.)* can reduce the success rate to 20% for the ℓ_2 and to 2% for the ℓ_∞ attacks with 8 features.

2.5.5 Comparison with VCEB: Adversarial Robustness

In this section, we compare Drop-Bottleneck (DB) with Variational Conditional Entropy Bottleneck (VCEB) [38, 39] on the same ImageNet [91] tasks for the adversarial robustness as in Section 2.5.4. VCEB variationally approximates the CEB objective, which is defined as

$$\text{minimize} -I(Z;Y) + \gamma I(Z;X|Y). \tag{2.14}$$

Note that Equation (2.14) is an alternative form of the original IB objective, Equation (2.1), as $I(Z;X,Y) = I(Z;Y) + I(Z;X|Y) = I(Z;X) + I(Z;Y|X)$ and $I(Z;Y|X) = 0 \quad (\because Z \perp\!\!\!\perp Y|X)$. As in Section 2.5.4, we employ the experimental setup from VIB [11] with small modifications to the hyperparameters

for the Adam optimizer [65] that $\beta_1 = 0.9$ and no learning rate decay is used. Additionally for VCEB, we apply the configurations suggested by Fischer and Alemi [39]: 1) at test time, use the mean of the Gaussian encoding instead of sampling from the distribution, and 2) reparameterize $\gamma = \exp(-\rho)$ and anneal the value of ρ from $\rho = 100$ to the final ρ during training. For our experiments, the annealing is performed via the first 100000 training steps, where each epoch consists of 6405 steps.

Employing the experimental setup from Alemi et al. [11], we adopt the targeted ℓ_2 and ℓ_∞ adversarial attacks from Carlini and Wagner [26]. We determine the first 200 correctly classified validation images on ImageNet for each setting and perform the attacks where the attack target classes are chosen randomly and uniformly.

Figure 2.6 visualizes the classification accuracy for each corresponding final value of ρ, and the adversarial robustness results are shown in Table 2.6. Overall, both VCEB and DB provide meaningful robustness to the targeted ℓ_2 and ℓ_∞ attacks. For the targeted ℓ_2 attacks, although VCEB could achieve the higher average perturbation distance over successful attacks, DB and its deterministic form show better robustness compared to VCEB in terms of the attack success rates: 18.5% (*DB* at $\beta = 0.01$) and 20.0% (*DB (determ.)* at $\beta = 0.01$) versus 45.0% (VCEB at the final $\rho = 3.454$). For the targeted ℓ_∞ attacks, DB and its deterministic version again achieve the lower attack success rates than

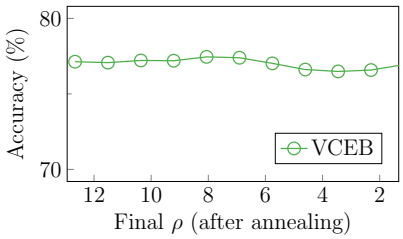


Figure 2.6: Classification accuracy of Inception-ResNet-v2 equipped with VCEB [38, 39] on ImageNet [91]. ρ is annealed from 100 to the final ρ over the first 100000 training steps.

Table 2.6: Results of the adversarial robustness for Drop-Bottleneck (DB) and Variational Conditional Entropy Bottleneck (VCEB) [38, 39] with the targeted ℓ_2 and ℓ_∞ attacks [26]. *Succ.* denotes the attack success rate in % (lower is better), and *Dist.* is the average perturbation distance over successful adversarial examples (higher is better). $^\dagger\rho$ for VCEB is annealed from 100 to the final ρ over the first 100000 training steps.

Attack type	Constraint on $I(Z;X\|Y)$			Constraint on $I(Z;X)$				
	Final $\rho^\dagger$	VCEB		β	DB		DB (determ.)	
		Succ.	Dist.		Succ.	Dist.	Succ.	Dist.
ℓ_2	9.210	99.0	1.200	0.0001	100.0	0.929	99.5	0.923
	8.059	92.0	2.028	0.0003162	100.0	0.944	100.0	0.941
	6.908	86.5	5.040	0.001	99.5	1.097	100.0	1.134
	5.757	65.0	7.198	0.003162	27.0	3.434	23.0	2.565
	4.605	46.0	12.016	0.01	18.5	6.847	20.0	6.551
	3.454	45.0	10.744	0.03162	41.0	2.160	39.5	1.953
	2.303	53.0	14.021	0.1	85.5	2.850	85.5	2.348
ℓ_∞	9.210	86.0	0.012	0.0001	91.0	0.013	95.5	0.009
	8.059	64.5	0.013	0.0003162	85.0	0.016	91.5	0.009
	6.908	48.0	0.016	0.001	62.5	0.020	70.0	0.012
	5.757	29.0	0.019	0.003162	1.5	0.009	1.5	0.020
	4.605	17.0	0.025	0.01	2.0	0.022	2.0	0.013
	3.454	12.5	0.025	0.03162	8.5	0.022	8.0	0.023
	2.303	17.5	0.027	0.1	23.0	0.017	23.0	0.019

VCEB: 1.5% and 2.0% (*DB* and *DB (determ.)* at $\beta \in \{0.003162, 0.01\}$)) versus 12.5% (VCEB at the final $\rho = 3.454$).

2.5.6 Removal of Task-irrelevant Information and Validity of Deterministic Compressed Representation

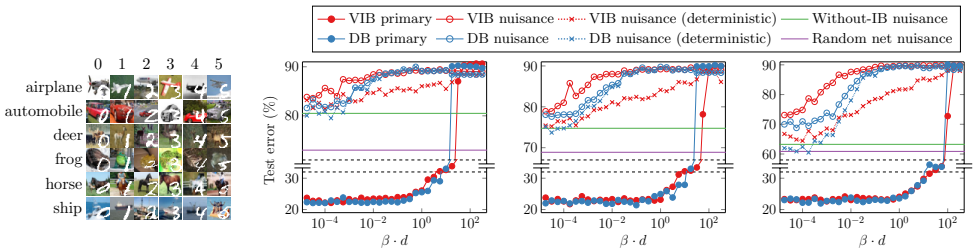
We experiment Drop-Bottleneck (DB) and Variational Information Bottleneck (VIB) [11] on occluded image classification tasks to show the following:

Table 2.7: The network architecture for the occluded image classification experiments.

Input image $32 \times 32 \times 3$		
Feature extractor	Conv [3×3, 96, stride 1] Conv [3×3, 96, stride 1] Conv [3×3, 96, stride 2]	
	Conv [3×3, 192, stride 1] Conv [3×3, 192, stride 1] Conv [3×3, 192, stride 2]	
	FC [d]	FC [$2d$]
	DB [d]	VIB [d]
Classifier	FC [10]	
	softmax	

- DB can control the degree of compression (*i.e.* degree of removal of task-irrelevant information) in the same way with VIB as the popular IB method.
- DB's deterministic compressed representation works as a reasonable replacement for its stochastic compressed representation and it maintains the learned indistinguishability better than the attempt of VIB's deterministic compressed representation.

We employ the Occluded CIFAR dataset using the experimental settings from Achille and Soatto [4]. The Occluded CIFAR dataset is created by occluding CIFAR-10 [69] images with MNIST [71] images as shown in Figure 2.7a, and each image has two labels of CIFAR and MNIST. We use a modified version of All-CNN-32 [4] equipped with an IB method (either of DB or VIB) for the feature extractor whose output dimension is d. The model architecture for



(a) Sample images (b) $d = 32$ (c) $d = 64$ (d) $d = 128$

Figure 2.7: (a) A few samples from Occluded CIFAR dataset [4]. (b)–(d) Test error plots on the *primary* task (*i.e.* the classification of occluded CIFAR images) and on the *nuisance* tasks (*i.e.* classification of the MNIST digits). For all the three types of tasks with VIB and DB, we use the same feature extractor trained for the *primary* task, where its deterministic representation is used only for the training and testing on the *nuisance (deterministic)* task. For comparison, we also include the performance on the *nuisance* task with the feature extractor from the *primary* task trained with no IB (*Without-IB*) and with the randomly initialized feature extractor (*Random net*).

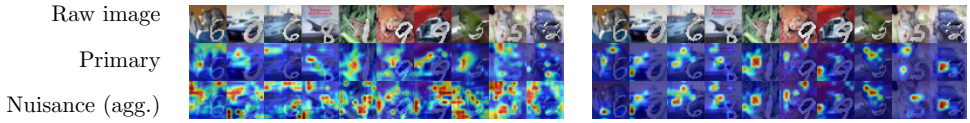
feature dimension d is described in Table 2.7. Batch normalization [54] is applied to Conv layers, and the ReLU [42, 81] activation is used at every hidden layer. Each run consists of two phases. In the first phase, we train the feature extractor with a logistic classifier using both the classification loss for CIFAR and the compression objective of the IB method. Fixing the trained feature extractor, we train a logistic classifier for MNIST in the second phase. We train two different versions of classifiers for each of VIB and DB using stochastic or deterministic compressed representation from the feature extractor. For the deterministic representation of VIB, we use the mode of the output Gaussian distribution. We use the Adam optimizer [65] with a learning rate of 0.001. To ensure convergence, in each training, the model is trained for 200 epochs with a batch size of 100.

Figures 2.7b–2.7d contain the experimental results with $d = 32$, $d = 64$, and $d = 128$. In the first phase, DB retains only a subset of features that

concentrate more on the CIFAR part of the images. Thus, the trained feature extractor preserves less information about the MNIST parts, and the errors of the MNIST classification are high. The first observation is that for both DB and VIB with the original stochastic compressed representation, *nuisance* plots show that increasing β from the minimum value to $\sim 0.1/d$ barely changes the *primary* CIFAR errors but grows the *nuisance* MNIST errors up to $\sim 90\%$ (*i.e.* the maximum error for the 10-way classification). With even larger β, enforcing stronger compression results in the increase of the *primary* errors too, as shown in *primary* plots. This suggests that both DB and VIB provide fine controllability over the removal of task-irrelevant information.

Secondly, if we move our focus to the *nuisance (deterministic)* plots in Figures 2.7b–2.7d, which show the test errors on the *nuisance* MNIST classification with the feature extractor's deterministic representation, the results become different between DB and VIB. In DB, the *nuisance (deterministic)* plots follow the *nuisance* plots in the range of β where the compression takes effect (*i.e.* where the *nuisance* errors increase). Moreover, the two plots get closer as β increases. It means that Drop-Bottleneck's deterministic compressed representations maintain the majority of the indistinguishability for the task-irrelevant information learned during the first phase, especially when β is large enough to enforce some degree of the compression. On the other hand, VIB's *nuisance (deterministic)* plots are largely different from the *nuisance* plots; even the *primary* errors rise before the *nuisance (deterministic)* errors reach their maximum values. This shows that employing the mode of VIB's output distribution as its deterministic representation results in loss of the learned indistinguishability.

In Figures 2.7b–2.7d, we also compare DB's *nuisance* errors with two special cases: employing the feature extractor trained on the primary task the



(a) Without DB (b) With DB

Figure 2.8: Grad-CAM [97] visualization for the last convolutional layer of the feature extractor on the Occluded CIFAR classification task. For the visualization, $d = 128$, and $\beta = 5.623/d$ for (b) are used. *Primary* denotes the maps of the logits for the primary labels. *Nuisance (agg.)* means the maps on the nuisance task aggregated over all the logits (*i.e.* 10 logits). (a) indicates that the feature extractor without DB trained on the primary task still outputs much information about the nuisance tasks, and thus the nuisance classifier could depend on the features extracted from the nuisance (MNIST) regions. In contrast, (b) suggests that the feature extractor with DB could learn to discard the nuisance features, so that the nuisance classifier mostly fails to learn due to the lack of nuisance-relevant features.

same way but without VIB or DB (*Without-IB*) and using the feature extractor whose weights are randomly initialized and fixed (*Random net*), where each number is obtained by averaging over 8 runs. The results indicate that while training the feature extractor on the *primary* task itself increases the *nuisance* errors by some degree compared to the errors with the randomly initialized feature extractors, DB can be employed to effectively eliminate the task-irrelevant information while keeping the *primary* errors low.

In summary, we confirm that DB provides controllability over the degree of compression in a similar way as VIB. On the other hand, DB's deterministic representation can be a reasonable replacement for its original stochastic representation in terms of preserving the learned indistinguishability, which is not exhibited by VIB.

2.5.7 Visualization of Task-irrelevant Information Removal

In this section, we visualize the removal of task-irrelevant information with Drop-Bottleneck (DB). To this end, we employ the Occluded CIFAR dataset [4] with the same experimental setup as in Section 2.5.6. Each image of Occluded CIFAR dataset is one of the CIFAR-10 [69] images occluded by MNIST [71] digit images. In the first phase of the experiments, the feature extractor and the classifier are trained on the primary (occluded CIFAR classification) task in a normal way. During the second phase, the learned feature extractor is fixed, and only a new classifier is trained on the nuisance (MNIST classification) task. In Section 2.5.6, we quantitatively showed that the feature extractor with DB could focus more on the occluded CIFAR images and preserve less information about the MNIST occlusions. We take a qualitative approach in this section and visualize the phenomenon using Grad-CAM [97]. Grad-CAM is popular for providing visual explanation given convolutional neural networks with their target values (*e.g.* target logits in classification tasks).

We first sample multiple test images from the Occluded CIFAR dataset, and load full, trained models, which include the feature extractor, primary classifier and nuisance classifier. We then obtain the activation maps for the last convolutional layer of the feature extractor on the primary and nuisance tasks. On the primary task, we compute the activation maps simply targeting the logits for the sample labels. However, on the nuisance task, we get the activation maps of all the logits and aggregate them by taking the maximum of the maps at each pixel location. Therefore, the aggregated maps on the nuisance task visualize the activation related to not only the logits for the true class labels but also the other logits, capturing most of the feature usage induced during the training on the nuisance task.

Figure 2.8 compares two trained models: the $d = 128$ model without DB, and the $d = 128$ model with DB (deterministic). We use the DB model with $\beta = 5.623/d$ for the visualization, as the value is sufficiently large enough to enforce strong compression while it still keeps the primary error not high. Figure 2.8a shows that regarding the logits for the nuisance (MNIST) task, a large portion of each image including the MNIST digit is activated in most cases, and thus it indicates that the feature extractor trained without DB preserves much of the nuisance features. On the other hand, Figure 2.8b visualizes that the feature extractor with DB outputs notably less of the nuisance features, preventing the nuisance classifier from learning correctly.

To sum up, we provide the visual demonstration that on the classification task with the Occluded CIFAR dataset, the feature extractor equipped with DB trained on the primary task could discard majority of the nuisance *i.e.* task-irrelevant information given a value of β that is strong enough.

2.5.8 Ablation Study: Exploration without DB

We perform an ablation study to show Drop-Bottleneck (DB)'s ability of dealing with task-irrelevant input information. We examine the performance of the same exploration method as described in Section 2.4 but without DB; that is, the feature vectors are fully used with no dropping. In order to emphasize the effectiveness of DB for noisy and task-irrelevant information, we conduct experiments with both noisy and original (*i.e.* without explicit noise injection) settings.

Table 2.8 shows the results on "Very Sparse" DMLab environments with "Image Action", "Noise", "Noise Action" and "Original" settings. Compared to "Original" setting where observations contain only implicit, inherent noisy

Table 2.8: Comparison of the average episodic sum of rewards in DMLab tasks (over 30 runs), where PPO + Ours (No-DB) denotes our exploration method without DB. The original (*i.e.* without explicit noise injection) and three noisy settings are tested: Image Action (IA), Noise (N), Noise Action (NA) and Original (O). The values are measured after 20M (4 action-repeated) time steps, with no seed tuning done. The baseline results are from Savinov et al. [92].

Method	DMLab			
	Very Sparse			
	IA	N	NA	O
PPO [95]	6.3	8.7	6.1	8.6
PPO + ICM [87]	4.9	6.0	5.7	11.2
PPO + EC [92]	7.4	13.4	11.3	24.7
PPO + ECO [92]	16.8	26.0	12.5	40.5
PPO + Ours (No-DB)	14.9	11.7	10.3	33.0
PPO + Ours (DB)	**28.8**	**29.1**	**26.9**	**42.7**
Improvement (%)	93.3	148.7	161.2	29.4

information irrelevant to state transitions, DB brings much more significant improvement to exploration methods in "Image Action", "Noise", and "Noise Action" settings, which inject explicit, severe transition-irrelevant information to observations. These results suggest that DB plays an important role handling noisy or task-irrelevant input information.

2.6 Summary

We presented Drop-Bottleneck as a novel information bottleneck method where compression is done by discretely dropping input features, taking into account each input feature's relevance to the target variable and allowing its joint training with a feature extractor. We then proposed an exploration method based on

Drop-Bottleneck, and it showed state-of-the-art performance on multiple noisy and reward-sparse navigation environments from VizDoom and DMLab. The results showed the robustness of Drop-Bottleneck's compressed representation against noise or task-irrelevant information. With experiments on ImageNet, we also showed that Drop-Bottleneck achieves better adversarial robustness compared to VIB and can reduce the feature dimension for inference.

Chapter 3

Disentangled Temporal Abstraction for Reusable Skills

3.1 Overview

Deep reinforcement learning (RL) has recently shown great advancement in solving various tasks, from playing video games [20, 76, 77] to controlling robot navigation [57]. While the standard RL is to maximize rewards from environments as a form of supervision, there has been a surge of interest in unsupervised learning without the assumption of extrinsic rewards [100, 103]. Discovering inherent skills in environments without supervision is important and desirable for multiple reasons. First, since it is still challenging to define an effective reward function for practical tasks [34, 48], unsupervised skill discovery helps reduce the burden of it by identifying effective skills for environments. Second, in sparse-reward environments, learned skills can encourage the exploration for encountering rewards, not only by providing useful primitives for the explo-

ration but also by reducing the effective horizon. Third, those skills can be directly used to solve downstream tasks, for example, by employing a meta-controller on top of the discovered skills in a hierarchical manner [2, 36, 99]. Finally, discovered skills could help better understand environments by providing interpretable sets of behaviors.

Unsupervised skill discovery can be formalized with the *options* framework [104], which generalizes primitive actions with the notion of *options*. For ease of learning, options, or synonymously *skills*, are often formulated by introducing a skill latent parameter z to an ordinary policy, resulting in a skill policy with a form of $\pi(a|s, z)$ keeping the same z for multiple steps or the full episode horizon [2, 36, 45, 99]. In recent research on the unsupervised skill discovery problem, information-theoretic approaches have been prevalent [2, 36, 45, 99].

In this work, we propose a novel unsupervised skill discovery method named *Information Bottleneck Option Learning (IBOL)*, whose two major novelties over existing approaches are (i) the *linearizer* and (ii) the *information bottleneck*-based skill learning. First, the linearizer is a kind of low-level policy to be suitable for skill discovery by converting a given environment into one with simplified dynamics. It reduces the skill discovery algorithm's burden to learn how to make transitions to diverse states in a given environment without any external rewards, which is not a straightforward job with fairly complex dynamics such as Ant and Humanoid from MuJoCo [107]. Once the linearizer is trained, it can be reused for multiple training sessions with different skill discovery approaches. Figure 3.1 compares the qualitative visualization of the skills learned by different methods in the locomotion (*i.e.* x-y) plane, in Ant. As shown, DIAYN [36], VALOR [2] and DADS [99] with the linearizer (with suffix '-L') learn far more diverse skills than the same methods without the linearizer.

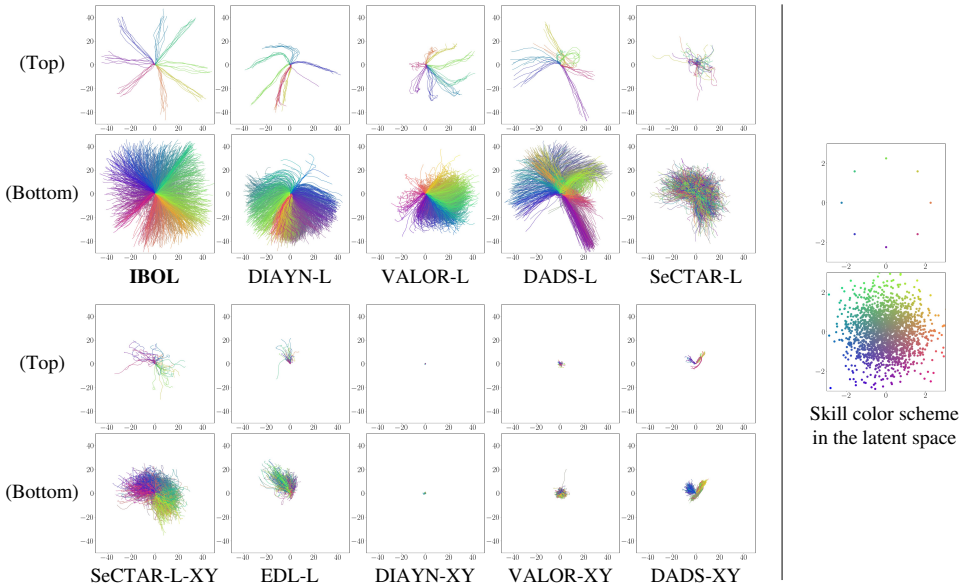


Figure 3.1: Visualization of the x-y traces of skills discovered by each algorithm in Ant, where the colors represent the two-dimensional skill latents used for the sampling of the skills (see the color scheme on the right). (Top) Trajectories of the six roll-outs from each of the eight different skill latents. (Bottom) Trajectories of 2000 skill latents sampled from the standard normal distribution.

Leveraging the environment simplified with the linearizer, IBOL discovers and learns skills based on the information bottleneck (IB) framework [11, 106]. Compared to prior approaches, IBOL can introduce some desirable properties to the learned skills. It discovers and learns skills with the skill latent variable Z in a more disentangled way, which makes the learned skills better interpretable with respect to Z. Interpretable models help understand their behaviors and provide intuition about their further uses [5]. Figure 3.1 demonstrates that the skill trajectories instantiated by IBOL have a visually simpler and more predictable mapping with the skill latents, which is one of the main requirements for increasing interpretability [5]. Moreover, the skills learned by IBOL cover

the locomotion plane more uniformly and widely. Finally, with the IB-style objective, the skill latent variable Z is learned to be not only informative about the discovered skills but also parsimonious to keep unrelated information about the skills.

Our key contributions can be summarized as follows.

- To the best of our knowledge, our method is the first to separate the problem of making transitions in the state space from skill discovery, simplifying the environment dynamics with independent pre-training, whose learning cost is amortized across multiple skill discovery trainings. It aids skill discovery methods to learn diverse skills by making the environment dynamics as linear as possible.
- We propose a novel skill discovery method with information bottleneck, which provides multiple benefits including learning skills in a more disentangled and interpretable way with respect to skill latents and being robust to nuisance information.
- Our method shows superior performance to various state-of-the-art unsupervised skill discovery methods including DADS [99], DIAYN [36] and VALOR [2] in multiple MuJoCo [107] environments. To verify this, we measure the information-theoretic metrics and the performance on four downstream tasks.

3.2 Preliminaries and Related Work

We review previous information-theoretic approaches to unsupervised skill discovery and discuss their limitations.

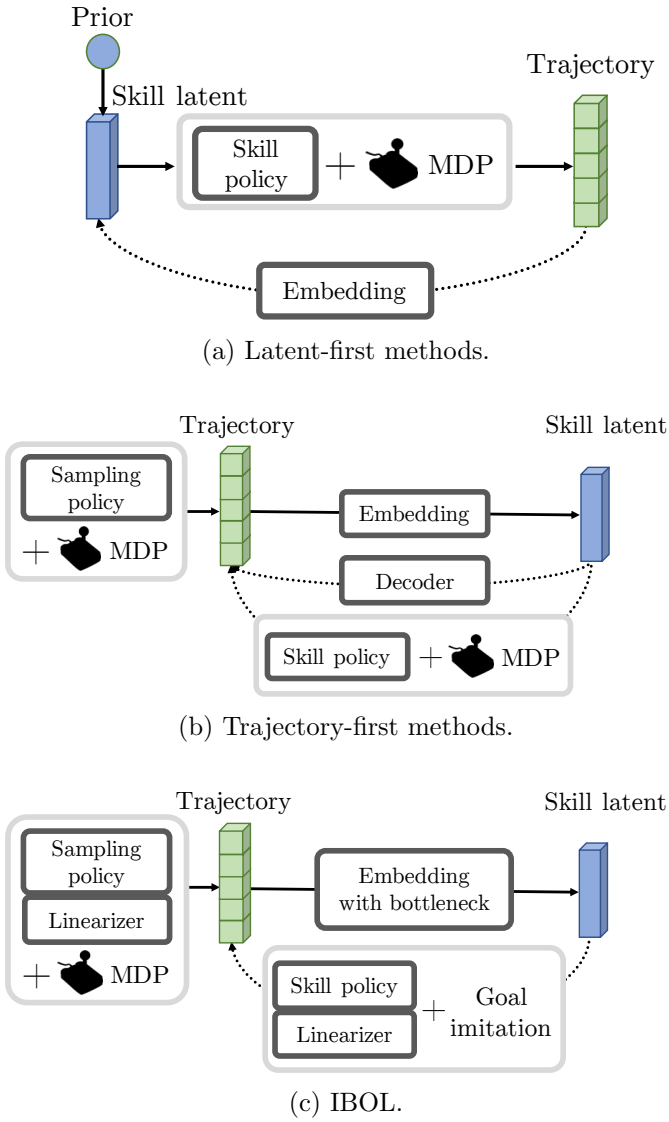
Preliminaries. We consider a Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p)$ *without external rewards*. $\mathcal{S}$ and $\mathcal{A}$ respectively denote the state and action spaces, and $p(s_{t+1}|s_t, a_t)$ is the transition function where $s_t, s_{t+1} \in \mathcal{S}$ and $a_t \in \mathcal{A}$. Given a policy $\pi(a_t|s_t)$, a trajectory $\tau = (s_0, a_0, \ldots, s_T)$ follows the distribution $\tau \sim p(\tau) = p(s_0) \prod_{t=0}^{T-1} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t)$. Within the options framework [104], we formulate the unsupervised skill discovery problem as learning a latent-conditioned skill policy $\pi(a_t|s_t, z)$ where $z \in \mathcal{Z}$ represents the *skill latent*. We consider continuous skill latents $z \in \mathbb{R}^d$. $h(\cdot)$ and $I(\cdot;\cdot)$ denote differential entropy and mutual information, respectively.

We introduce existing skill discovery methods in two groups: *latent-first* and *trajectory-first* methods.

Latent-first methods. Skill discovery methods in this category, such as VIC [45], DIAYN [36], VALOR [2], DADS [99] and HIDIO [110], first sample a skill latent z and then trajectories conditioned on z, as illustrated in Figure 3.2a. They aim to increase $I(Z; S)$, the mutual information between the skill latent and state variables. VALOR [2], which incorporates VIC and DIAYN as its special forms [2], optimizes a lower bound of the identity $I(Z; S) = h(Z) - h(Z|S)$. Its objective is to maximize

$$\mathbb{E}_{z \sim p(z)} \left[\mathbb{E}_{\tau \sim p(\tau|z)} [\log p_D(z|s_{0:T})] + \beta \cdot \sum_{t=0}^{T-1} h(A_t) \right],$$

where A_t is the action variable that follows $\pi(a_t|s_t, z)$, β is the entropy coefficient, $p(z)$ is the prior distribution over z, and $p_D(z|s_{0:T})$ is a trainable decoder that reconstructs the original z given $s_{0:T}$. Achiam et al. [2] show that this objective has an equivalency to β-VAE [49] with the structure of z (input) $\rightarrow \tau$ (latent) $\rightarrow z$ (reconstruction). However, this objective does not take advantage of the benefits that the VAE formulations can provide, such as the theoretical



(a) Latent-first methods.

(b) Trajectory-first methods.

(c) IBOL.

Figure 3.2: Architecture overview of (a) latent-first methods, (b) trajectory-first methods and (c) IBOL.

connection to more disentangled and interpretable z [3, 4, 28]. DADS [99] optimizes the other identity $I(Z;S) = h(S) - h(S|Z)$, using a skill dynamics model $q(s_{t+1}|s_t, z)$ that predicts the next state conditioned on z. While the learned

dynamics model enables model-based planning, it lacks an explicit mapping from states to skill latents, and thus hardly obtains disentangled skill latents z.

Trajectory-first methods. Another group of methods first samples trajectories and then encodes them into skill latents using the variational autoencoder (VAE) [66], as visualized in Figure 3.2b. This category includes SeCTAR [29], EDL [25] and OPAL [9]. SeCTAR and EDL have separate objectives for their *exploratory policies* to sample diverse trajectories by maximizing $h(p(\tau))$ or $h(S)$. OPAL assumes an offline RL setting where a fixed set of trajectories is priorly given. While these methods employ the VAE with the usual direction of $\tau \to z \to \tau$ (EDL has $s \to z \to s$) that encourages disentangled representations, they have a limitation that the exploratory policy *only* maximizes the diversity of trajectories. On the contrary, our IBOL method, which also falls into this category, *jointly* maximizes both the diversity and discriminability of trajectories (Section 3.3.6), which leads to a significant improvement in performance (Section 3.4).

Finally, all of the prior works learn the skill policies on top of raw environment dynamics. Although dealing with raw dynamics is not highly demanding in simple environments, it could hinder the skill learning in environments with complex dynamics such as Ant and Humanoid from MuJoCo [107]. IBOL solves the issue by *linearizing* the environment dynamics ahead of skill discovery so that it can acquire diverse skills by reaching different states more easily in the simplified environment dynamics. Furthermore, we find that the linearization benefits other existing skill discovery methods too (Section 3.4).

3.3 Information Bottleneck Option Learning (IBOL)

We decompose the skill discovery problem into two separate phases. Firstly,

Algorithm 1 (Phase 1) Training Linearizer

```
Initialize linearizer π^lin.
while not converged do
  for i = 1 to n do
    Sample goals (g_0^(i), g_ℓ^(i), g_2ℓ^(i), ...).
    Sample trajectory using π^lin and goals.
    Compute linearizer reward R^lin using Equation (3.1).
    Add trajectory to replay buffer.
  end for
  Update π^lin using collected samples from replay buffer with SAC [47].
end while
```

Algorithm 2 (Phase 2) Skill Discovery

```
Load pre-trained linearizer π^lin.
Initialize sampling policy π_θs, trajectory encoder p_ϕ, skill policy π_θz.
while not converged do
  for i = 1 to n do
    Sample trajectory using π_θs on top of π^lin.
  end for
  Compute objective from Equation (3.9).
  Compute its gradient w.r.t. ϕ, θ_z.
  Compute its policy gradient w.r.t. θ_s.
  Jointly update π_θs, p_ϕ, π_θz with gradients.
end while
```

IBOL trains the *linearizer* that lifts the burden from the skill discovery algorithm to generate diverse states and trajectories under complex environment dynamics (Section 3.3.1). Secondly, on top of the pre-trained linearizer, IBOL learns to map trajectories into the continuous skill latent space, with the information bottleneck principle [11, 106] (Section 3.3.2). Figure 3.2c provides the conceptual illustration of IBOL. Algorithm 1 overviews the training of the linearizer in the first phase and Algorithm 2 describes the skill discovery process in the second.

3.3.1 Linearization of Environments

The *linearizer* π^{lin} is a pre-trained low-level policy that aims to "linearize" the environment dynamics. It takes as input *goals* produced by IBOL's policies for skill discovery (will be discussed in Section 3.3.2), and translates them into raw actions in the direction of a given goal while interacting with the environment. We define the linearizer $\pi^{\text{lin}}(a_t|s_t, g_t)$ as a goal-conditioned policy [93], which takes both a state $s_t \in \mathcal{S}$ and a goal $g_t \in \mathcal{G}$ as input and outputs a probability distribution over actions $a_t \in \mathcal{A}$. The goal space $\mathcal{G}$ is defined as $\mathcal{G} = [-1, 1]^{dim(\mathcal{S})}$, which has the same dimensionality as the state space (up to 47 in our experiments). Each goal dimension provides a signal for the direction in the corresponding state dimension. We assume that a goal $g_t \in \mathcal{G}$ is given at every ℓ-th time step such that $t \equiv 0 \pmod{\ell}$ (called a *macro* time step), and otherwise kept fixed, *i.e.* $g_t = g_{t-1}$ for $t \not\equiv 0 \pmod{\ell}$.

We sample goals $(g_0, g_\ell, g_{2\ell}, \ldots)$ at the beginning of each roll-out and train the linearizer with a reward function of

$$R^{\text{lin}}(s_t, g_t, a_t, s_{t+1}) = \frac{1}{\ell}(s_{(c+1)\cdot\ell} - s_{c\cdot\ell})^\top g_t, \tag{3.1}$$

where $c = \lfloor \frac{t}{\ell} \rfloor$. It corresponds to the inner product of the goal g_t and the state difference between macro time steps: $(s_{(c+1)\cdot\ell} - s_{c\cdot\ell})$. Intuitively, each goal dimension value (ranging from -1 to $+1$) indicates the desired direction and the degree of change in the corresponding state dimension.

The inner product in the reward function has several advantages for skill discovery compared to the Euclidean distance in prior approaches [79, 80]. First, unlike the Euclidean distance that needs to specify the valid range of each state dimension, the inner product only takes care of directions in the state space. Thus, training of the linearizer requires no additional supervision on

specifying valid goal spaces or state ranges. Second, by setting some dimensions of a goal to be (near-)zero values, we can ignore changes in the corresponding state dimensions, which is not achievable with the Euclidean distance. This enables IBOL's policies for skill discovery to ignore nuisance dimensions without manually specifying them (Section 3.3.2).

We find that the linearizer benefits not only IBOL but also other skill discovery methods since it can promote reaching diverse and distant states easier, as shown in Figure 3.1.

3.3.2 Skill Discovery with Bottleneck Learning

On top of the pre-trained and fixed linearizer π^{lin}, we learn policies that produce *goals* and acquire a continuous set of skills that is not only distinguishable and diverse but also disentangled and interpretable. The linearizer alone is highly limited to discovery abstractive and informative skills, since it is trained with the inner product reward function and thus optimized for transitioning to distant states rather than the mapping with the latent space. Additionally, IBOL can fix possibly imperfect linearization with the linearizer by combining appropriate high-level goals. In Section 3.4, we will demonstrate that how such limitations of the linearizer can be resolved by the following skill discovery process.

In contrast to previous skill discovery methods that maximize $I(S; Z)$ [2, 36, 45, 99], IBOL consists of the following three learnable components based on the information bottleneck:

1. The sampling policy $\pi_{\theta_s}(g_t|s_t)$ produces diverse and easily mappable trajectories.

2. The trajectory encoder $p_\phi(z|s_{0:T})$ encodes the state trajectories into the skill latent space.

3. The skill policy $\pi_{\theta_z}(g_t|s_t, z)$ learns to imitate the skills given their latents.

Note that the sampling and skill policies produce goals g_t instead of raw actions a, as they operate on top of the linearizer. We will first start with the sampling policy π_{θ_s} and introduce our IB objective for the trajectory encoder p_ϕ. We then show that it naturally leads to the emergence of the skill policy π_{θ_z} as a variational approximation to the sampling policy π_{θ_s}.

IBOL's objective. Assuming trajectories generated by the sampling policy, $\{\tau^{(1)}, \tau^{(2)}, \ldots, \tau^{(n)}\}$, our objective is to embed the state trajectories $\{s^{(1)}_{0:T}, \ldots, s^{(n)}_{0:T}\}$ into the skill latent space $\mathcal{Z}$. We encode the *state* trajectory $s_{0:T}$, not the *whole* trajectory τ, because an outside observer can only see the agent's state, not its underlying actions or goals. However, the encoded skill latent z should contain sufficient information about the underlying goals so that the whole trajectory is reproducible from z. Furthermore, since raw states often contain nuisance information not pertaining to skill discovery, z is encouraged to minimally contain unnecessary or noisy information in the states irrelevant to the goals. This leads to the Information Bottleneck objective [11, 106] over the structure of $S_{0:T}$ (input) $\rightarrow Z$ (latent) $\rightarrow G_{0:T-1}$ (target).

Formally, let us first define the *sampling policy* parameterized by θ_s as $\pi_{\theta_s}(g_t|s_t)\colon \mathcal{S} \rightarrow \mathcal{P}(\mathcal{G})$, which maps a state to a probability distribution over goals. A trajectory $\tau = (s_0, g_0, s_1, \ldots, g_{T-1}, s_T)$ obtained by the sampling policy is acquired from the distribution $\tau \sim p_{\theta_s}(\tau) = p(s_0)\prod_{t=0}^{T-1} \pi_{\theta_s}(g_t|s_t)p(s_{t+1}|s_t, g_t)$. Under the distribution $p_{\theta_s}(\tau)$, let S_t be a random variable corresponding to s_t and G_t be a random variable for g_t. We define the *trajectory encoder* parameterized by ϕ as $p_\phi(z|s_{0:T})\colon \mathcal{S}^{T+1} \rightarrow \mathcal{P}(\mathcal{Z})$ that

maps a state trajectory to a probability distribution over skill latents z in the skill space $\mathcal{Z}$. Let Z be a random variable for z.

We formulate our IB objective as follows. First, given S_t, the skill latent Z should be informative about the goal G_t that the sampling policy has produced, which leads to the *prediction term* $I(Z; G_t|S_t)$. Second, Z should be penalized for preserving information about the state trajectory but unrelated to the goals, which corresponds to the *compression term* $I(Z; S_{0:T})$. Summing these up, we obtain the following objective:

$$\text{maximize } \mathbb{E}_t[I(Z; G_t|S_t) - \beta \cdot I(Z; S_{0:T})], \tag{3.2}$$

where $\mathbb{E}_t$ is the expectation over $\{0, 1, \ldots, T-1\}$, and β is a constant that controls the weight of the compression term.

Lower bound optimization. Since the objective is practically intractable, we derive its lower bound [11] as follows (see Section 3.3.3 for the full derivation):

$$\begin{aligned}
&\mathbb{E}_t[I(Z; G_t|S_t) - \beta \cdot I(Z; S_{0:T})] && (3.3)\\
&\quad = \mathbb{E}_{\substack{\tau \sim p_{\theta_\mathrm{s}}(\tau), t, \\ z \sim p_\phi(z|s_{0:T})}} \left[\log \frac{p_{\theta_\mathrm{s}}(g_t|s_t, z)}{\pi_{\theta_\mathrm{s}}(g_t|s_t)} - \beta \log \frac{p_\phi(z|s_{0:T})}{p_\phi(z)} \right] \\
&\quad \geq \mathbb{E}_{\tau \sim p_{\theta_\mathrm{s}}(\tau)} \Big[\mathbb{E}_{z \sim p_\phi(z|s_{0:T}), t} \Big[\log \pi_{\theta_\mathrm{z}}(g_t|s_t, z) && (3.4)\\
&\qquad - \log \pi_{\theta_\mathrm{s}}(g_t|s_t) \Big] - \beta \cdot D_{\mathrm{KL}}(p_\phi(Z|s_{0:T}) \| r(Z)) \Big],
\end{aligned}$$

where D_{KL} denotes the Kullback-Leibler (KL) divergence. Here we use two variational approximations: the *skill policy*'s output distribution $\pi_{\theta_\mathrm{z}}(g_t|s_t, z)$ is a variational approximation of $p_{\theta_\mathrm{s}}(g_t|s_t, z)$ and $r(z)$ is that of the marginal distribution $p_\phi(z)$. In Equation (3.4), the first term $\log \pi_{\theta_\mathrm{z}}(g_t|s_t, z)$ makes the skill policy $\pi_{\theta_\mathrm{z}}(g_t|s_t, z)$ imitate the sampling policy's output given the skill latent z; thus we call this the *imitation term*. The third term $-\beta D_{\mathrm{KL}}(p_\phi(Z|s_{0:T}) \| r(Z))$

is the *compression term* that forces the output distributions of the trajectory encoder to be close to $r(z)$. We will revisit the second term $-\log \pi_{\theta_s}(g_t|s_t)$ later.

We fix $r(z)$ to $\mathcal{N}(0, I)$ as in Alemi et al. [11] for the following reasons. First, it enables us to analytically compute the KL divergence. Second, more importantly, it induces disentanglement between the dimensions of z [3, 4, 28]. Disentangled representations lead to more interpretable skills with respect to their skill latents z. In Section 3.3.4, we provide further details on how the compression term encourages the disentanglement of skill latent dimensions.

It is worth noting that the first and third terms in Equation (3.4) are related to the β-VAE objective [11, 49, 66] and previous skill discovery methods that use trajectory VAEs [9, 29]. The first and the third term correspond to the reconstruction loss and the KL divergence loss in β-VAEs, respectively. One important difference is that we reconstruct not the original state trajectories but their underlying goals. It eliminates the need for state decoders or sampling with the skill policy during training.

3.3.3 Derivation of the Lower Bound

We describe the derivation of Equation (3.4). Starting with Equation (3.3),

$$\begin{aligned}
&\mathbb{E}_t[I(Z; G_t|S_t) - \beta \cdot I(Z; S_{0:T})] \\
&\quad = \mathbb{E}_{\substack{\tau \sim p_{\theta_s}(\tau), t, \\ z \sim p_\phi(z|s_{0:T})}} \left[\log \frac{p_{\theta_s}(g_t|s_t, z)}{\pi_{\theta_s}(g_t|s_t)} - \beta \log \frac{p_\phi(z|s_{0:T})}{p_\phi(z)} \right].
\end{aligned}$$

For the first term, as described earlier, we use the skill policy's output distribution $\pi_{\theta_z}(g_t|s_t, z)$ as a variational approximation of $p_{\theta_s}(g_t|s_t, z)$, which derives

$$\begin{aligned}
&\mathbb{E}_{\substack{\tau\sim p_{\theta_s}(\tau),t,\\ z\sim p_\phi(z|s_{0:T})}}\left[\log \frac{p_{\theta_s}(g_t|s_t, z)}{\pi_{\theta_s}(g_t|s_t)}\right] \\
&\quad = \mathbb{E}_{\substack{\tau\sim p_{\theta_s}(\tau),t,\\ z\sim p_\phi(z|s_{0:T})}}\left[\log \frac{\pi_{\theta_z}(g_t|s_t, z)}{\pi_{\theta_s}(g_t|s_t)}\right] \\
&\qquad + \mathbb{E}_{\substack{s_{0:T}\sim p_{\theta_s}(\cdot),t,\\ z\sim p_\phi(z|s_{0:T})}}\Big[D_{\mathrm{KL}}(p_{\theta_s}(G_t|s_t, z)\|\pi_{\theta_z}(G_t|s_t, z))\Big] \\
&\quad \geq \mathbb{E}_{\substack{\tau\sim p_{\theta_s}(\tau),t,\\ z\sim p_\phi(z|s_{0:T})}}\left[\log \frac{\pi_{\theta_z}(g_t|s_t, z)}{\pi_{\theta_s}(g_t|s_t)}\right]. \qquad (3.5)
\end{aligned}$$

Also, for the second term, we use $r(z)$ as the variational approximation of the marginal distribution $p_\phi(z)$, and it derives

$$\begin{aligned}
&\mathbb{E}_{\substack{\tau\sim p_{\theta_s}(\tau),\\ z\sim p_\phi(z|s_{0:T})}}\left[\log \frac{p_\phi(z|s_{0:T})}{p_\phi(z)}\right] \\
&\quad = \mathbb{E}_{\substack{\tau\sim p_{\theta_s}(\tau),\\ z\sim p_\phi(z|s_{0:T})}}\left[\log \frac{p_\phi(z|s_{0:T})}{r(z)}\right] - D_{\mathrm{KL}}(p_\phi(Z)\|r(Z)) \\
&\quad \leq \mathbb{E}_{\substack{\tau\sim p_{\theta_s}(\tau),\\ z\sim p_\phi(z|s_{0:T})}}\left[\log \frac{p_\phi(z|s_{0:T})}{r(z)}\right] \\
&\quad = \mathbb{E}_{\tau\sim p_{\theta_s}(\tau)}\Big[D_{\mathrm{KL}}(p_\phi(Z|s_{0:T})\|r(Z))\Big]. \qquad (3.6)
\end{aligned}$$

Combining the derivation of Equations (3.5) and (3.6) obtains Equation (3.4).

3.3.4 Encouraging Disentanglement

Disentanglement learning methods often model the generation process as $X \sim p(\cdot|Z)$, assuming that data X is generated with its underlying latent factors Z [32, 60]. While the aggregated posterior of Z is defined as $q(Z) = \int_x q(Z|x)p(x)dx$ for an encoder $q(Z|X)$ [75], one of common disentanglement approaches is to penalize the total correlation of $q(Z)$, expressed as $TC(q(Z)) = D_{\mathrm{KL}}(q(Z)\| \prod_i^d q(Z_i))$.

The IB framework has theoretical connections to the disentanglement of representation [3, 4, 28]. In Section 3.3.2, we derived our objective in the form of IB. Especially, if we model the prior of Z as $r(Z) = \prod_i^d r(Z_i)$, the term $D_{\mathrm{KL}}(p_\phi(Z|s_{0:T})\|r(Z))$ in Equation (3.4) is decomposed into three terms revealing the total correlation term $TC(p_\phi(Z)) = D_{\mathrm{KL}}(p_\phi(Z)\|\prod_i^d p_\phi(Z_i))$ [28] (refer to Section 3.3.5 for the proof), where the aggregated posterior of the trajectory encoder is $p_\phi(Z) = \int_{s_{0:T}} p_\phi(Z|s_{0:T})p(s_{0:T})ds_{0:T}$. By penalizing $TC(p_\phi(Z))$, we encourage each dimension of the skill latent space $\mathcal{Z}$ disentangled from the others with respect to $S_{0:T}$. As a result, the learned skill latent Z can provide improved abstraction, where each dimension focuses more on only its corresponding behavior of the discovered skills.

3.3.5 Decomposition of the KL Divergence Term

When the prior of Z is modeled as a factorized distribution, *i.e.* $r(Z) = \prod_i^d r(Z_i)$, the KL divergence term in Equation (3.4) can be decomposed as follows [28]:

$$
\begin{aligned}
&D_{\mathrm{KL}}(p_\phi(Z|s_{0:T})\|r(Z)) \qquad (3.7)\\
&\quad= \mathbb{E}_{z\sim p_\phi(Z|s_{0:T})}\left[\log\frac{p_\phi(z|s_{0:T})}{r(z)}\right]\\
&\quad= \mathbb{E}_{z\sim p_\phi(Z|s_{0:T})}\left[\log\frac{p_\phi(z|s_{0:T})}{p_\phi(z)}\right] + \mathbb{E}_{z\sim p_\phi(Z|s_{0:T})}\left[\log\frac{p_\phi(z)}{\prod_{i=1}^d p_\phi(z_i)}\right]\\
&\qquad+ \mathbb{E}_{z\sim p_\phi(Z|s_{0:T})}\left[\log\frac{\prod_{i=1}^d p_\phi(z_i)}{\prod_{i=1}^d r(z_i)}\right]\\
&\quad= D_{\mathrm{KL}}(p_\phi(Z|s_{0:T})\|p_\phi(Z)) + D_{\mathrm{KL}}(p_\phi(Z)\|\prod_{i=1}^d p_\phi(Z_i))\\
&\qquad+ \sum_{i=1}^d D_{\mathrm{KL}}(p_\phi(Z_i)\|r(Z_i)), \qquad (3.8)
\end{aligned}
$$

where $p_\phi(Z) = \int_{s_{0:T}} p_\phi(Z|s_{0:T})p(s_{0:T})ds_{0:T}$ denotes the aggregated posterior.

The second term corresponds to the total correlation of Z, as $TC(p_\phi(Z)) = D_{\mathrm{KL}}(p_\phi(Z)\| \prod_i^d p_\phi(Z_i))$, encouraging Z to have a more disentangled representation. The third term operates as a regularizer, which pushes each dimension of the aggregated posterior $p_\phi(Z)$ to be located in the vicinity of the prior. The expectation of the first term can be represented in the form of mutual information, as $\mathbb{E}_{s_{0:T}}\left[D_{\mathrm{KL}}(p_\phi(Z|s_{0:T})\|p_\phi(Z))\right] = I(S_{0:T}; Z)$. This corresponds to the original compression term before applying the variational approximation.

3.3.6 Training

The trajectory encoder and the skill policy can be trained using the reparameterization trick as in VAEs [66]; we optimize those two terms in Equation (3.4) with respect to their parameters, θ_{z} and ϕ. Note that the skill policy does not interact with the environment during training and the second term $-\log \pi_{\theta_{\mathrm{s}}}(g_t|s_t)$ is independent of these parameters.

The sampling policy $\pi_{\theta_{\mathrm{s}}}(g_t|s_t)$ can be trained with the same objective of Equation (3.4). This is the key difference with prior trajectory-first methods that employ similar VAE architectures [9, 25, 29] (Section 3.2). They either have a separate objective for training their sampling policies [25, 29] or assume the offline RL setting [9]. In contrast, we jointly train all the components with the same objective.

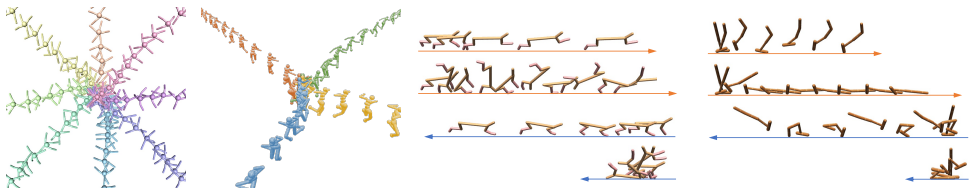
There are several merits of using the same objective. First, the second term $-\log \pi_{\theta_{\mathrm{s}}}(g_t|s_t)$, referred to as the *entropy term*, encourages the sampling policy to produce diverse trajectories. In deterministic environments, maximizing this term is equivalent to maximizing the entropy of whole trajectories, as $h(p_{\theta_{\mathrm{s}}}(\tau)) = T \cdot \mathbb{E}_{\tau \sim p_{\theta_{\mathrm{s}}}(\tau), t}[-\log \pi_{\theta_{\mathrm{s}}}(g_t|s_t)] + (\mathrm{const})$. Note that this entropy term

often remains constant in IB literature [11], assuming that the training data (*e.g.* images) are given, whereas we can diversify the "training data" too. Second, optimizing the whole Equation (3.4) makes the sampling policy generate trajectories that are not only diverse but also *easily encoded into the skill space for the trajectory encoder and skill policy* thanks to the first and third terms, which helps the learning of the two components as well. This is not achievable when the sampling policy is trained with a diversity maximizing objective only. In Section 3.4, we will demonstrate how taking into account both diversity and encodability leads to a huge difference in performance, comparing with baselines without such consideration.

Practical training. Since the expectation in Equation (3.4) involves the sampling policy's roll-outs in the environment, we optimize the sampling policy via the policy gradient method. However, there exists one practical difficulty when training IBOL. Since the sampling policy $\pi_{\theta_\mathrm{s}}(g_t|s_t)$ lacks a variable about the context (*e.g.* z) compared to the skill policy $\pi_{\theta_\mathrm{z}}(g_t|s_t, z)$, π_{θ_s} is less expressive than π_{θ_z}, which could end up with a suboptimal convergence. To solve this issue, we introduce a new context parameter $u \in \mathcal{U}$ with its prior $p(u)$ to the sampling policy, redefining it as $\pi_{\theta_\mathrm{s}}(g_t|s_t, u) : \mathcal{S} \times \mathcal{U} \to \mathcal{P}(\mathcal{G})$. The new parameter u for π_{θ_s} plays a similar role to the skill latent z for π_{θ_z}. We also fix $p(u) = \mathcal{N}(0, I)$ as in $r(z)$. To obtain roll-outs from the sampling policy, we first sample u from its prior, and then keep sampling goals with the fixed u.

Given that $r(z)$ and $p(u)$ are identical, we additionally include an auxiliary term $\mathbb{E}_{u\sim p(u),\tau\sim p_{\theta_\mathrm{s}}(\tau|u)}[\lambda \cdot p_\phi(u|s_{0:T})]$ to further stabilize the training. This term guides the output of the trajectory encoder p_ϕ to u, which is from $p(u) = r(z)$, operating compatibly with the compression term.

Finally, with the revised sampling policy, we approximate the second term



(a) Ant (b) Humanoid (c) HalfCheetah (d) Hopper

Figure 3.3: Examples of rendered scenes illustrating the skills that IBOL discovers with no rewards in MuJoCo environments. (a) Ant moving in various directions. (b) Humanoid running in different directions. (c) (Top to Bottom) HalfCheetah running forward, rolling forward, running backward and flipping backward. (d) (Top to Bottom) Hopper hopping forward, crawling forward, jumping backward and flipping backward.

in Equation (3.4) as done in DADS [99]: $\pi_{\theta_s}(g_t|s_t) = \int_u \pi_{\theta_s}(g_t|s_t, u)p(u|s_t)du \approx \int_u \pi_{\theta_s}(g_t|s_t, u)p(u)du \approx \frac{1}{L}\sum_{i=1}^{L} \pi_{\theta_s}(g_t|s_t, u_i)$ for $u_i \overset{\text{i.i.d.}}{\sim} p(u)$, where L is the number of samples from the prior. Therefore, the final objective of our method is

$$\mathbb{E}_{\substack{u\sim p(u),\\ \tau\sim p_{\theta_s}(\tau|u)}} \left[\mathbb{E}_{\substack{z\sim p_\phi(z|s_{0:T}),t\\ u_i \overset{\text{i.i.d.}}{\sim} p(u)}} \left[J^{\mathrm{P}} \right] - \beta \cdot J^{\mathrm{C}} + \lambda \cdot p_\phi(u|s_{0:T}) \right] \tag{3.9}$$

$$\text{where } J^{\mathrm{P}} = \log \pi_{\theta_z}(g_t|s_t, z) - \log \left(\frac{1}{L} \sum_{i=1}^{L} \pi_{\theta_s}(g_t|s_t, u_i) \right)$$

$$J^{\mathrm{C}} = D_{\mathrm{KL}}(p_\phi(Z|s_{0:T}) \| r(Z)).$$

3.4 Experiments

We compare our IBOL with other state-of-the-art methods in multiple aspects. First, we visualize the learned skills with the trajectory plots and the rendered scenes from environments (Section 3.4.2). Second, we quantitatively evaluate the skill discovery methods in terms of multiple information-theoretic metrics (Section 3.4.3). Third, we evaluate the trained policies on the downstream tasks

with different configurations (Section 3.4.5). Finally, we present additional behaviors of IBOL in the absence of the locomotion signals and with the distorted goal space (Section 3.4.8).

3.4.1 Experimental Setup

Basic setup and baselines. We experiment with MuJoCo environments [107] for multiple tasks: Ant, HalfCheetah, Hopper and Humanoid from OpenAI Gym [22] with the setups by Sharma et al. [99] and D'Kitty from ROBEL [8] adopting the configurations by Sharma et al. [98]. We use D'Kitty with the random dynamics setting; in each episode, multiple properties of the environment, such as its joint dynamics, friction and height field, are randomized, which provides an additional challenge to agents. We mainly compare our method with recent information-theoretic unsupervised skill discovery methods, VALOR [2], DIAYN [36] and DADS [99]. Since IBOL operates on top of the linearized environments, we also consider a variant of each algorithm that uses the same linearizer, denoted with the suffix '-L' (*e.g.* VALOR-L). In Ant experiments, we use the suffix '-XY' to refer to the methods with the *x-y prior* [99], which forces them to focus exclusively on the locomotion skills by restricting the observation space of the trajectory encoder (or the skill dynamics model in DADS) to the x-y coordinates.

Implementation. For experiments, we use pre-trained linearizers with two different random seeds on each environment. When training the linearizers, we sample a goal g at the beginning of each roll-out and fix it within that episode to learn consistent behaviors, as in SNN4HRL [40]. We consider continuous priors for skill discovery methods. Especially, we use the standard normal distribution for $p(u)$ and $r(z)$ in IBOL and for $p(z)$ in other methods. We employ garage

[41] and PyTorch [86] to implement IBOL, DIAYN [36], VALOR [2] and DADS [99]. We use the official implementations for EDL[1] [25] and SeCTAR[2] [29] with additional tuning of hyperparameters to ensure fair comparisons.

Environments. We experiment with robot simulation environments in MuJoCo [107]: Ant, HalfCheetah, Hopper and Humanoid from OpenAI Gym [22] adopting the configurations by Sharma et al. [99] and D'Kitty with random dynamics from ROBEL [8] with the setups provided by Sharma et al. [98]. We use a maximum episode horizon of 200 environment steps for Ant, HalfCheetah and D'Kitty, 500 for Hopper and 1000 for Humanoid. Note that D'Kitty and Humanoid have variable episode horizons, and we use an alive bonus of $3e-2$ at each step in the training of the linearizers for Humanoid to stabilize the training. For the linearizer, we omit the locomotion coordinates of the torso (x and y for Ant, Humanoid and D'Kitty, and x for the others) from the input of the policy. Note that the linearizer could be agnostic to the agent's global location since its rewards are computed only with the change of the state. On the other hand, we retain them for skill discovery policies and meta-controller policies since, without those coordinates, the expressiveness of learnable skills may be restricted. However, as DADS originally omits the x-y coordinates from the inputs in Ant [99], we also test baseline methods of $d = 2$ with both the omission and the x-y prior [99], denoted with the suffix '-XYO', in Figure 3.4.

Models. In the experiments, we use an MLP with two hidden layers of 512 dimensions for each non-recurrent learnable component except for the linearizer, which uses two hidden layers of 1024 dimensions. We use the tanh and ReLU nonlinearities for the policies and the others, respectively. We model the outputs of the linearizer and the meta-controller for downstream tasks with

[1] `https://github.com/victorcampos7/edl`
[2] `https://github.com/wyndwarrior/Sectar`

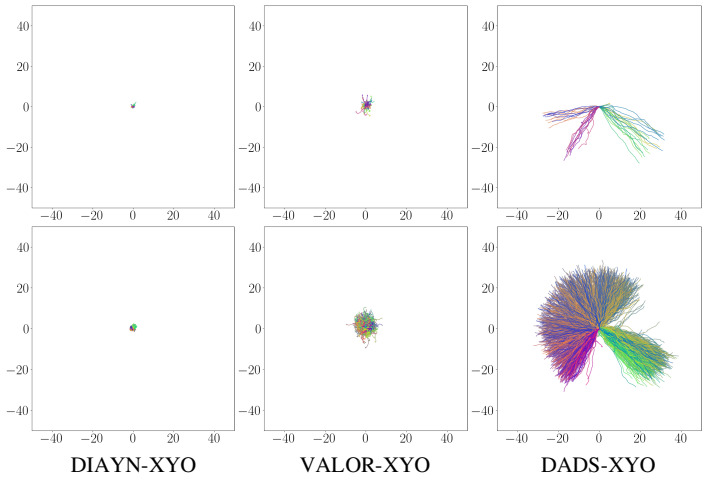


Figure 3.4: Visualization of the x-y traces of the skills for Ant discovered by each baseline method trained with the omission of the x-y coordinates from the inputs and the x-y prior [99]. The same skill latents are used with Figure 3.1.

the factorized Gaussian distribution followed by a tanh transformation to fit into the action space of environments. We use the Beta distribution policies for the skill discovery methods. To feed policies with the skill latent variable, we concatenate the skill latent z for each episode with its state s_t at every time step t. For the trajectory encoder of IBOL and VALOR, we use a bidirectional LSTM with a 512-dimensional hidden layer followed by two 512-dimensional FC layers. When training VALOR without the linearizer, we use a subset of the full state sequence of each trajectory with evenly spaced states to match the effective horizon with VALOR-L, following Achiam et al. [2]. We employ the original implementation choice of DADS to predict $\Delta s = s' - s$ (instead of s') from s and z with its skill dynamics model [99]. Both s and Δs are batch-normalized, with a fixed covariance matrix of I and a Gaussian mixture model with four heads, again following Sharma et al. [99].

Common setup for training. We use the Adam optimizer [65] with a

learning rate of $1e-4$ for skill discovery methods and $3e-4$ for the others. We normalize each dimension of states, which is important since it helps skill discovery methods equally focus on every dimension of the state space rather than solely on large-scale dimensions. Note that while we observe that the skill discovery methods primarily focus on the locomotion dimensions in the absence of the x-y prior [99] as in Figure 3.1, this is not due to the scale of those dimensions, as all the state dimensions are normalized. We hypothesize it is because the locomotion dimensions are those which can have high informativeness with the skill latent variable. When training meta-controllers or skill policies with the linearizer, we use the exponential moving average. For the rest, we use the mean and standard deviation pre-computed from 10000 trajectories with an episode length of 50. Meta-controllers for downstream tasks and skill policies use the mode of each output distribution from their lower-level policies. At every epoch of the training of the linearizer or meta-controller for downstream tasks, we collect ten trajectories for Ant, HalfCheetah, Hopper and D'Kitty, and five trajectories for Humanoid. For the skill discovery methods with the linearizer, at each epoch 64 trajectories are sampled for Ant, HalfCheetah and D'Kitty and 32 for Hopper and Humanoid. When training the methods without the linearizer (*e.g.* VALOR-XY), we collect ten trajectories for Ant, since their effective horizon is longer than that with the linearizer.

Training of the linearizer. We train the linearizer using SAC [47] with the automatic entropy adjustment [46] for $8e4$ epochs for D'Kitty, $3e5$ epochs for Humanoid and $1e5$ epochs for the others. We apply 4 gradient steps and consider training with and without a replay buffer, where rewards are normalized with their exponential moving average without a buffer and 2048-sized mini-batches are used with a buffer of $1e6$. We set the initial entropy to 0.1, the target entropy to $-dim(\mathcal{A})/2$, the target smoothing coefficient to 0.005 and the

discount rate to 0.99. We choose a prior goal distribution for each environment from $\{\text{Beta}(1,1), \text{Beta}(2,2)\}$. We determine the hyperparameters based on the state coverage of the trained linearizer.

Training of skill discovery methods. We train IBOL and the '-L' variants of other skill discovery methods for $1e4$ epochs with $\ell = 10$, while the methods without the linearizer are trained with the number of transitions that matches the total number of transitions for the training of both the linearizer and each skill discovery method on top of it. We employ SAC for DADS and DIAYN, and the vanilla policy gradient (VPG) for IBOL and VALOR. We set the entropy regularization coefficient to 1e-3 for VALOR and VALOR-L (searched over $\{1e-1, 1e-2, 1e-3, 0\}$), and use the automatic entropy adjustment for DADS with an initial entropy coefficient of $1e-1$, DADS-L with $1e-3$, DIAYN with $1e-1$ and DIAYN-L with $1e-2$ (searched over $\{1e-1, 1e-2, 1e-3\}$ with and without the automatic regularization). For those using VPG, we apply four gradient steps with the entire batch at each epoch. For SAC, we apply 64 gradient steps (or 32 steps for the skill dynamics model in DADS) with 256-sized mini-batches, since increased gradient steps expedite the training by exploiting the off-policy property of SAC. We use $L = 100$ for DADS and IBOL, and set $\lambda = 2$ (searched over $\{0.1, 1, 2\}$) and $\beta = 1e-2$ (searched over $\{1e-1, 1e-2, 1e-3\}$) for IBOL.

Training of meta-controllers for downstream policies. SAC is used for training the meta-controllers. We fix the entropy coefficient to 0.01, and apply four gradient steps with the full-sized batches. The meta-controllers select skill latents in a range of $[-2, 2]$, where they are fed into the learned skill policies.

3.4.2 Visualization of Learned Skills

Figure 3.3 shows that IBOL, with no extrinsic rewards, discovers diverse locomotion skills for Ant and Humanoid and multiple skills with various speeds and poses in both directions for HalfCheetah and Hopper. We present the discovery of orientation primitives for Ant in Section 3.4.8 and additional results including the videos of the discovered skills at `https://vision.snu.ac.kr/projects/ibol`.

Figure 3.1 demonstrates that while all the algorithms mainly discover locomotion skills, IBOL discovers visually less entangled primitives with the most diverse directions compared to the latent-first and trajectory-first baselines. We train IBOL, DIAYN-L, VALOR-L, DADS-L, SeCTAR-L, SeCTAR-L-XY and EDL-L on Ant with the skill latent variables of $d = 2$, where SeCTAR-L-XY is equipped with the x-y *prior* [99]. We qualitatively examine their trajectories in the x-y plane; since the x-y dimensions are interpretable and have a large range of values, they can illustrate the characteristic differences between skill discovery algorithms well. We also train DIAYN-XY, VALOR-XY and DADS-XY to enforce them to discover skills on the x-y plane without the linearizer. We observe that the linearizer significantly improves not only the diversity of trajectories but also the correspondence between skill latents and trajectories by reducing the burden of making transitions in the x-y dimensions.

3.4.3 Information-Theoretic Evaluations

We present the metrics that evaluate the unsupervised skill discovery methods without the need for external tasks. While the quantities between skill latents Z and state sequences $S_{0:T}$ generated with π_{θ_z} are attractive, the high dimensionality of $S_{0:T}$ makes it a less viable choice. One workaround is to examine

only the last states S_T instead of the whole sequences, as S_T still characterizes skills in environments to some degree. That is, we can simply estimate $I(Z; S_T)$ instead of $I(Z; S_{0:T})$ to measure how informative Z is. This can also be viewed as follows: in $I(Z; S_{0:T}) = I(Z; S_T) + \sum_{i=0}^{T-1} I(Z; S_i|S_{i+1:T})$, only the first term $I(Z; S_T)$ is taken into account, as $I(Z; S_i|S_{i+1:T}) = h(Z|S_{i+1:T}) - h(Z|S_{i:T})$ and adding S_i to $S_{i+1:T}$ to the condition would change only little entropy of Z.

We also consider metrics for measuring the disentanglement of Z. We find Do and Tran [32] provide a helpful viewpoint to our evaluation. They suggest that the concept of disentanglement has three considerations: *informativeness*, *separability* and *interpretability*. Informativeness denotes how much information each latent dimension contains about the data, and separability is a concept about *no* information sharing between two latent dimensions on the data. Interpretability considers the alignment between the ground-truth and learned factors. Among them, we do not employ the interpretability measure because the lack of supervision in unsupervised skill discovery prevents achieving a high value [72]. For example, if data points are uniformly distributed in a two-dimensional circle, there can be infinite equally good ways to disentangle the data into two axes. To measure informativeness and separability, we use the SEPIN@k and WSEPIN metrics [32] evaluated for skill latents and the last states. For SEPIN@k, $I(S_T^{(\mathrm{loc})}; Z_i|Z_{\neq i})$ quantifies the amount of information about $S_T^{(\mathrm{loc})}$ contained by Z_i but not $Z_{\neq i}$, and the metric is defined as

$$\mathrm{SEPIN@}k = \frac{1}{k}\sum_{j=1}^{k} I(S_T^{(\mathrm{loc})}; Z_{r_j}|Z_{\neq r_j}), \tag{3.10}$$

where r_j is the dimension index with the j-th largest value of $I(S_T^{(\mathrm{loc})}; Z_i|Z_{\neq i})$ for $i = 1, \ldots, d$. That is, SEPIN@k is the average of the top k values of

$I(S_T^{(\mathrm{loc})}; Z_i|Z_{\neq i})$. They also define WSEPIN as

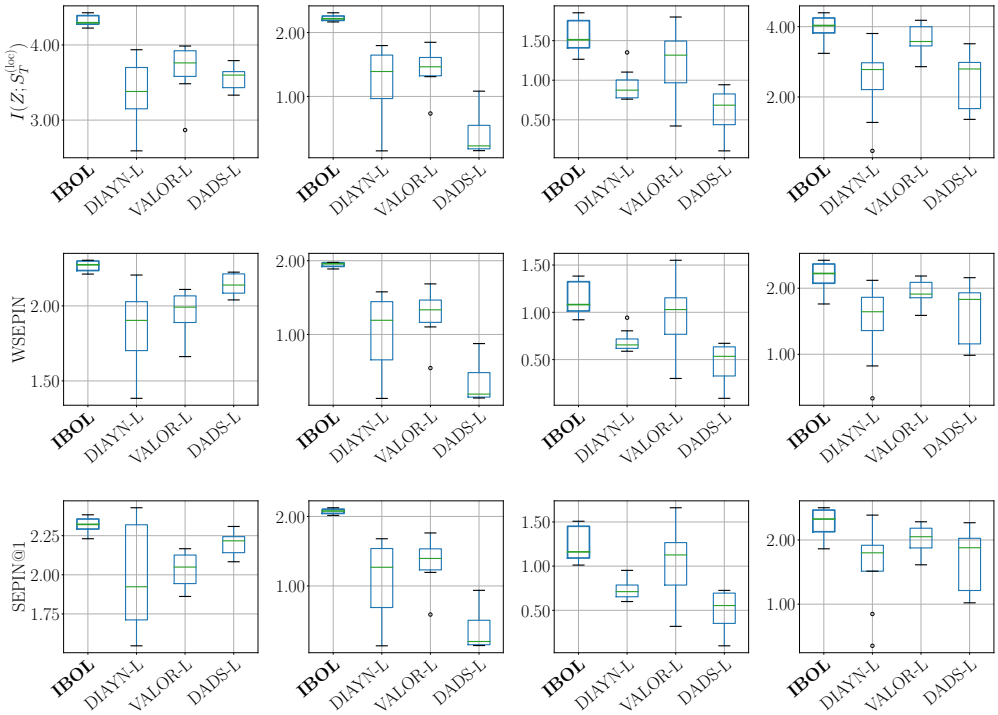
$$\mathrm{WSEPIN} = \sum_{i=1}^{d} \rho_i \cdot I(S_T^{(\mathrm{loc})}; Z_i|Z_{\neq i}) \tag{3.11}$$

for $\rho_i = \frac{I(S_T^{(\mathrm{loc})};Z_i)}{\sum_{j=1}^{d} I(S_T^{(\mathrm{loc})};Z_j)}$. It is the sum of $I(S_T^{(\mathrm{loc})}; Z_i|Z_{\neq i})$ weighted based on their informativeness, $I(S_T^{(\mathrm{loc})}; Z_i)$.

We compare the skill policies trained by IBOL, DIAYN-L, VALOR-L and DADS-L with $d = 2$. We use the three evaluation metrics, $I(Z; S_T^{(\mathrm{loc})})$, SEPIN@1 and WSEPIN on Ant, HalfCheetah, Hopper and D'Kitty, keeping only the state dimensions for the agent's locomotion (*i.e.* x-y coordinates for Ant and D'Kitty and x for the rest) denoted as (loc). One rationale behind it is that the algorithms on the linearized environments successfully discover the locomotion skills (*e.g.* Figure 3.1). The locomotion coordinates are also suitable for assessing skill *discovery*, since these values can vary in large ranges.

For each environment, we employ two pre-trained linearizers, and train every method four times for each linearizer, resulting in eight runs in total. To measure the quantities, we sample 2000 trajectories per run and use quantization, where for each variable we divide the range of the values from all the runs into 32 bins.

Figure 3.5 shows the box plots of the results. With the same linearizers, IBOL outperforms the three baselines, DIAYN-L, VALOR-L and DADS-L, in all three information-theoretic evaluation metrics on Ant, HalfCheetah, Hopper and D'Kitty. The plots for $I(Z; S_T^{(\mathrm{loc})})$ show that IBOL can stably discover diverse skills from the environments conditioned on the skill latent parameter Z. Also, the results with WSEPIN and SEPIN@1 suggest that IBOL outperforms the baselines, with regard to both informativeness and separability of Z's individual dimensions. Overall, IBOL shows the lower average deviation compared



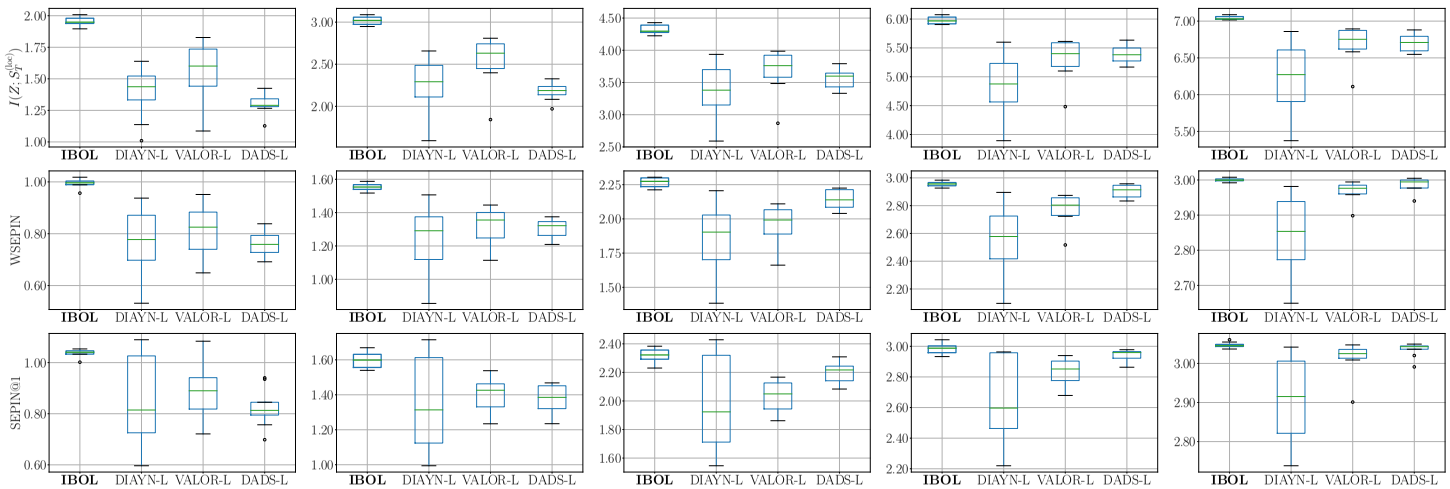
(a) Ant (b) HalfCheetah (c) Hopper (d) D'Kitty

Figure 3.5: Comparison of IBOL (ours) with the baseline methods, DIAYN-L, VALOR-L and DADS-L, in the evaluation metrics of $I(Z; S_T^{(\mathrm{loc})})$, WSEPIN and SEPIN@1, on Ant, HalfCheetah, Hopper and D'Kitty. For each method, we use the eight trained skill policies.

to the other methods, which demonstrates its stability in learning.

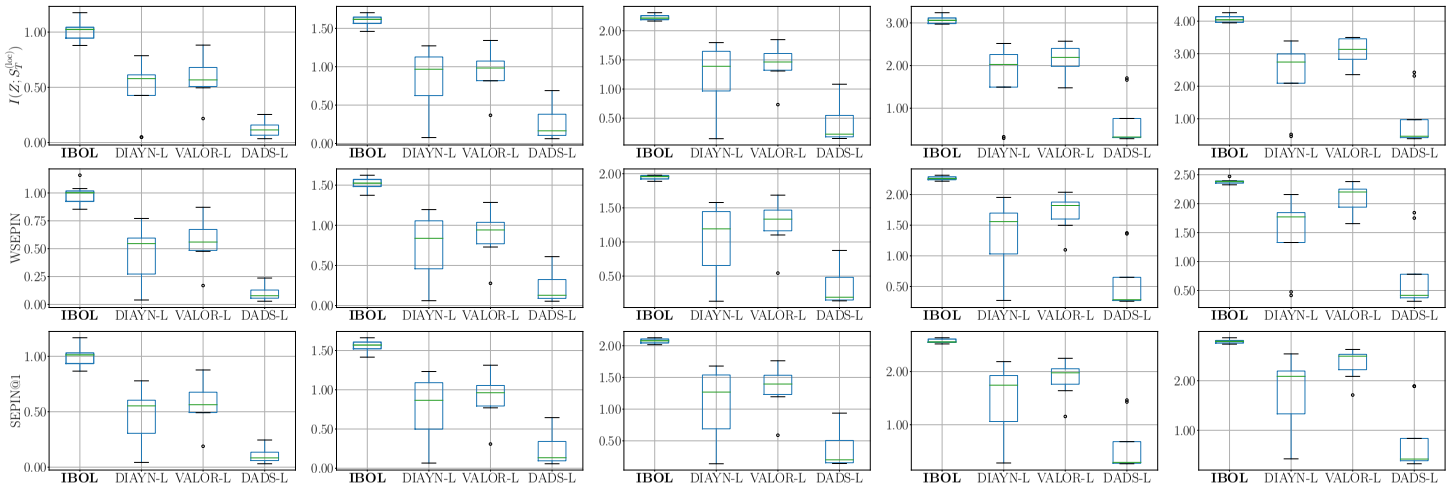
3.4.4 Varying Number of Bins for MI Estimation

We quantize variables for the estimation of mutual information for measuring the information-theoretic metrics in Section 3.4.3. To show that IBOL outperforms the baseline skill discovery methods under different evaluation configurations, we make a more comprehensive comparison between the methods using different numbers of bins for quantization.



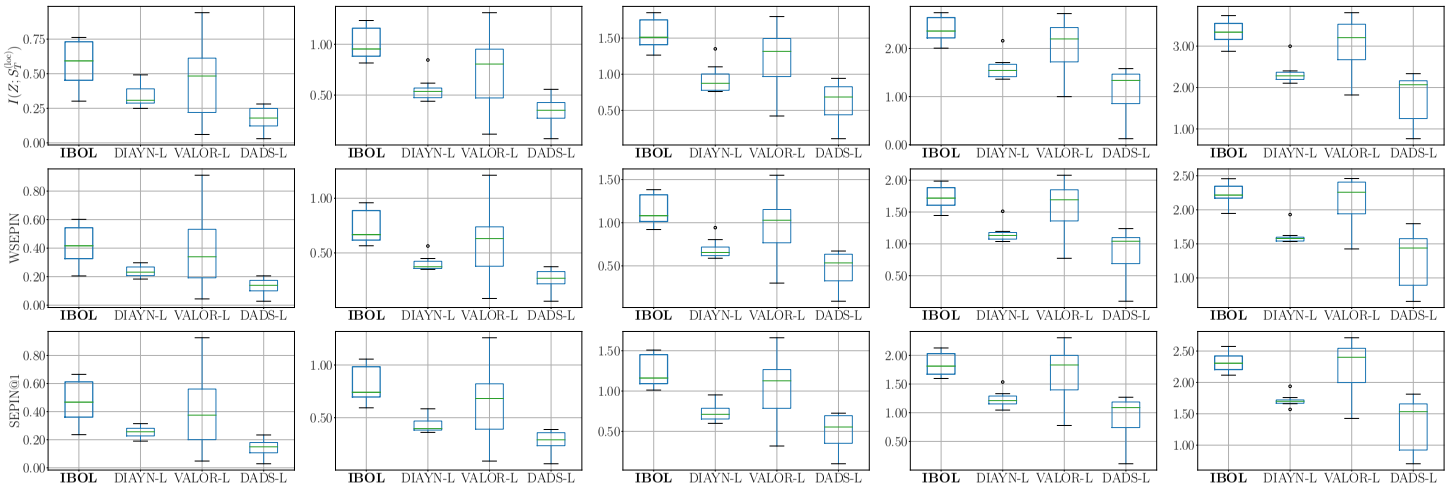
(a) # bins = 8 (b) # bins = 16 (c) # bins = 32 (d) # bins = 64 (e) # bins = 128

Figure 3.6: Comparison of IBOL (ours) with the baseline methods, DIAYN-L, VALOR-L and DADS-L, in the evaluation metrics of $I(Z; S_T^{(\mathrm{loc})})$, WSEPIN and SEPIN@1, on Ant, with different bin counts for the range of each variable estimating mutual information. For each method, we use the eight trained skill policies.



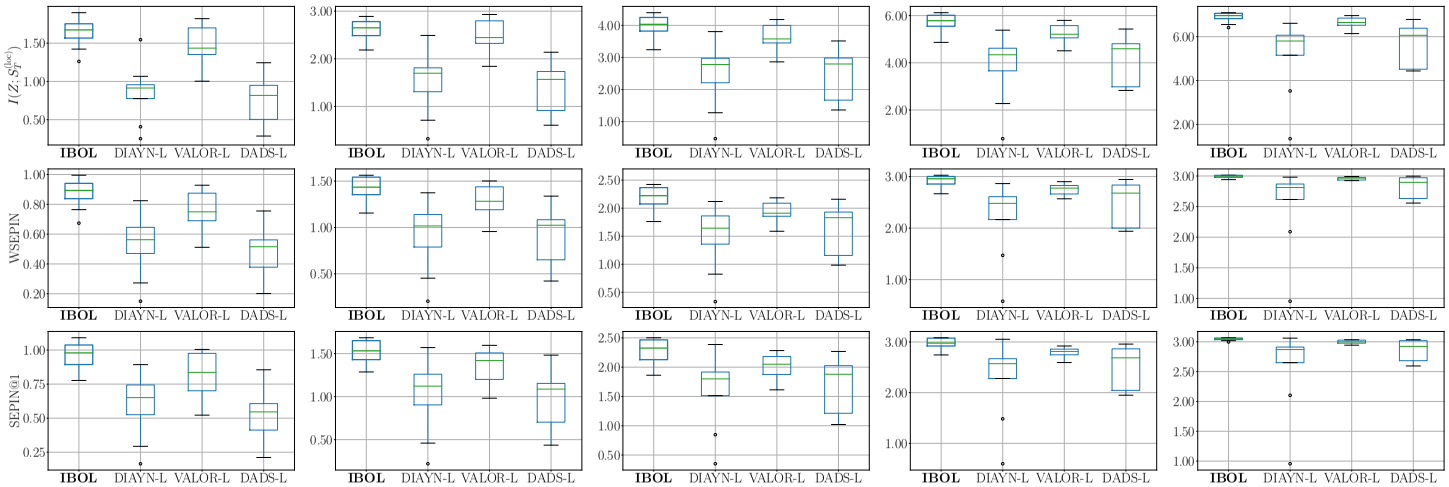
(a) # bins = 8 (b) # bins = 16 (c) # bins = 32 (d) # bins = 64 (e) # bins = 128

Figure 3.7: Comparison of IBOL (ours) with the baseline methods, DIAYN-L, VALOR-L and DADS-L, in the evaluation metrics of $I(Z; S_T^{(\mathrm{loc})})$, WSEPIN and SEPIN@1, on HalfCheetah, with different bin counts for the range of each variable estimating mutual information. For each method, we use the eight trained skill policies.



(a) # bins = 8 (b) # bins = 16 (c) # bins = 32 (d) # bins = 64 (e) # bins = 128

Figure 3.8: Comparison of IBOL (ours) with the baseline methods, DIAYN-L, VALOR-L and DADS-L, in the evaluation metrics of $I(Z; S_T^{(\mathrm{loc})})$, WSEPIN and SEPIN@1, on Hopper, with different bin counts for the range of each variable estimating mutual information. For each method, we use the eight trained skill policies.



(a) # bins = 8 (b) # bins = 16 (c) # bins = 32 (d) # bins = 64 (e) # bins = 128

Figure 3.9: Comparison of IBOL (ours) with the baseline methods, DIAYN-L, VALOR-L and DADS-L, in the evaluation metrics of $I(Z; S_T^{(\mathrm{loc})})$, WSEPIN and SEPIN@1, on D'Kitty, with different bin counts for the range of each variable estimating mutual information. For each method, we use the eight trained skill policies.

Figures 3.6, 3.7, 3.8 and 3.9 compare the skill discovery methods on Ant, HalfCheetah, Hopper and D'Kitty, varying the number of bins for the mutual information estimation. The results show that on all the four environments, IBOL outperforms DIAYN-L, VALOR-L and DADS-L in the evaluation metrics of $I(Z; S_T^{(\mathrm{loc})})$, WSEPIN and SEPIN@1 regardless of binning, which robustly supports IBOL's improved performance.

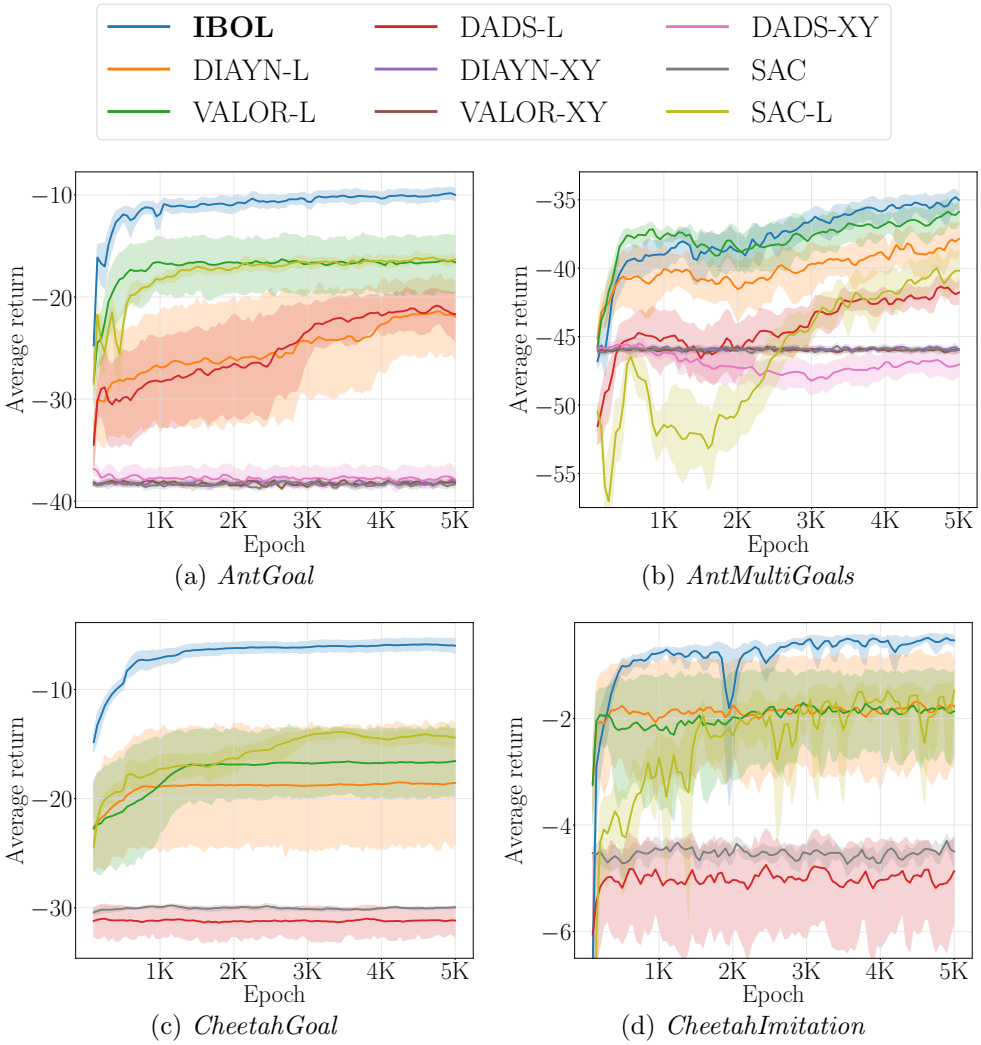
3.4.5 Evaluation on Downstream Tasks

We demonstrate the effectiveness of the abstraction learned by IBOL on downstream tasks. In Ant, we modify the environment to obtain two tasks, *AntGoal* and *AntMultiGoals*, inspired by Eysenbach et al. [36], Sharma et al. [99]. In HalfCheetah, we test the methods on two tasks, *CheetahGoal* and *CheetahImitation*.

AntGoal is a task for evaluating how capable the agent is in reaching diverse goals. For every new episode, a goal $w = [w^{(x)}, w^{(y)}] \in [-50, 50]^2$ is randomly sampled in the x-y plane. The agent can observe the goal w at every step, and receives a reward of $\big(-\|w - [s_T^{(x)}, s_T^{(y)}]\|_2\big)$ where $[s_T^{(x)}, s_T^{(y)}]$ is the agent's final position, when each episode ends.

AntMultiGoals is a repeated version of *AntGoal*. At time step $t \equiv 0 \pmod{\eta}$ in each episode, a new goal $w = [w^{(x)}, w^{(y)}]$ is sampled based on the agent's current position, $[s_t^{(x)}, s_t^{(y)}]$, and is held for the next η steps. Specifically, w is sampled from $[s^{(x)} - 15, s^{(x)} + 15] \times [s^{(y)} - 15, s^{(y)} + 15]$, where $[s^{(x)}, s^{(y)}]$ denotes the agent's position when the goal is about to be sampled. Similarly to *AntGoal*, at the end of each η-sized chunk (before sampling of a new goal), the agent gains a reward of $\big(-\|w - [s_t^{(x)}, s_t^{(y)}]\|_2\big)$. We set $\eta = 50$.

CheetahGoal is a task similar to *AntGoal* but in HalfCheetah. For each



(a) *AntGoal* (b) *AntMultiGoals*

(c) *CheetahGoal* (d) *CheetahImitation*

Figure 3.10: Comparison of IBOL (ours) with the baseline methods on the four downstream tasks. Each line is the mean return over the last 100 epochs at each time step, averaged over eight runs. The shaded areas denote the 95% confidence interval.

episode, a goal $w^{(x)} \in [-60, 60]$ in the x axis is sampled and observed by the agent at every step. At the end of the episode, the agent receives a reward of $\big(-|w^{(x)} - s_T^{(x)}|\big)$ where $s_T^{(x)}$ is the final position of the agent.

We also experiment with a different type of task, *CheetahImitation*. Each of the skill policies learned by the four skill discovery methods is used to sample 1000 random skill trajectories, all of whose x traces are gathered to form a set of imitation targets. For a new episode of *CheetahImitation*, one imitation target $w = [w_1^{(x)}, \ldots, w_T^{(x)}]$, a sequence of T positions in the x axis, is randomly sampled from the set. The goal of this task is to imitate the target w in the x axis; at the t-th step, a reward of $\big(- (w_t^{(x)} - s_t^{(x)})^2 \big)$ is given, where the agent perceives the target w as part of its observation. *CheetahImitation* can evaluate the diversity and coverage of skill policies.

For comparison, we employ a meta-controller on top of each skill policy learned by skill discovery methods. The meta-controller iterates observing a state from the environment and picking a *skill* with its own meta-policy, which invokes the pre-trained skill policy with the same skill latent value z for ℓ_m time steps. We employ Soft Actor-Critic (SAC) [47] to train the meta-controller, and also compare a pure SAC agent as an additional baseline method.

Figure 3.10 compares the performance of IBOL with the baseline methods on the four tasks: *AntGoal*, *AntMultiGoals*, *CheetahGoal* and *CheetahImitation*. We set $\ell_m = 5$ for *AntMultiGoals* and $\ell_m = 20$ for the others. Figures 3.10a and 3.10b suggest that the abstraction by IBOL is more effective for the meta-controller to learn to reach a goal from the initial state, in comparison to the baselines. They confirm that the linearizer greatly helps different skill policies' learning of locomotion in Ant. Figure 3.10c shows that IBOL provides better abstraction to the meta-controller for reaching goals in HalfCheetah. Also, Figure 3.10d demonstrates that IBOL's skills can be used to imitate skills not only from itself but also from the other baselines. It supports the improved diversity of skills learned by IBOL. Overall, IBOL presents significantly smaller variances

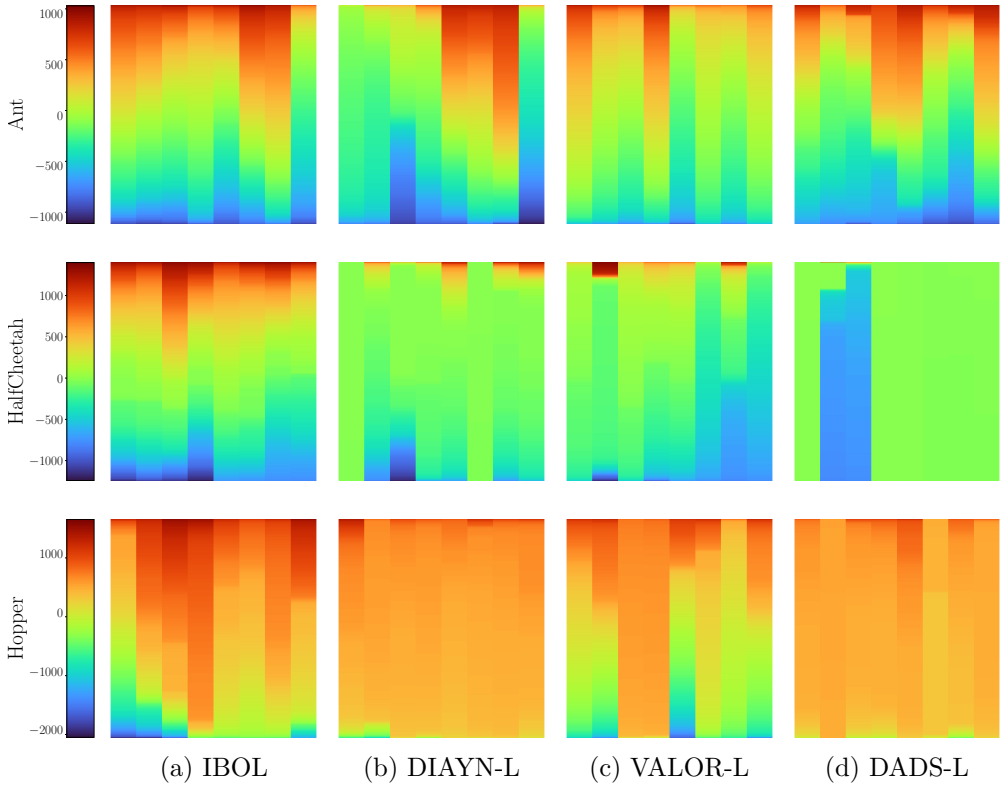


Figure 3.11: Comparison of IBOL (ours) with the baseline methods, DIAYN-L, VALOR-L and DADS-L, in the diversity of external returns for the skills discovered without any rewards. For every method, each of the eight vertical bars visualizes the external returns for 2000 skills sampled randomly with one trained skill policy of the skill discovery method, as a stacked histogram with corresponding colors from the color bar on the left.

than the other baselines.

3.4.6 Diversity of External Returns

We qualitatively demonstrate the diversity of external returns the methods receive for their skills. As the skill discovery methods learn their skill policies without any external rewards, examining their skills in regard to external returns can be used to evaluate the diversity of the skills as well as their usefulness

on the original tasks.

We compare IBOL with DIAYN-L, VALOR-L and DADS-L in Ant, HalfCheetah and Hopper. For every pair of a skill discovery method and an environment, each of the eight skill policies learned by the method with $d = 2$ in the environment is used to sample trajectories given 2000 random skill latents from their prior distribution, $p(z)$ *i.e.* the standard normal distribution. Figure 3.11 visualizes the results, where each vertically stacked histogram denotes the external returns for the 2000 skills with corresponding colors from the color bar for the environment. In the visualizations, the skills learned by IBOL exhibit not only wider but also more diverse ranges of external returns compared to the baseline methods, which suggests that IBOL can acquire a more varied and useful set of skills in the environments.

3.4.7 Comparison of Reward Function Choices for the Linearizer

Prior work on hierarchical reinforcement learning. We first review previous works that train low-level policies similarly to ours. SNN4HRL [40] trains a high-level policy on top of a context-conditioned low-level policy, which is pre-trained with a task-related auxiliary reward function that facilitates the desired behaviors as well as exploration. For example, as a reward for its low-level policy in locomotion tasks, it uses the speed of the agent combined with an information-theoretic regularizer that encourages diversity. FuN [108] jointly trains both a high-level policy and a low-level goal-conditioned policy rewarded by the cosine similarity between goals and directions in its latent space. HIRO [79] takes a similar approach to FuN, but its high-level policy generates goals in the raw state space, without having a separate latent goal space. Its low-level policy is guided by the Euclidean distance instead of the cosine similarity.

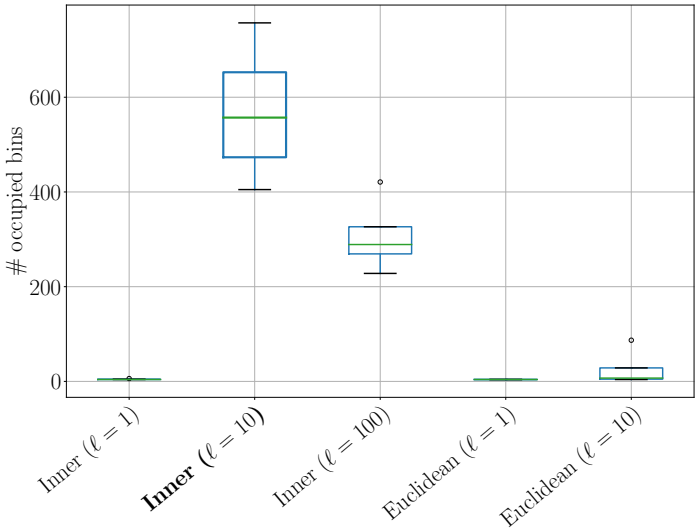


Figure 3.12: Comparison of various reward function choices for the linearizer. The box plot shows the state coverage of each reward function, measured by the number of bins occupied by the 2000 trajectories in the state space. We use four random seeds for each method.

Nachum et al. [80] train a goal-conditioned low-level policy with the Huber loss, which is a variant of the Euclidean distance, in a learned representation space within the framework of sub-optimality.

In contrast to these approaches, we train the linearizer, which can be viewed as a low-level policy, with the reward in the inner-product form. Also, we reward the linearizer with the state difference between *macro* time steps: $(s_{(i+1)\cdot\ell} - s_{i\cdot\ell})$, where ℓ is the interval of the macro step (we use $\ell = 10$ in our experiments).

Comparison of different reward function choices. We now compare our reward function for the linearizer with other choices. We experiment on Ant and evaluate them by their state coverage in the x-y plane. We sample 2000 trajectories from each of the linearizers, where we only change the values of x and y dimensions in the goal space and set the other dimensions' value to 0. We

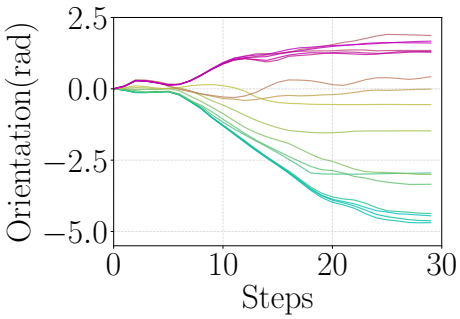
measure the state coverage by the number of bins occupied by the trajectories out of 1024 equally divided bins in the x-y plane. For the comparison, we test different values of $\ell = 1, 10, 100$ with our inner-product reward function, as well as one in the form of the Euclidean distance as in HIRO [79] with $\ell = 1, 10$. Since the Euclidean distance reward function requires the specification of the valid goal ranges, we employ the goal range values used by HIRO. As a consequence, we follow the practice of HIRO to exclude the state dimensions for velocities in specifying the goal space for the Euclidean distance reward function. On the contrary, we use the full state dimensions to design the goal space for the inner-product reward function.

Figure 3.12 compares the performances of the reward function choices. It suggests that using an appropriate size of the macro step (*i.e.* $\ell = 10$) improves the state coverage, especially exhibiting drastic performance improvement over the case of $\ell = 1$. We also observe that our inner-product reward function shows a better state coverage compared to the Euclidean reward function.

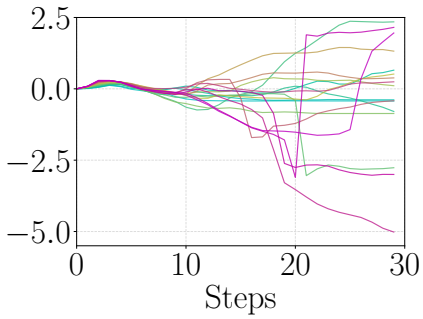
3.4.8 Additional Observations

We present more experiments on Ant to confirm that IBOL can pick appropriate goals at different states for the linearizer in order to learn skills with high distinguishability.

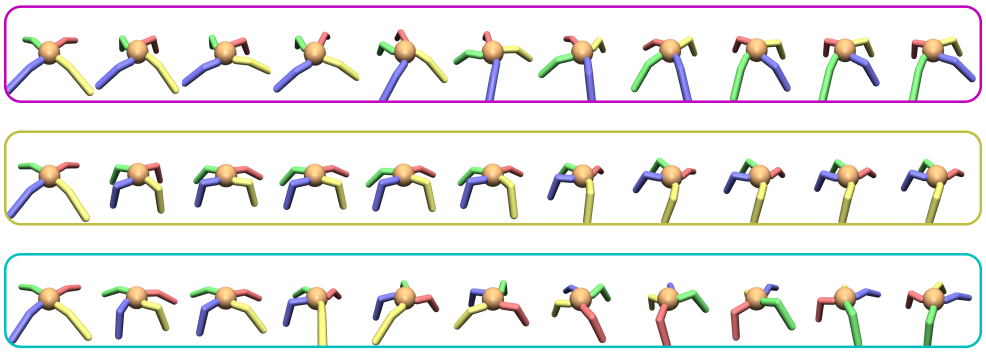
Learning non-locomotion skills. In the absence of locomotion signals (*i.e.*, the x and y dimensions), IBOL can learn orientation primitives, which is not easy unless the skill discovery algorithm produces diverse goals for the linearizer. Figure 3.13a shows examples of orientation skills by IBOL on Ant with $d = 1$. Figure 3.13b depicts that using the linearizer alone fails to produce comparable results, while IBOL utilizes various goal dimensions of the linearizer



(a) IBOL



(b) Linearizer only

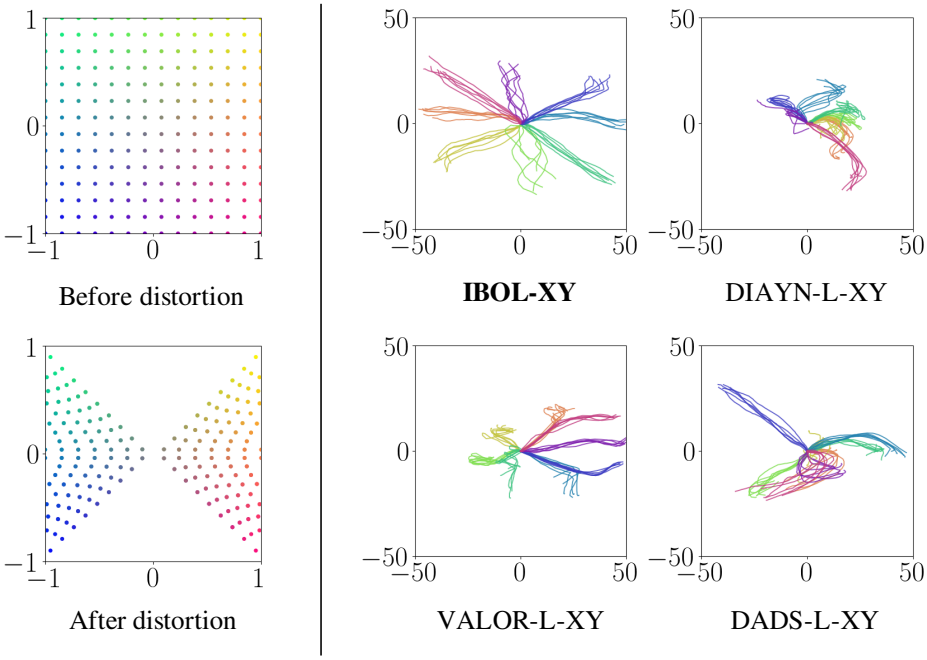


(c) Rendered scenes

Figure 3.13: Orientation trajectories from (a) the *skill policy* of IBOL and (b) the *linearizer*. The skill latent value is interpolated from -4 (cyan) to 4 (magenta) for IBOL, while the orientation dimension value of the goal is interpolated from -1 (cyan) to 1 (magenta) for the linearizer (since it is trained with the goal range of $[-1, 1]$). (c) Rendered scenes of IBOL's trajectories from (a).

to obtain an interpolable skill set.

Overcoming goal space distortion. We conduct additional experiments to validate IBOL's capability of discovering more discriminable trajectories even under harsh conditions. We distort the linearizer's goal space as Figure 3.14a, so that reaching vertically distant states becomes more demanding. We train IBOL-XY, DIAYN-L-XY, VALOR-L-XY and DADS-L-XY with $d = 2$ on top of the modified linearizer. Figure 3.14b suggests that IBOL discovers locomo-



(a) Distortion scheme (b) Visualization of x-y traces

Figure 3.14: (a) Distortion scheme of the linearizer. It distorts the x and y dimensions of goals, and produces the corresponding actions for the modified goals. (b) Visualization of the x-y traces of the skills discovered by each algorithm using the distorted linearizer. The same skill latents are used with the top row of Figure 3.1.

tion skills in various angles including vertical directions in the most visually disentangled manner.

3.4.9 Ablation Study

In this section, we demonstrate the effect of each hyperparameter of IBOL by showing qualitative results on a synthetic environment named *PointEnv*, which is suitable for clear illustrations. In PointEnv, a state $s \in \mathbb{R}^2$ is defined as the x-y coordinates of the agent (point), and an action $a \in [-0.1, 0.1]^2$

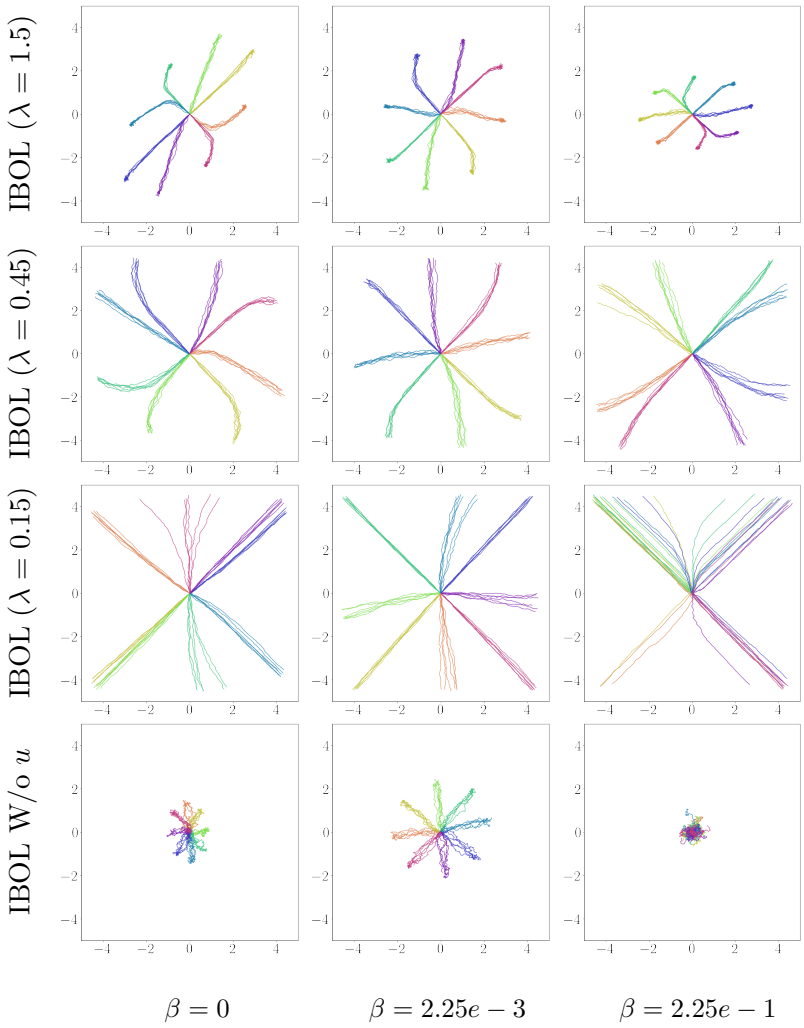


Figure 3.15: Visualization of the x-y traces of the skills discovered by IBOL in PointEnv with various hyperparameter settings. The fourth row corresponds to IBOL without u and the auxiliary term, modelling π_{θ_s} as a LSTM policy. The same skill latents are used with the top row of Figure 3.1.

indicates a vector by which the agent moves. The initial state is sampled from $[-0.05, 0.05]^2$ uniformly at random. As PointEnv is already linearized, we do not use the linearizer for IBOL as well as other baseline methods. We also

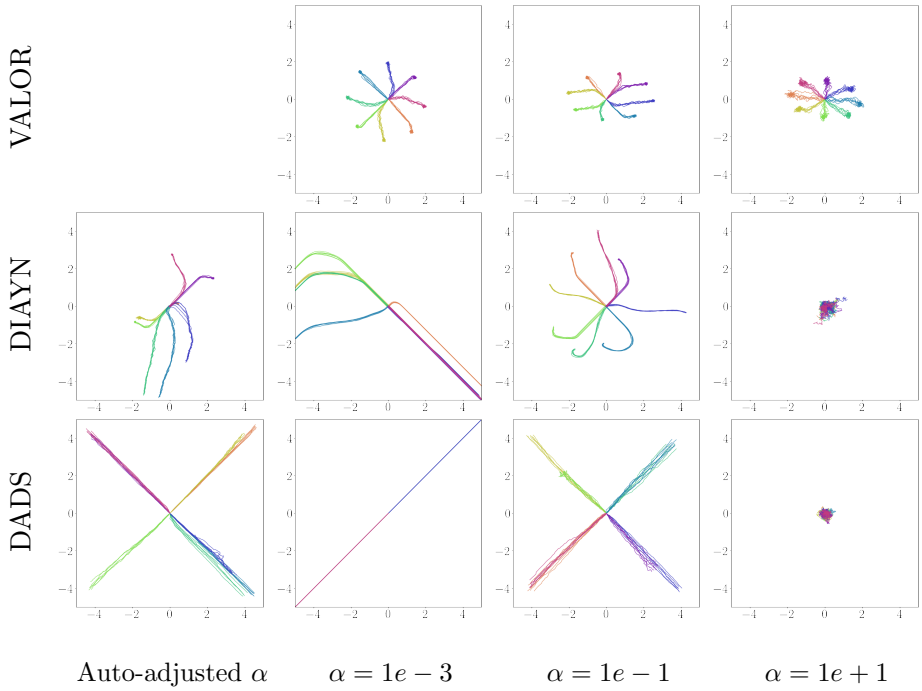


Figure 3.16: Visualization of the x-y traces of the skills discovered by VALOR, DIAYN and DADS in PointEnv with various hyperparameter settings. The same skill latents are used with the top row of Figure 3.1.

reduce the common dimensionality of the neural networks to 32 in lieu of 512. We train IBOL, DIAYN, VALOR and DADS for $5e3$ epochs with an episode length of 50 and a learning rate of $3e-4$, having two-dimensional skill latents with various hyperparameter settings on this environment. For IBOL, we test $\beta \in \{2.25e-1, 2.25e-3, 0\}$ and $\lambda \in \{1.5, 0.45, 0.15\}$, and we also consider the setting without the auxiliary parameter u for the sampling policy π_{θ_s}, in which we model the sampling policy as an LSTM policy (instead of a non-recurrent policy) to compensate for the reduced expressiveness that comes from the dropping of u. We examine the entropy regularization coefficient $\alpha \in \{1e+1, 1e-1, 1e-3\}$ for VALOR, DADS and DIAYN, and we test the automatic

entropy regularization for SAC [46] as well for the latter two.

Figures 3.15 and 3.16 illustrate the x-y traces of the skills discovered by each method with various settings. First, we observe that an appropriate value of β (especially $\beta = 2.25e - 3$ in Figure 3.15) for IBOL helps discover more disentangled and evenly distributed skills. Also, since the auxiliary term $\mathbb{E}_{u\sim p(u),\tau\sim p_{\theta_s}(\tau|u)}[\lambda \cdot p_\phi(u|s_{0:T})]$ encourages IBOL to discover skills that can be easily reconstructed from the trajectories, increasing λ results in having relatively condensed trajectories. The fourth row of Figure 3.15 shows that IBOL can still discover visually disentangled (yet slightly noisy) skills in the absence of u and the auxiliary term. Figure 3.16 presents that for the baseline methods, overly increasing α could result in collapsing while having a moderate value of α improves the quality of discovered skills.

3.5 Summary

We presented Information Bottleneck Option Learning (IBOL) as a novel unsupervised skill discovery method. It first deals with the environment dynamics using the linearizer trained to transition in various directions in the state space. It then discovers skills taking advantage of the information bottleneck framework, which learns the skill latent parameter (or the parameter of the skill policy) as the learned representations of the skills. Our quantitative evaluation showed that the skill latent learned by IBOL provides improved abstraction measured as the disentanglement. We confirmed that IBOL outperforms other skill discovery methods with notably lower variances and the linearizer benefits both IBOL and other baselines on downstream tasks.

Also, there could be some possibilities to extend the skill-learning framework of IBOL to slightly different settings. For instance, one may consider the

case where a dataset of pre-collected trajectories is available. It might allow the pre-training of the trajectory encoder, which could be adopted in the online skill-learning phase that follows, and employing feature learning methods such as Drop-Bottleneck (Chapter 2) may help to improve the efficiency of the learning by reducing the dimensionality of intermediate features and caching the features. We leave it for future investigation to extend IBOL to such settings.

Chapter 4

Test-Time Improvement with Source Approximation

4.1 Overview

For sequential decision making, deep reinforcement learning (RL) has been shown to be effective for various types of problems including games [94] and robotics [58, 70]. With such great successes, interest in multi-task RL has also surged, where its goal is to train a single agent that can efficiently solve multiple varying tasks. In multi-task RL, we focus on the *transfer learning* setting, where the agent learns shared structural knowledge from a set of source tasks during training, and exploits and generalizes them in new, unseen target tasks at test time.

One popular approach to transfer in RL is to leverage the successor features (SFs) framework [1, 14, 15, 21, 82], which transfers policies learned on source tasks to target tasks, where the tasks share the same environment dynamics

but differ in their reward functions. Successor features build a representation of value functions decoupled from reward functions, and transfer to the tasks with arbitrary reward functions by taking an inner product with corresponding task vectors. They utilize generalized policy improvement (GPI) [14], which generalizes policy improvement with multiple policies and provides the performance lower bounds for GPI policies.

However, GPI does not take into account any information from the *smoothness* of optimal action-value functions with respect to task vectors. Tackling this issue, Borsa et al. [21] propose universal successor features approximators (USFAs), which can estimate the optimal successor features for novel task vectors. Nevertheless, the function approximator can make high approximation errors on the task vectors, especially when the new task vectors are distant from the source task vectors. For instance, when USFAs are trained with source tasks to get close to given goals, they may not generalize well to the target tasks where the agent should get away from the given goals. That is, if the elements of target task vectors have the opposite signs from the source task vectors, USFAs could output successor features with high approximation errors.

To improve the successor features approximation of USFAs for the new tasks, we aim at bounding value approximation errors on the new target tasks. We first introduce a new theorem on bounding the optimal values for the tasks that are expressible as linear combinations of source tasks. Our theorem generalizes the conical combination condition used by the prior theorem by Nemecek and Parr [82]. Using our new bounds as constraints, we can train the successor features approximators whose action-value approximation errors on novel tasks are bounded. We extend this idea so that we accomplish a similar effect with no additional training; as a result, we propose *constrained GPI* as a test-time

approach to bounding the approximation errors. Despite its simplicity and no need for modification to the training procedure, we empirically show that constrained GPI attains large performance improvements compared to the original GPI in multiple environments, including the Scavenger [16, 17] and Reacher [22] environments with state observations and the DeepMind Lab [15, 18, 21] environment with first-person visual observations.

Our main contributions can be summarized as follows:

- We present a novel theorem on lower- and upper-bounding optimal values for novel tasks that can be expressed as *linear* combinations of source tasks. It extends and generalizes the previous theorem for *conical* combinations by Nemecek and Parr [82], to enable a broader application of the bounds.
- Based on our new theorem, we propose *constrained GPI* as a simple test-time approach that can improve transfer to novel tasks by constraining action-value approximation errors on new target tasks, with no modification to the training procedure.
- We empirically show that our approach can improve the performance over the GPI baselines by large margins in the Scavenger, Reacher and DeepMind Lab environments. We also provide analyses for a better understanding of our results.

4.2 Related Work

Transfer in reinforcement learning aims at solving a new target task with no additional learning or sample-efficiently by exploiting agents and information

obtained from source tasks. We review a line of research with relevant approaches.

Transfer by reusing policies. This group of approaches reuses policies learned on source tasks for target tasks. Fernández and Veloso [37] suggest an exploration strategy for the learning of a new policy given a new task and learned source policies, where the gain of using each policy is estimated together on-line and one of the policies in the set is selected probabilistically at each step, based on the gain, but they focus on aiding the training of the target policy with samples from the target task rather than improving the zero-shot transfer performance. On the other hand, Dayan [31] introduce successor representations (SRs), state space occupancy representations disentangled from rewards, which allow linear decomposition of value functions. Barreto et al. [14] propose successor features (SFs) and using SFs as an extension of SRs. Especially, in addition to SFs, which can be combined with arbitrary task vectors for obtaining the corresponding values, Barreto et al. [14] also suggest generalized policy improvement (GPI), allowing composition of multiple source policies on a single task where the resulting GPI policy performs at least as well as any of the source policies. The transfer with SFs and GPI and a number of connected methods [15, 17] combine source policies using the reward decomposition structure but are limited in that they do not further make use of the smoothness of the optimal value functions with respect to different tasks *i.e.*, given two similar tasks, their values are likely to be close to each other. Nemecek and Parr [82] maintain a set of policies and determine whether to learn a new one for a given task or exploit existing ones based on their optimal value bounds, which is a different problem from ours, and they also target tasks expressible as conical combinations of source tasks, whereas we target linear combinations. Alver and Precup [12] suggest sets of assumptions and conditions under which

a group of basis policies can induce GPI policies that maximizes undiscounted returns on novel tasks and an iterative algorithm for constructing the basis, but thus they tackle a different problem, many of whose requirements do not apply to more general settings or our problem, due to *e.g.*, (non-binary) continuous reward features ϕ, no guarantee on possible trajectories, stochastic transition dynamics, etc. Alegre et al. [10] deal with the policy set construction problem as well by interpreting the SFs framework as the learning of multiple policies in the multi-objective RL problem and extending the optimistic linear support algorithm [89] for the SFs framework. Differently from the problem we are tackling, in the maximum entropy setting Hunt et al. [52] aim at compositing source policies for target tasks optimally by estimating and correcting the divergence between source policies, but they consider target tasks whose rewards are convex combinations of source task rewards and the divergence estimation becomes significantly harder when there are more than two source tasks [52].

Transfer with function approximation. There is a series of studies that directly exploits the smoothness of optimal values across tasks with function approximators. Schaul et al. [93] propose universal value functions and their approximators, which incorporate goals into their input, and use the approximators for generalization to novel goals. Inspired by [93], Borsa et al. [21] suggest universal successor features approximators (USFAs), which allow the use of GPI with arbitrary approximate policies by extending the original SFs approximators to take policy vectors as their inputs. However, the generalization to novel or distant policy vectors with the function approximators could result in SFs with large approximation errors. In this work, we tackle this problem by leveraging the reward decomposition structure for bounding estimated values around optimal values and thus their errors. On the other hand, Hong et al. [51] propose bilinear value networks (BVN) with a bilinear decomposition of

value functions into goal-agnostic and goal-specific components for better sample efficiency in multi-goal reinforcement learning. They provide an alternative formulation of value function approximators differently from the decomposition used in the SFs framework, and thus our approach of bounding approximate values at test time is orthogonal to BVN.

Compared to the existing work mentioned above in general, our constrained GPI is a simple test-time approach for target tasks which are expressible as linear combinations of source tasks.

4.3 Preliminaries

We describe the problem setting and background on successor features and universal successor features approximators.

4.3.1 The Zero-Shot Transfer Problem in RL

We define a Markov Decision Process (MDP) as $M \equiv (\mathcal{S}, \mathcal{A}, P, R, \gamma)$. $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, respectively. $P(\cdot|s,a)$ defines the transition probability distribution of the next states given $s \in \mathcal{S}$ and $a \in \mathcal{A}$. $R(s,a,s')$ is the reward for taking action a at state s resulting in s', and $\gamma \in (0,1]$ is the discount factor. We assume that rewards are bounded.

We consider the zero-shot transfer problem; as in [14], each task is defined by its task vector $\boldsymbol{w} \in \mathbb{R}^d$, and only the reward functions differ across tasks, being decomposed as

$$R_{\boldsymbol{w}}(s,a,s') = \phi(s,a,s')^\top \boldsymbol{w}, \tag{4.1}$$

where $\phi(s,a,s') \in \mathbb{R}^d$ is the features of (s,a,s'). We denote the set of source task vectors as $\mathcal{T}$, which is used for training. At test time, we evaluate the

transferred policy on each target task $\boldsymbol{w}' \notin \mathcal{T}$ with no additional update of pre-trained components. We examine both the possible scenarios: (i) the features $\phi(s, a, s')$ are available to the agent [14, 21] and (ii) no pre-defined features are available and the agent needs to construct its own features and task vectors. We first introduce the formulation for (i) in Section 4.3.2 and then its variant for (ii) in Section 4.3.3.

4.3.2 Successor Features and Universal Successor Features Approximators

We now review successor features (SFs) [14] and how they are transferred to different tasks. Equation (4.1) allows expressing the action-value function for policy π on task $\boldsymbol{w}$ as

$$Q^{\pi}_{\boldsymbol{w}}(s, a) = \mathbb{E}^{\pi}\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} \middle| S_t = s, A_t = a\right] \tag{4.2}$$

$$= \mathbb{E}^{\pi}\left[\sum_{i=0}^{\infty} \gamma^i \phi_{t+i} \middle| S_t = s, A_t = a\right]^{\top} \boldsymbol{w} = \psi^{\pi}(s, a)^{\top} \boldsymbol{w}, \tag{4.3}$$

where $\phi_t = \phi(s_t, a_t, s_{t+1}) \in \mathbb{R}^d$. Here, $\psi^{\pi}(s, a) \in \mathbb{R}^d$ is called the SFs for policy π at (s, a), and taking its inner product with an arbitrary task $\boldsymbol{w}$ results in the action-value for π on $\boldsymbol{w}$; *i.e.*, $Q^{\pi}_{\boldsymbol{w}}$(s, a). Thanks to the analogy between (rewards r, action-value functions Q) and (features ϕ, successor features ψ), the Bellman equation applies to SFs and thus they can be trained similarly to the way action-value functions are learned; *e.g.*, Q-learning.

The policy improvement theorem [19] states that a new policy that takes a greedy action according to a given policy's value function at each state performs at least as well as the original policy. Generalized policy improvement (GPI) [14] extends policy improvement to the case where the value functions of multiple policies are available. Given a task $\boldsymbol{w}'$, a set of policies $\pi_1, \ldots, \pi_n$, their action-

value functions $Q^{\pi_1}_{\boldsymbol{w}'}, \ldots, Q^{\pi_n}_{\boldsymbol{w}'}$ and their approximations $\tilde{Q}^{\pi_1}_{\boldsymbol{w}'}, \ldots, \tilde{Q}^{\pi_n}_{\boldsymbol{w}'}$, the GPI policy is defined as

$$\pi_{\text{GPI}}(s) \in \operatorname*{argmax}_a \max_i \tilde{Q}^{\pi_i}_{\boldsymbol{w}'}(s,a). \tag{4.4}$$

The GPI theorem by Barreto et al. [14] suggests that $Q^{\pi_{\text{GPI}}}_{\boldsymbol{w}'}(s,a) \geq \max_i Q^{\pi_i}_{\boldsymbol{w}'}(s,a) - \frac{2}{1-\gamma} \max_i \left\| Q^{\pi_i}_{\boldsymbol{w}'} - \tilde{Q}^{\pi_i}_{\boldsymbol{w}'} \right\|_\infty$. They also provide the upper bound on the suboptimality of the GPI policy as

$$\left\| Q^*_{\boldsymbol{w}'} - Q^{\pi_{\text{GPI}}}_{\boldsymbol{w}'} \right\|_\infty \leq \frac{2}{1-\gamma} \left\{ \min_i \|\phi\|_\infty \|\boldsymbol{w}' - \boldsymbol{w}_i\| + \max_i \left\| Q^{\pi_i}_{\boldsymbol{w}'} - \tilde{Q}^{\pi_i}_{\boldsymbol{w}'} \right\|_\infty \right\},$$

where each π_i is an optimal policy for $\boldsymbol{w}_i$.

While the GPI theorem allows the transfer of learned successor features to arbitrary tasks that share the same environment dynamics, it is limited in the following aspect. GPI uses the action-values for source tasks on target tasks based on the reward decomposition assumption (Equation (4.1)) *i.e.*, $\tilde{Q}^{\pi_i}_{\boldsymbol{w}'}(s,a) = \tilde{\psi}^{\pi_i}(s,a)^\top \boldsymbol{w}'$ for each i. However, it does not take any advantage of the *smoothness* of the optimal action-value functions with respect to different task vectors [21].

To overcome this limitation, Borsa et al. [21] introduce universal successor features approximators (USFAs). Inspired by universal value functions (UVFs) [93], they extend the original successor features with policy vectors $\boldsymbol{z} \in \mathbb{R}^l$ as input to their approximators. More specifically, universal successor features (USFs) are defined to satisfy

$$\psi^{\pi_{\boldsymbol{z}}}(s,a) \;\equiv\; \psi(s,a,\boldsymbol{z}) \;\approx\; \tilde{\psi}(s,a,\boldsymbol{z}), \tag{4.5}$$

where $\boldsymbol{z}$ is a policy vector for the policy $\pi_{\boldsymbol{z}}$, and USFAs $\tilde{\psi}$ are the learned approximators of USFs ψ. Naturally, the value functions are expressed as

$$\tilde{Q}^{\pi_{\boldsymbol{z}}}_{\boldsymbol{w}}(s,a) = \tilde{\psi}(s,a,\boldsymbol{z})^\top \boldsymbol{w} \approx \psi(s,a,\boldsymbol{z})^\top \boldsymbol{w} = Q^{\pi_{\boldsymbol{z}}}_{\boldsymbol{w}}(s,a). \tag{4.6}$$

Each reward function induces optimal policies, which can be encoded using the corresponding task vectors. That is, one can simply choose to define the policy vector space to be the same as the task vector space ($l = d$) and let $\boldsymbol{z} = \boldsymbol{w}$ be a policy vector of an optimal policy for task $\boldsymbol{w}$. Then, $\pi_{\boldsymbol{w}}$ and $Q^{\pi_{\boldsymbol{w}}}_{\boldsymbol{w}}$ denote an optimal policy for $\boldsymbol{w}$ and its action-value function, respectively.

The training of USFAs is similar to that of SFs, except for that it additionally involves sampling of policy vectors given task vectors. The update of USFAs at the k-th iteration is

$$\tilde{\psi}^{(k+1)} \leftarrow \operatorname*{argmin}_{\psi} \mathbb{E}_{\boldsymbol{w}\sim\mathcal{T},\boldsymbol{z}\sim\mathcal{D}_{\boldsymbol{z}}(\cdot|\boldsymbol{w}),(s,a,s')\sim\mu} \left[\left\| \delta(s,a,s',\boldsymbol{z}) \right\|^2 \right] \tag{4.7}$$

$$\text{for } \delta(s,a,s',\boldsymbol{z}) = \phi(s,a,s') + \gamma\tilde{\psi}^{(k)}(s',a',\boldsymbol{z}) - \psi(s,a,\boldsymbol{z}) \text{ and}$$

$$a' = \operatorname*{argmax}_{b} \tilde{\psi}^{(k)}(s',b,\boldsymbol{z})^{\top}\boldsymbol{z}.$$

$\mathcal{D}_{\boldsymbol{z}}(\cdot|\boldsymbol{w})$ is the policy vector distribution; for instance, $\mathcal{N}(\boldsymbol{w}, \sigma I)$ can be used for better training with diversified inputs. μ is the transition sampling distribution, which involves the GPI policy of the samples from $\mathcal{D}_{\boldsymbol{z}}(\cdot|\boldsymbol{w})$ or a replay buffer. We use gradient descent to update the parameters.

USFAs provide a benefit that they allow a GPI policy to use an arbitrary set of policies $\{\pi_{\boldsymbol{z}}\}_{\boldsymbol{z}\in\mathcal{C}}$ as $\pi_{\text{GPI}}(s) \in \operatorname{argmax}_a \max_{\boldsymbol{z}\in\mathcal{C}} \tilde{Q}^{\pi_{\boldsymbol{z}}}_{\boldsymbol{w}'}(s,a)$. However, the generalization of USFAs to new policy vectors depends on a function approximator ψ, and thus if $\mathcal{C}$ contains policy vector(s) distant from source vectors, a GPI policy with $\mathcal{C}$ may have high approximation errors and perform poorly or even worse than a GPI policy with only source vectors [21], as will be demonstrated later in our experiments.

4.3.3 Universal Successor Features Approximators with Learned ϕ

For the scenario where features ϕ's are not provided to the agent[1], we adopt the problem formulation from Ma et al. [73] where for each task the task information $\boldsymbol{g} \in \mathcal{G}$ is given to the agent. Although the task information $\boldsymbol{g}$, unlike a task vector, cannot be directly combined with successor features for transfer to a novel task, zero-shot inference could still be possible by leveraging the information about the task.

Specifically, we not only perform the original learning of $\tilde{\psi}$ letting the task information induce policy vectors instead, but also train $\tilde{\phi}$ and $\tilde{\boldsymbol{w}}$ to approximate the reward decomposition with transition samples. As done in [73], we update $\tilde{\psi}$, $\tilde{\phi}$ and $\tilde{\boldsymbol{w}}$ using gradient descent to minimize $\mathbb{E}_{\boldsymbol{g}\sim\mathcal{T}^{\boldsymbol{g}},\boldsymbol{z}\sim\mathcal{D}_{\boldsymbol{z}}^{\boldsymbol{g}}(\cdot|\boldsymbol{g}),(s,a,r,s')\sim\mu}\left[\mathcal{L}^{\psi}+\mathcal{L}^{Q}\right]$ for

$$\mathcal{L}^{\psi} := \frac{1}{d}\left\|\tilde{\phi}(s,a,s') + \gamma\tilde{\psi}^{(k)}(s',a',\boldsymbol{z}) - \tilde{\psi}(s,a,\boldsymbol{z})\right\|^2 \tag{4.8}$$

$$\mathcal{L}^{Q} := \left\{r + \gamma\tilde{\psi}^{(k)}(s',a',\boldsymbol{z})^{\top}\tilde{\boldsymbol{w}}^{(k)}(\boldsymbol{z}) - \tilde{\psi}(s,a,\boldsymbol{z})^{\top}\tilde{\boldsymbol{w}}(\boldsymbol{z})\right\}^2 \tag{4.9}$$

and $a' = \operatorname{argmax}_b \tilde{\psi}^{(k)}(s',b,\boldsymbol{z})^{\top}\tilde{\boldsymbol{w}}^{(k)}(\boldsymbol{z})$ at the k-th iteration. The superscript (k) denotes the target, $\mathcal{T}^{\boldsymbol{g}}$ is the source task information set, $\mathcal{D}_{\boldsymbol{z}}^{\boldsymbol{g}}(\cdot|\boldsymbol{g})$ is the policy vector distribution conditioned on the task information and μ is the sampling distribution.

4.4 Constrained GPI for Improved Zero-Shot Transfer of Successor Features

[1]One typical example presented later in our experiments is the case where the agent observes visual inputs. Then, it is not trivial to derive features that linearly decompose reward functions.

To mitigate the aforementioned issue of the possibly unlimited approximation errors of USFAs, we propose a simple yet effective method that improves the transfer of successor features by further leveraging the reward decomposition structure in Equation (4.1). We first present, under a more relaxed condition, lower and upper bounds on the optimal values for novel task vectors that are expressed as linear combinations of source task vectors (Section 4.4.1). Then, we propose a novel approach called *constrained GPI*, which effectively confines the approximated action-values inside the computed lower and upper bounds (Section 4.4.2).

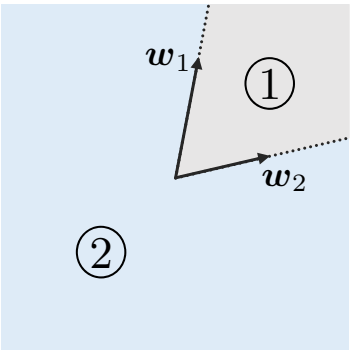


Figure 4.1: An example comparison of task space coverage. While the conical combinations of $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ covers only area ①, the linear combinations covers both area ①–②.

4.4.1 Bounding Optimal Values for New Tasks

Theorem 1 of [82] provides the lower and upper bounds on the value of an optimal policy for a new task, whose vector $\boldsymbol{w}'$ is a *positive conical combination* of source task vectors *i.e.*, $\boldsymbol{w}' = \sum_{\boldsymbol{w}\in\mathcal{T}} \alpha_{\boldsymbol{w}} \boldsymbol{w}$ such that $\alpha_{\boldsymbol{w}} \geq 0, \forall \boldsymbol{w} \in \mathcal{T}$ and $\sum_{\boldsymbol{w}\in\mathcal{T}} \alpha_{\boldsymbol{w}} > 0$[2]. However, for a broad application of such bounds, the positive conical combination condition can be too restrictive, since the resulting bounds only apply to the task vectors that appear inside the conical hull of source task vectors.

Therefore, we suggest a more relaxed theorem, which holds for an arbitrary task vector $\boldsymbol{w}'$ that is expressed as a linear combination of the source task

[2]We slightly abuse the notation and let $\alpha_{\boldsymbol{w}}$ denote the coefficient for vector $\boldsymbol{w}$.

vectors *i.e.*, $\boldsymbol{w}' = \sum_{\boldsymbol{w}\in\mathcal{T}} \alpha_{\boldsymbol{w}}\boldsymbol{w}$ for $\alpha_{\boldsymbol{w}} \in \mathbb{R}, \forall \boldsymbol{w} \in \mathcal{T}$. Figure 4.1 shows an example that compares the task space coverage of conical [82] and our linear combinations. With our extended task space coverage, we can apply the bounds to more general target tasks outside of the conical hull, which will be further discussed in the next section.

We define $\epsilon_{\boldsymbol{w}_2}^{\pi_{\boldsymbol{w}_1}}$ to be an upper bound on the approximation error of $\tilde{Q}_{\boldsymbol{w}_2}^{\pi_{\boldsymbol{w}_1}}$ for arbitrary tasks $\boldsymbol{w}_1, \boldsymbol{w}_2$ such that

$$|Q_{\boldsymbol{w}_2}^{\pi_{\boldsymbol{w}_1}}(s,a) - \tilde{Q}_{\boldsymbol{w}_2}^{\pi_{\boldsymbol{w}_1}}(s,a)| \leq \epsilon_{\boldsymbol{w}_2}^{\pi_{\boldsymbol{w}_1}}(s,a), \quad \forall (s,a) \in \mathcal{S}\times\mathcal{A}, \tag{4.10}$$

and we present our theorem as follows.

Theorem 1. *Given a task vector* $\boldsymbol{w}' = \sum_{\boldsymbol{w}\in\mathcal{T}} \alpha_{\boldsymbol{w}}\boldsymbol{w}$ *for* $\alpha_{\boldsymbol{w}} \in \mathbb{R}, \forall \boldsymbol{w} \in \mathcal{T}$*, for all state-action pairs* $(s,a) \in \mathcal{S}\times\mathcal{A}$*, the action-value of* $\pi_{\boldsymbol{w}'}$*, which is an optimal policy for task* $\boldsymbol{w}'$*, on task* $\boldsymbol{w}'$ *is lower- and upper-bounded as* $L_{\boldsymbol{w}',\mathcal{T}}(s,a) \leq Q_{\boldsymbol{w}'}^{\pi_{\boldsymbol{w}'}}(s,a) \leq U_{\boldsymbol{w}',\mathcal{T},\boldsymbol{\alpha}}(s,a)$ *for*

$$L_{\boldsymbol{w}',\mathcal{T}}(s,a) := \max_{\boldsymbol{w}\in\mathcal{T}} \left[\tilde{Q}_{\boldsymbol{w}'}^{\pi_{\boldsymbol{w}}}(s,a) - \epsilon_{\boldsymbol{w}'}^{\pi_{\boldsymbol{w}}}(s,a)\right], \tag{4.11}$$

$$U_{\boldsymbol{w}',\mathcal{T},\boldsymbol{\alpha}}(s,a) := \sum_{\boldsymbol{w}\in\mathcal{T}} \max\left\{\alpha_{\boldsymbol{w}}\left(\tilde{Q}_{\boldsymbol{w}}^{\pi_{\boldsymbol{w}}}(s,a) + \epsilon_{\boldsymbol{w}}^{\pi_{\boldsymbol{w}}}(s,a)\right), \alpha_{\boldsymbol{w}} C_{\boldsymbol{w}}(s,a)\right\}, \tag{4.12}$$

for some $C_{\boldsymbol{w}}(s,a) \leq \min_\pi Q_{\boldsymbol{w}}^{\pi}(s,a)$ *such as* $C_{\boldsymbol{w}}(s,a) = \frac{1}{1-\gamma} r_{\boldsymbol{w}}^{min}$ *where* $r_{\boldsymbol{w}}^{min}$ *is the minimum reward on* $\boldsymbol{w}$ i.e., $r_{\boldsymbol{w}}^{min} = \min_{(s,a)\in\mathcal{S}\times\mathcal{A}} R_{\boldsymbol{w}}(s,a)$ *and* $\boldsymbol{\alpha} = \{\alpha_{\boldsymbol{w}}\}_{\boldsymbol{w}\in\mathcal{T}}$.

Proof. For the derivation of the lower bound $L_{\boldsymbol{w}',\mathcal{T}}(s,a)$, since $Q_{\boldsymbol{w}'}^{\pi_{\boldsymbol{w}'}}$ is the optimal action-value function for task $\boldsymbol{w}'$ and $Q_{\boldsymbol{w}'}^{\pi_{\boldsymbol{w}'}}(s,a) \geq Q_{\boldsymbol{w}'}^{\pi_{\boldsymbol{w}}}(s,a)$ for arbitrary task $\boldsymbol{w}$ and state-action pair (s,a),

$$Q_{\boldsymbol{w}'}^{\pi_{\boldsymbol{w}'}}(s,a) \;\geq\; \max_{\boldsymbol{w}\in\mathcal{T}} Q_{\boldsymbol{w}'}^{\pi_{\boldsymbol{w}}}(s,a) \;\geq\; \max_{\boldsymbol{w}\in\mathcal{T}} \left[\tilde{Q}_{\boldsymbol{w}'}^{\pi_{\boldsymbol{w}}}(s,a) - \epsilon_{\boldsymbol{w}'}^{\pi_{\boldsymbol{w}}}(s,a)\right]. \tag{4.13}$$

For the upper bound $U_{\boldsymbol{w}',\mathcal{T},\boldsymbol{\alpha}}(s,a)$, we use that $Q_{\boldsymbol{w}}^{\pi_{\boldsymbol{w}'}}(s,a) \leq Q_{\boldsymbol{w}}^{\pi_{\boldsymbol{w}}}(s,a)$ and $Q_{\boldsymbol{w}}^{\pi_{\boldsymbol{w}'}}(s,a) \geq \min_\pi Q_{\boldsymbol{w}}^{\pi}(s,a) \geq C_{\boldsymbol{w}}(s,a)$ for arbitrary task $\boldsymbol{w}$ and state-action

pair (s, a), which leads to

$$
\begin{aligned}
Q^{\pi_{\boldsymbol{w}'}}_{\boldsymbol{w}'}(s,a) &= \sum_{\boldsymbol{w}\in\mathcal{T}} \alpha_{\boldsymbol{w}} \left(Q^{\pi_{\boldsymbol{w}'}}_{\boldsymbol{w}}(s,a) - C_{\boldsymbol{w}}(s,a)\right) + \sum_{\boldsymbol{w}\in\mathcal{T}} \alpha_{\boldsymbol{w}} C_{\boldsymbol{w}}(s,a) \\
&\leq \sum_{\boldsymbol{w}\in\mathcal{T}} \max\left\{\alpha_{\boldsymbol{w}} \left(Q^{\pi_{\boldsymbol{w}'}}_{\boldsymbol{w}}(s,a) - C_{\boldsymbol{w}}(s,a)\right), 0\right\} + \sum_{\boldsymbol{w}\in\mathcal{T}} \alpha_{\boldsymbol{w}} C_{\boldsymbol{w}}(s,a) \\
&\leq \sum_{\boldsymbol{w}\in\mathcal{T}} \max\left\{\alpha_{\boldsymbol{w}} \left(Q^{\pi_{\boldsymbol{w}}}_{\boldsymbol{w}}(s,a) - C_{\boldsymbol{w}}(s,a)\right), 0\right\} + \sum_{\boldsymbol{w}\in\mathcal{T}} \alpha_{\boldsymbol{w}} C_{\boldsymbol{w}}(s,a) \\
&= \sum_{\boldsymbol{w}\in\mathcal{T}} \left\{\max\left\{\alpha_{\boldsymbol{w}} \left(Q^{\pi_{\boldsymbol{w}}}_{\boldsymbol{w}}(s,a) - C_{\boldsymbol{w}}(s,a)\right), 0\right\} + \alpha_{\boldsymbol{w}} C_{\boldsymbol{w}}(s,a)\right\} \\
&= \sum_{\boldsymbol{w}\in\mathcal{T}} \max\left\{\alpha_{\boldsymbol{w}} Q^{\pi_{\boldsymbol{w}}}_{\boldsymbol{w}}(s,a), \alpha_{\boldsymbol{w}} C_{\boldsymbol{w}}(s,a)\right\} \\
&\leq \sum_{\boldsymbol{w}\in\mathcal{T}} \max\left\{\alpha_{\boldsymbol{w}} \left(\tilde{Q}^{\pi_{\boldsymbol{w}}}_{\boldsymbol{w}}(s,a) + \epsilon^{\pi_{\boldsymbol{w}}}_{\boldsymbol{w}}(s,a)\right), \alpha_{\boldsymbol{w}} C_{\boldsymbol{w}}(s,a)\right\}. \qquad (4.14)
\end{aligned}
$$

□

In Equation (4.14), for each $\boldsymbol{w} \in \mathcal{T}$, the sign of $\alpha_{\boldsymbol{w}}$ determines which of the two terms in the max operator is used. If $\alpha_{\boldsymbol{w}} \geq 0$, the max operator selects the first term, whereas a negative $\alpha_{\boldsymbol{w}}$ lets the second term be used. Note that our Theorem 1 recovers Theorem 1 of [82] when $\boldsymbol{w}'$ is a conical combination of $\boldsymbol{w}$'s from $\mathcal{T}$ *i.e.*, $\alpha_{\boldsymbol{w}} \geq 0, \forall \boldsymbol{w} \in \mathcal{T}$.

Intuitively, this theorem states the condition that the optimal action-value for an arbitrary target task must satisfy, given the optimal successor features for the source tasks. The theorem is applicable to different problems wherever bounding of optimal values is useful. One example is policy cache construction, where the agent should decide whether to reuse existing policies in the cache set or learn a new one given each new task [82]. As will be shown in the next section, we employ the bounding as a constraint on the action-values for novel target tasks, for the guidance of transfer. In Sections 4.5.1–4.5.3, we empirically show that the application of our Theorem 1 can significantly improve the performance in the cases where target tasks are outside the conical hull of source tasks.

4.4.2 Constrained Training and Constrained GPI

As described in Section 4.3.2, the universal successor features approximators (USFAs) [21] improve the original successor features so that arbitrary policy vectors, including the ones for target tasks, can be used for GPI. However, the use of arbitrary policy vectors with USFAs solely relies on the generalization power of the approximators (*e.g.*, neural networks). Thus, the obtained successor features on novel tasks might contain high approximation errors, which could make the GPI policy perform poorly.

Our high-level idea to tackle the issue is to exploit the reward decomposition structure in Equation (4.1) even for obtaining SFs for new tasks, instead of solely relying on the approximators. We employ the lower and upper bounds on optimal values from Theorem 1 to enforce the bounds on the approximate successor features. As a result, the approximation errors can be reduced by restricting the estimated optimal values to be inside those bounds around the optimal values, which can prevent the use of erroneous values during the transfer to unseen tasks.

For now, we will first introduce how to train the successor features approximators to output the successor features that satisfy the bounds on novel tasks. Then, we will point out that an analogous effect can be accomplished by modifying only the inference algorithm, and propose *constrained GPI* as a simple yet effective *test-time* approach to improving zero-shot transfer to novel tasks.

Constrained training of SF approximators. In the original training of USFAs, the approximators are learned with a set of source tasks $\mathcal{T}$ in Equation (4.7). We propose to guide the training by employing Theorem 1; we impose constraints for the approximators using the lower and upper bounds on the op-

timal values for arbitrary linear combinations of source tasks. Specifically, for the training of USFAs, we use Equation (4.7) but with the following constraints:

$$L_{\boldsymbol{w}',\mathcal{T}}(s,a) \leq \tilde{\psi}(s,a,\boldsymbol{w}')^\top \boldsymbol{w}' \leq U_{\boldsymbol{w}',\mathcal{T},\xi(\boldsymbol{w}',\mathcal{T},s,a)}(s,a) \quad \text{for} \quad \boldsymbol{w}' \in \mathcal{W}, \tag{4.15}$$

where (s,a) is the same sample as the main objective of Equation (4.7). $\mathcal{W}$ is a set of task vectors for the constraints, which can be independent of the source task set $\mathcal{T}$, and $\xi(\cdot)$ determines the coefficients $\boldsymbol{\alpha}$ given a target task $\boldsymbol{w}'$ and $\mathcal{T}$. We will explain later how to determine $\xi(\cdot)$. $\mathcal{W}$ can be any subset of the linear span of source task vectors, but practically, we randomly sample a number of vectors from the span at each update. Since the targets of the constraints are not fixed with respect to both $\boldsymbol{w}'$ and (s,a) throughout the training, we use penalty terms (or soft constraints) that linearly penalize the constraint violations as

$$\frac{1}{|\mathcal{W}|} \sum_{\boldsymbol{w}' \in \mathcal{W}} \Big(\Big\{ L_{\boldsymbol{w}',\mathcal{T}}(s,a) - \tilde{Q}^{\pi_{\boldsymbol{w}'}}_{\boldsymbol{w}'}(s,a) \Big\}_+ \\ + \Big\{ \tilde{Q}^{\pi_{\boldsymbol{w}'}}_{\boldsymbol{w}'}(s,a) - U_{\boldsymbol{w}',\mathcal{T},\xi(\boldsymbol{w}',\mathcal{T},s,a)}(s,a) \Big\}_+ \Big),$$

where $\{x\}_+$ denotes $\max\{x, 0\}$.

The constrained training suggested above can make the approximators comply with the bounds for any tasks without requiring any additional interactions with the environment. However, it has some downsides. First, since it is a new training procedure, existing pre-trained models cannot be used. It requires some additional computational cost compared to the naive training of successor features approximators. Second, the enforcement of the constraints for training can introduce additional hyperparameters (*e.g.*, the weight coefficient for the penalty terms). Thus, suboptimal hyperparameters may introduce either instability in the training or a decrease in the performance.

Test-time constrained GPI. Our idea starts with the observation that in the constrained training, the learned successor features from source tasks are considered the "trustworthy" features for the constraints, because the USFAs are trained on the source tasks. Besides, only the source successor features are used for computing the constraints for all the other tasks. It implies that the learning of the source successor features better not be affected by other criteria, and more accurate source successor features would produce better constraints for other tasks with smaller errors.

Based on the implication, we propose *constrained GPI*, which can not only overcome the limitation of USFAs as done by the aforementioned constrained training but also have two additional practical merits: (i) it is computationally simpler, and (ii) it is a test-time approach with no training. Simply put, we propose replacing the usual GPI policy with the constrained GPI policy as

$$\pi_{\mathrm{CGPI}}(s) \in \operatorname*{argmax}_{a} \max_{\boldsymbol{z} \in \mathcal{C}} \Big[\min \Big\{ \max \big\{ \tilde{Q}^{\pi_{\boldsymbol{z}}}_{\boldsymbol{w}'}(s,a), L_{\boldsymbol{w}',\mathcal{T}}(s,a) \big\}, U_{\boldsymbol{w}',\mathcal{T},\xi(\boldsymbol{w}',\mathcal{T},s,a)}(s,a) \Big\} \Big], \tag{4.16}$$

where the target task $\boldsymbol{w}'$ is expressible as a linear combination of the source tasks and $\xi(\cdot)$ again outputs $\boldsymbol{\alpha}$ given $\boldsymbol{w}'$ and $\mathcal{T}$ as in Equation (4.15). $\mathcal{C}$ is a set of policies that we can freely choose when applying the constrained GPI.

The constrained GPI policy selects the actions that maximize the maximum action-values as the original GPI policy does but also caps the values with the lower and upper bound constraints derived from the source successor features. The upper bound constraint fixes the overestimation of values computed with approximate successor features for either the target task $\boldsymbol{w}'$ or any other tasks used for constrained GPI. The lower bound constraint ensures that action-values on the target task for the greedy action selection are at least as close to the

optimal target action-values as the lower bounds.

The approximation error terms in the lower and upper bounds *i.e.*, $\epsilon^{\pi_{\boldsymbol{w}}}_{\boldsymbol{w}'}(s,a)$ and $\epsilon^{\pi_{\boldsymbol{w}}}_{\boldsymbol{w}}(s,a)$ in Theorem 1 could be ignored in practice, as long as the approximation errors of the source successor features are sufficiently small. Also, we can obtain the tightest upper bound by defining $\xi(\cdot)$ as

$$\begin{aligned} \xi(\boldsymbol{w}', \mathcal{T}, s, a) &\coloneqq \operatorname*{argmin}_{\{\alpha_{\boldsymbol{w}}\}_{\boldsymbol{w}\in\mathcal{T}}} U_{\boldsymbol{w}',\mathcal{T},\{\alpha_{\boldsymbol{w}}\}_{\boldsymbol{w}\in\mathcal{T}}}(s,a) \\ \text{subject to} \quad \boldsymbol{w}' &= \sum_{\boldsymbol{w}\in\mathcal{T}} \alpha_{\boldsymbol{w}} \boldsymbol{w}. \end{aligned} \tag{4.17}$$

The objective $U_{\boldsymbol{w}',\mathcal{T},\{\alpha_{\boldsymbol{w}}\}_{\boldsymbol{w}\in\mathcal{T}}}(s,a)$ is the sum of the piecewise linear functions. Thus, Equation (4.17) can be solved with linear programming.

We observe that using the lower bound constraint with $L_{\boldsymbol{w}',\mathcal{T}}(s,a)$ is equivalent to including the successor features for source tasks in the input to the constrained GPI; *i.e.*, $\mathcal{T} \subseteq \mathcal{C}$. Also, since $L_{\boldsymbol{w}',\mathcal{T}}(s,a) \leq U_{\boldsymbol{w}',\mathcal{T},\xi(\boldsymbol{w}',\mathcal{T},s,a)}(s,a)$, there would be no difference between GPI and constrained GPI when $\mathcal{C} = \mathcal{T}$. Thus, in our experiments, we mainly use $\mathcal{C} = \{\boldsymbol{w}'\}$, which is equivalent to using $\mathcal{C} = \mathcal{T} \cup \{\boldsymbol{w}'\}$.

4.5 Experiments

4.5.1 Scavenger Experiments

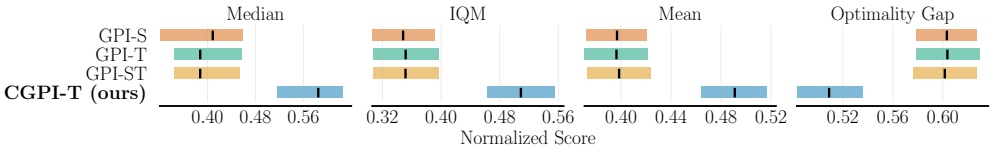
We start our experiments in the Scavenger environment [16, 17], which can assess our approach with minimal influence from external causes. In Scavenger, the agent is positioned at one of the cells in a $G \times G$ grid, and the goal is to maximize the return by collecting objects. Both the agent and objects are spawned at random locations, and there are d classes of objects where the class determines the value of the reward. The state space is $\mathcal{S} = \{0,1\}^{G\times G\times(d+1)}$, where

the first d channels describe the current locations of the objects on the map and the last channel specifies the walls where the agent cannot go and objects do not appear. There are four actions available: $\mathcal{A} = \{\texttt{UP}, \texttt{DOWN}, \texttt{LEFT}, \texttt{RIGHT}\}$, and the agent picks up an object by visiting the cell of the object, which spawns a new object of a random class at a random location. The feature $\phi(s, a, s') \in \{0, 1\}^d$ is a one-hot vector whose element represents whether the agent picks up an object of that type or not within the transition. The task vector $\boldsymbol{w} \in \mathbb{R}^d$ determines the reward values for the d different classes of objects. We use a maximum episode horizon of 50 and a discount factor of 0.9. Please see Barreto et al. [17] for the full details.

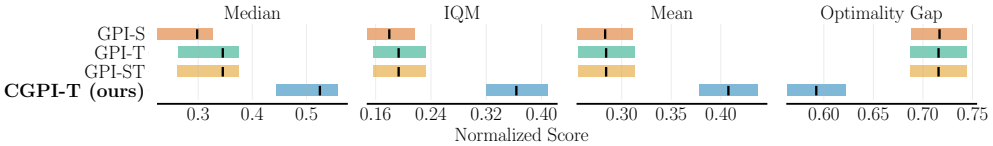
We evaluate the zero-shot transfer performance of different approaches *i.e.*, we first train USFAs as proposed in [21], and measure the performance of GPI and constrained GPI policies that use the same set of USFAs on target tasks with no further policy updates. We set $G = 11$ and use 20 objects in total with the different numbers of classes; $d = 2$ and $d = 4$. With $d = 4$, we also test the USFAs that are learned with the constrained training for a comparison. We use the standard basis vectors of $\mathbb{R}^d$ as the set of source tasks as done in [21], and evaluate agents on the set of target tasks defined by $\{-1, 1\}^d$. Therefore, all the target tasks except for the all-ones vector $\mathbf{1}$ are not covered by the conical hull of source tasks, which requires Theorem 1 for bounding of values.

We train eight USFAs agents for 1M steps, and evaluate them on each target vector 10 times with a fixed set of 10 random seeds. To be invariant to the reward scale differences between different tasks, we normalize the scores (or returns) from the environment by the minimum and maximum scores with respect to all the agents' evaluation episodes on each task.

Figures 4.2 and 4.3 compare the performance of the USFAs agents with GPI



(a) All

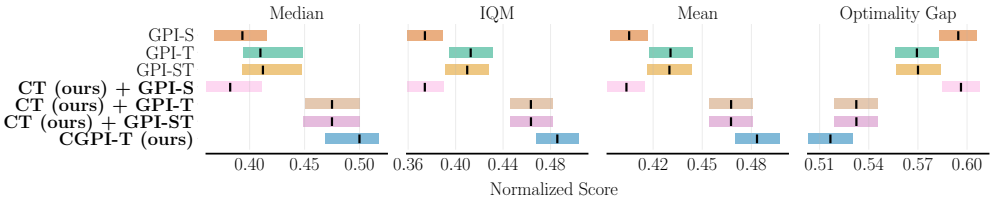


(b) Harsh

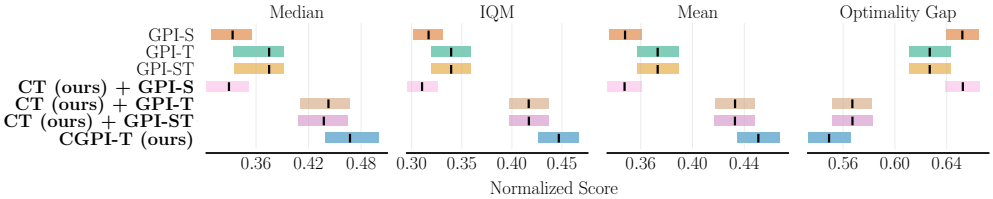
Figure 4.2: The aggregated performance metrics with 95% bootstrap confidence intervals [6, 35] of different test-time approaches on Scavenger with $d = 2$. `CGPI` represents our *constrained GPI*, whereas `GPI` is the original GPI. The suffixes `-S`, `-T` and `-ST` denote using the set of source task vectors, the target task vector and both as $\mathcal{C}$, respectively. (a) All is the evaluation on the entire set of target task vectors from $\{-1, 1\}^d$, whereas (b) Harsh denotes the evaluation on a subset consisting of 'harsh' tasks, whose number of -1's is no less than that of 1's (*e.g.*, $\boldsymbol{w} = (-1, 1)$).

and constrained GPI for exploitation, following the evaluation scheme suggested by [6]. Although they use the same set of trained USFAs, the constrained GPI brings a notable performance improvement in comparison with the original GPI. Also, Figure 4.3 suggests that the constrained GPI, the test-time method, can match or even outperform the agents learned with the constrained training. One possible explanation is that the constrained training might experience some instability in learning depending on the choice of the hyperparameters, as described in Section 4.4.2.

In the first and the second columns of Table 4.1, we present the proportions of the action-values that are changed by the lower and upper bounds of the constrained GPI, measured for the evaluation on Scavenger. The third column shows the proportions of resulting greedy actions changed by them. It implies



(a) All



(b) Harsh

Figure 4.3: The aggregated performance metrics with 95% bootstrap confidence intervals [6, 35] of different test-time approaches on Scavenger with $d = 4$. CT denotes the constrained training. We refer the reader to Figure 4.2 for the full description.

Table 4.1: The first and second columns show the proportions of the maximum action-values changed by constrained GPI's lower and upper bounds, during the evaluation on Scavenger. The final column is the proportions of resulting actions changed by them.

Setting	Lower-bounding	Upper-bounding	Action change
$d = 2$	23.94%	43.01%	22.11%
$d = 4$	5.60%	46.86%	20.78%

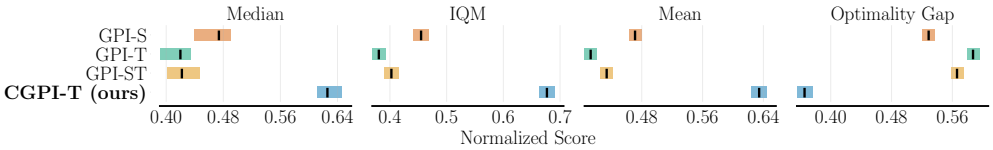
that USFAs *i.e.*, the function approximators of successor features, may not satisfy the optimal value bounds presented in Theorem 1, and applying the bounds could change a fair proportion of greedy actions to improve the performance.

4.5.2 Robotic Locomotion Experiments

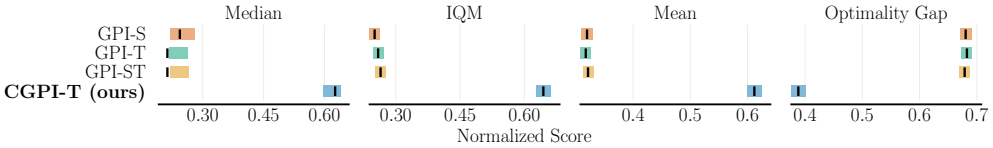
To evaluate our approach in a physical domain, following [14, 82], we employ Reacher, a MuJoCo's robotic locomotion environment [107] from OpenAI Gym [22]. Reacher simulates a robotic arm with two joints, and its state space $\mathcal{S}$ is 11-dimensional. Its original action space is continuous and two-dimensional, which represents torques at the two hinge joints, and we discretize each action dimension into three values resulting in nine discrete actions in total, as in [14]. We set the maximum episode horizon as 500 and the discount factor as 0.9.

For its use in the zero-shot transfer problem, we first set four fixed goal locations at $(0.1, 0.0), (0.0, 0.1), (-0.1, 0.0), (0.0, -0.1)$, and define $\phi(s, a, s') \in \mathbb{R}^4$ to be a vector whose elements are the negative distances between the agent's fingertip and the four goals. USFAs agents are trained with the four standard basis vectors as source tasks, learning to reach or get close to one of the four goals on each of the four source tasks. However, for the evaluation, we define $\{-1, 1\}^d$ to be the set of target tasks. A negative value at each dimension implies not only that the target vector is outside the conical hull of the source tasks but also that the agent would obtain higher rewards with respect to that dimension by getting away from the corresponding goal, instead of reaching the goal. This can make this zero-shot transfer problem challenging, as it requires the agent to do very different behaviors suddenly at test time. For better understanding, Figure 4.5 shows a rendered scene of Reacher, where the four red dots represent the goal locations. Since the original states defined for Reacher contain information about target coordinates, we set those coordinates to zeros for our experiments with different tasks.

In our experiments, we train 16 USFAs agents for 1M environment steps to obtain statistically more meaningful results. We evaluate each of the trained



(a) All



(b) Harsh

Figure 4.4: The aggregated performance metrics with 95% bootstrap confidence intervals [6, 35] of different test-time approaches on Reacher with $d = 4$. `CGPI` represents *constrained GPI*. The suffixes `-S`, `-T` and `-ST` denote using the set of source task vectors, the target task vector and both as $\mathcal{C}$, respectively. (a) All is the evaluation on the entire set of target task vectors from $\{-1, 1\}^d$, whereas (b) Harsh denotes the evaluation on a subset consisting of 'harsh' tasks, whose number of -1's is no less than that of 1's.

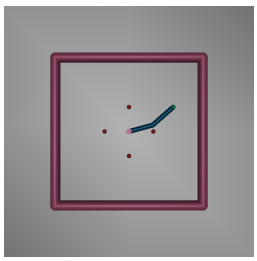
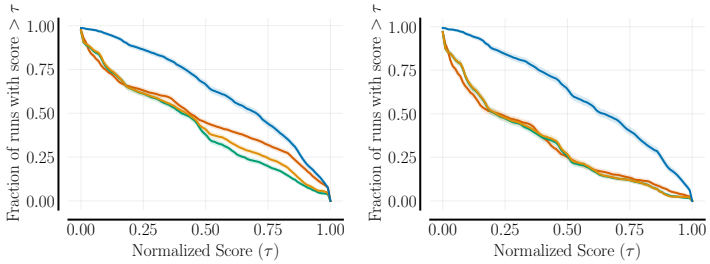


Figure 4.5: An example scene of Reacher. The four red dots indicate the four goal locations.



(a) All (b) Harsh

Figure 4.7: The performance profiles [6, 33] of the inference with GPI and constrained GPI on Reacher. The categories *i.e.*, (a) All and (b) Harsh, and the colors match the ones in Figure 4.4, and thus the blue lines represent constrained GPI.

agents with 10 episodes for each target vector, again with a fixed set of 10 environment random seeds. Similarly to the Scavenger experiments in the previous section, to take into account the difference in reward scales between different

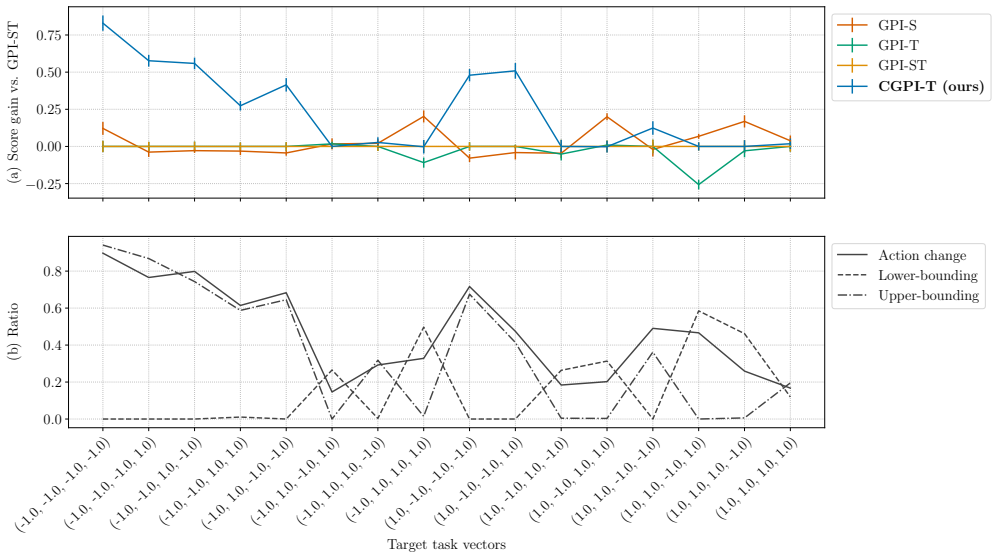


Figure 4.8: The performance gains of different test-time approaches and the ratios of action and action-value changes due to constrained GPI, with respect to the target task vectors on Reacher. `CGPI` represents *constrained GPI*. The suffixes `-S`, `-T` and `-ST` denote using the set of source task vectors, the target task vector and both as $\mathcal{C}$, respectively. (a) presents the interquartile means (IQMs) of their normalized scores, visualized as performance gains (or differences) compared to `GPI-ST`, where the error bars are the 95% confidence intervals. `Action change` in (b) shows the average portion of the final actions changed by constrained GPI for each target task, during the evaluation. `Lower-bounding` and `Upper-bounding` in (b) denote the average ratios of the action-values after taking the maximums changed by constrained GPI's lower and upper bounds, respectively.

target tasks, for each of the target tasks, we normalize the returns *i.e.*, scores using the minimum and maximum scores that any of the agents achieve during the evaluation.

Figure 4.4 provides the comparison of GPI and constrained GPI in terms of the normalized scores. As it shows, the inference with constrained GPI significantly outperforms the ones with GPI on the zero-shot transfer problem. Especially, the gap becomes even larger when they are evaluated on the "Harsh" set

of target task vectors, which implies that constrained GPI can be helpful for transferring to different types of target tasks outside the conical hull of source tasks. On the other hand, Figure 4.7 presents the performance profiles [6, 33] of the approaches with the matching evaluation categories and color mapping. It qualitatively suggests that constrained GPI is more likely to achieve higher scores than GPI.

We provide further analyses of the experimental results for a better understanding of our approach. Figure 4.8 visualizes four quantities with respect to the target task vectors, in the Reacher environment. Figure 4.8a reports the performance gains of the approaches compared to `GPI-ST` in terms of the interquartile means (IQMs) of the normalized scores on each target task. In Figure 4.8b, `Action change` shows the average ratio of the action changes due to constrained GPI, for each of the target tasks. `Lower-bounding` and `Upper-bounding` are the average portions of the maximum action-values for the inference that are bounded and changed by constrained GPI's lower and upper bounds, respectively. We restate that the USFAs are trained with the four standard basis task vectors *i.e.*, a (positive) one-hot vector for each of the four dimensions of the task vector space, $\mathbb{R}^4$.

Our first observation is that while the transferred agents perform comparably on some tasks, constrained GPI makes significant differences on the others, especially more on the "Harsh" target tasks with many -1's as elements in their task vectors. It implies that the USFAs, the function approximators, might work reasonably on tasks to which the approximations could be extrapolated similarly as in the source tasks, but they could perform poorly when there is little or no knowledge easily transferable from the source tasks only with the function approximators.

It also coincides with our second observation: examining the two plots in Figure 4.8 together, there is a tendency that more action changes by constrained GPI usually result in greater performance gains. This suggests that bounding the action-values at test time with constrained GPI for reducing the value approximation errors often has a meaningful effect on the resulting performance, and the highest increases in both the performance gains and the action change ratios are observed on the "Harsh" target tasks. Thus, we infer that the bounds from Theorem 1 could be effective for constraining the approximation errors of USFAs, especially including target tasks outside the conical combinations of source tasks.

Additionally, `Lower-bounding` and `Upper-bounding` in Figure 4.8b indicate that a large portion of the action changes and thus the performance gains are related to the upper-bounding in the Reacher environment, and the reduction of the USFAs' overestimation with constrained GPI is an important factor of the performance gains, in this case. Depending on the underlying environment and tasks, there can be target tasks where USFAs underestimate the corresponding action-values and the lower-bounding, which is equivalent to performing GPI with the source task set $\mathcal{T}$ included in the input policy set $\mathcal{C}$, could help improve the resulting performance, as well.

4.5.3 DeepMind Lab Experiments with Learned ϕ

For evaluation of our approach in a more complex and realistic setting, we employ DeepMind Lab [15, 18, 21] and conduct experiments in a first-person view 3D environment. In a single room, a goal object is placed arbitrarily, and the objective is to reach the goal before the episode ends where its location changes between tasks. Figure 4.9 shows an example scene that the agent sees

with the goal object in red.

At every time step, the agent observes an $84 \times 84 \times 3$ image from the environment and outputs one of 45 possible actions, which include 5, 3 and 3 choices for `LOOK_LEFT_RIGHT_PIXELS_PER_FRAME`, `STRAFE_LEFT_RIGHT` and `MOVE_BACK_FORWARD` controls, respectively. Since observations are in the first-person view, the goal object may not be seen by the agent, which makes transfer given the task information $\boldsymbol{g}$ critical to the success of the tasks. In each task, we divide the room into an 11×11 grid and place the goal object in one of the cells. The task information $\boldsymbol{g}$ is a two-dimensional vector that contains the coordinate of the goal in the grid. Starting at the center of the room, the agent receives a reward of one if it reaches the goal within the episode horizon or no rewards otherwise. Therefore, the reward functions are sparse. We render 10 frames per second and use an episode horizon of 50 and a discount factor of 0.99. Also, as the tasks we use are sparse-reward tasks, we sample each episode with one source task vector during the training.

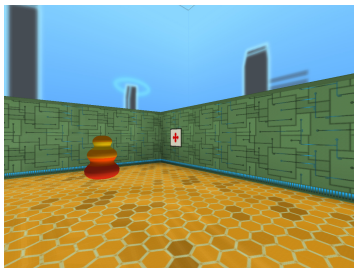
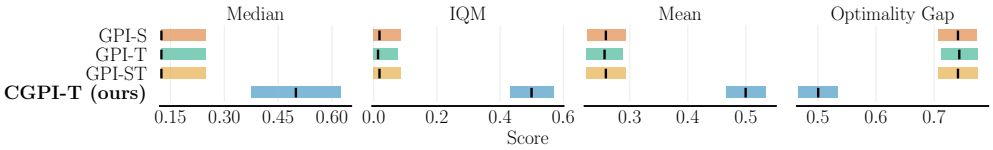


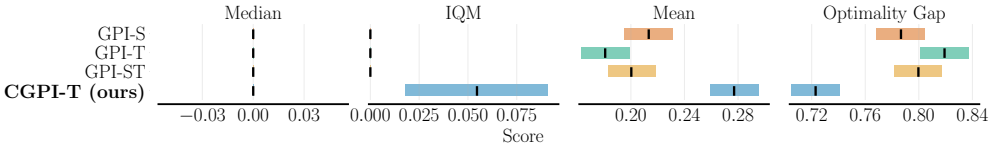
Figure 4.9: An example scene that the agent sees in the DeepMind Lab tasks. The object is the goal.

For these experiments where the agent observes rendered images rather than the underlying states, it may not be viable to define features ϕ's and task vectors $\boldsymbol{w}$'s that linearly decompose reward functions. Therefore, we train agents with the learning of $\tilde{\phi}$ and $\tilde{\boldsymbol{w}}$ from samples from the source tasks with $d = 2$ as described in Section 4.3.3.

Inspired by Hong et al. [51], we examine zero-shot transfer with the GPI and constrained GPI using two transfer settings: "left-to-right" and "near-to-far". In the "left-to-right" setting, the agent is trained on the source tasks whose



(a) Left-to-right



(b) Near-to-far

Figure 4.10: The aggregated performance metrics with 95% bootstrap confidence intervals [6, 35] of different test-time approaches with the two scenarios in the DeepMind Lab environment. `CGPI` represents *constrained GPI*. The suffixes `-S`, `-T` and `-ST` denote using the source task information set, the target task information and both for $\mathcal{C}$, respectively. (a) In the left-to-right setting, we train the agent for the goals from the left half and test for the right half. (b) In the near-to-far setting, we train the agent for nearby goals and test for farther goals.

goals are sampled from the left half of the room and is tested on the target tasks with goals from the right half. In the "near-to-far" setting, the source tasks have the goals within an $L^\infty = 2$ distance from the center of the room, and target tasks set the goals farther than $L^\infty = 2$.

For each setting, we train eight USFAs agents with different seeds for 3M environment steps on the source tasks and test them on the target tasks. Figure 4.10 presents the comparison of the GPI with different $\mathcal{C}$'s and the constrained GPI. Leveraging the same set of trained USFAs with learned $\tilde{\phi}$ and $\tilde{\boldsymbol{w}}$, the constrained GPI outperforms the GPI with the three $\mathcal{C}$'s in both settings by a notable margin. Another observation is that the trained USFAs agents seem to overfit more to the source tasks in the "near-to-far" setting compared to the "left-to-right" setting. It makes the performance on the target tasks much worse. Nonetheless, the constrained GPI is still helpful in such overfitting

situations.

4.5.4 Implementation Details

We employ the autonomous learning library [83] and PyTorch [86] for the implementation of USFAs. The following selection of hyperparameters was done based on the performance of the USFAs on source tasks, because we have no access to target tasks during the training in the transfer problem and the learning on the source tasks is important for the transfer with both GPI and constrained GPI.

For both of the Scavenger and Reacher environments, we use an MLP with the ReLU nonlinearity and two hidden layers whose sizes are $(128, 256)$, which is chosen over $(64, 128)$ and $(256, 256)$, as USFAs. We set the number of output heads of the networks to the respective number of discrete actions; four for Scavenger and nine for Reacher. For the training of the USFAs, we make use of the Adam optimizer [65] with a learning rate of $1e-4$, which is selected out of $\{3e-4, 1e-4, 3e-5\}$. The target update frequency is set to 100 for Scavenger and 500 for Reacher, where we consider $\{100\}$ for Scavenger and $\{100, 500\}$ for Reacher, and the update frequency is configured as 1 for the two environments. On Scavenger, we use 256-sized replay buffers (chosen out of $\{1, 128, 256\}$) and mini-batches with a size of 1 after testing $\{1, 4, 8, 16\}$. For Reacher, we make replay buffers and mini-batches 2048-sized and 32-sized, which are chosen from $\{2048\}$ and $\{1, 8, 32\}$. For the exploration, we use the ϵ-greedy with its value of 0.1.

For the DeepMind Lab environment, we employ the frame stacking strategy and the network architecture for our shared feature extractor for $\tilde{\psi}$ and $\tilde{\phi}$ from Mnih et al. [77], where the latent feature dimensionality is 64. $\tilde{\boldsymbol{w}}$ has one hidden

layer with 16 units. A learning rate of $1e-5$ for the Adam optimizer is chosen out of $\{1e-4, 1e-5\}$. The target update frequency and the update frequency are 500 and 1, respectively. We use a replay buffer whose size is 65536 and sample 16-sized mini-batches, which is chosen from $\{16, 64\}$.

Specifically for USFAs, for the Scavenger and Reacher environments, we sample five policies from $\mathcal{D}_{\boldsymbol{z}}(\cdot|\boldsymbol{w})$ at each step, where each dimension of a policy vector is sampled from a Gaussian distribution with a standard deviation of 0.1 and a mean of the corresponding element of $\boldsymbol{w}$. We use the same number of policy vectors and policy vector distribution but conditioned on the task information instead of task vector, for the DeepMind Lab environment.

For the constrained training of USFAs, we use the same set of hyperparameters, as well as a weight coefficient for the constraints of 0.1 (chosen from $\{0.1, 1.0, 5.0\}$), where one task is uniformly randomly sampled from the bounded linear span of source tasks at each step. Also, for a more stable learning, we load the model checkpoints of USFAs at the 0.5M-th step and train for the remaining 0.5M steps. We universal employ cvxpylayers [7] as a general LP solver for the upper bound with the default solver configurations, for both constrained GPI and the constrained training.

For the Scavenger and Reacher experiments, we conduct the experiments with our CPU machines, where majority of the CPUs are Intel Xeon Gold 6130 or Intel Xeon E5-2695. For each training run, we use two CPU cores without any GPUs for about 6-12 hours. For the DeepMind Lab experiments, we employ our GPU machines and run experiments for about 48-72 hours.

4.6 Summary

We presented constrained GPI, a simple yet effective test-time approach for transfer with approximate successor features. We first focused on the issue that although universal successor features approximators (USFAs) exploit the smoothness of optimal values across different tasks, their approximation errors on novel target tasks could be large especially when those tasks are quite distant from source tasks. Thus, we introduced a theorem about lower and upper bounds on the optimal values for novel task vectors that belong to the task vector space linearly spanned by the set of source task vectors, relaxing the conical combination condition used for the theorem by Nemecek and Parr [82]. We proposed a constrained training scheme making use of those bounds for reducing the action-value errors of the learned approximators on novel tasks. We then suggested *constrained GPI* that uses the bounds at test time to achieve an analogous effect, allowing the use of previously trained models. We empirically showed that this test-time approach can improve the zero-shot transfer performance by a large margin in multiple environments.

Limitations and possible improvements. There may be some cases where the minimum rewards for source tasks *i.e.*, $r_{\boldsymbol{w}}^{\min}$'s are overly small, which could lead to less changes of both action-values and behaviors induced by the upper-bounding in Theorem 1 with $C_{\boldsymbol{w}}(s,a) = \frac{1}{1-\gamma} r_{\boldsymbol{w}}^{\min}$. An interesting direction to tackle the issue is to learn the minimum action-value function during the training and to use the approximate minimum value at each state-action pair as $C_{\boldsymbol{w}}(s,a)$ for deriving the upper bound in Theorem 1. It may allow computing upper bounds more tightly and adaptively for different state-action pairs. Also, if the learned successor features approximators have large errors even on source tasks, not only GPI but also constrained GPI's bounding may not be meaning-

fully helpful. One idea to mitigate the issue is to take the uncertainty in the approximators and the approximation error term that appears in Theorem 1 into account, *e.g.*, by using ensemble models.

Chapter 5

Conclusion

5.1 Summary

In this thesis, we discussed approaches to building generalizable agents in the two critical aspects: abstraction and transfer. We covered the specific topics on feature abstraction for robustness and efficiency, unsupervised temporal abstraction of behaviors for reuse and transfer trained agents between tasks with distinct reward functions.

In Chapter 2, we introduced Drop-Bottleneck (DB), a new information bottleneck method that discretely drops features. It compresses the input information with learned feature-wise drop probabilities, which can be jointly optimized with the feature extractor. As a result of the training, high drop probabilities can be assigned to features that are less relevant to the task. Moreover, we defined the deterministic version of the compressed representations so that they can be used for inference tasks that require consistent and stable results or

efficient and optimized computation, with the majority of the compression and robustness in force. The experimental results showed that DB can be adopted for improved robustness to adversarial samples and practical computational efficiency for inference.

In Chapter 3, we proposed Information Bottleneck Option Learning (IBOL), an unsupervised skill discovery method that learns to abstract behaviors with disentanglement for better reusability on top of linearized environment dynamics. Its lowest-level component, the linearizer, is a policy that is trained to perform raw actions to make a state transition in the specified direction. Combined with the linearizer, IBOL discovers extensive skills and learns the disentangled abstraction of them for a simpler and more interpretable mapping. Our empirical results suggested that its abstractions are not only more disentangled and simpler but also performant when combined with meta-controllers for solving downstream tasks than those of the baseline methods,

In Chapter 4, we presented a theorem that expresses the lower and upper bounds for the optimal values for new tasks with learned successor features for source tasks. Based on the theorem, we proposed a method that bounds optimal value approximations for target tasks during the inference, named constrained GPI. Without requiring modifications to the training process or additional training, constrained GPI empirically outperformed the baselines on target tasks with novel reward functions.

5.2 Future Work

Not only the high-level goal of building agents that generalize but also the topics discussed in this thesis are still open for diverse further research. We discuss a few potential directions for future work in this section.

More dynamics-aware behavior and skill abstractions. The goal of abstracting behaviors and skills is to leverage them for different tasks, and it would be beneficial for the abstractions to have better controllability in the environment and its state space. Currently, there are relevant attempts to improve their alignment with the resulting transitions [84, 85]. For future research, we hope to observe further exploration in the direction, possibly taking the environment dynamics into account in the abstractions. The future development might gain from existing studies such as successor representations [31], successor features [14] and γ-models [56].

Skill discovery with minimal guidance. While unsupervised skill discovery can be useful for discovering reusable skills without the cost of integrating supervisory signals, challenging real-world problems might involve exceedingly complex environments with huge behavior spaces. In such environments, due to the enormity of the behavior spaces, it may not be trivial to establish a set of skills that could benefit the downstream tasks of interest in a completely unsupervised manner. Therefore, it would be interesting to investigate how to aid the discovery of skills with some guidance on which behaviors are considered common and useful, while still minimizing the introduced cost of such guidance. Providing human preferences for the learning of skills [109] could be a good direction to achieve the goal, and we look forward to further research on this topic.

Refinable sets of skill abstractions. While the main goal of skill discovery is to identify and learn reusable behaviors, depending on the tasks for which the learned behaviors are used, their abstractions might not be ideally suited for those new tasks. In such situations, having the ability to adjust the set of learned skill abstractions for new tasks would be a helpful and important

asset. While fine-tuning trained skill abstractions such as skill policies has been an existing option, it would be tempting to investigate how to better adjust them for downstream tasks, possibly with further contextual information.

Bibliography

[1] Majid Abdolshah, Hung Le, Thommen George Karimpanal, Sunil Gupta, Santu Rana, and Svetha Venkatesh. A new representation of successor features for transfer across dissimilar environments. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, pages 1–9. PMLR, 2021.

[2] Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299*, 2018.

[3] Alessandro Achille and Stefano Soatto. Emergence of invariance and disentanglement in deep representations. *The Journal of Machine Learning Research*, 19(1):1947–1980, 2018.

[4] Alessandro Achille and Stefano Soatto. Information dropout: Learning optimal representations through noisy computation. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2897–2905, 2018.

[5] Tameem Adel, Zoubin Ghahramani, and Adrian Weller. Discovering interpretable representations for both deep generative and discriminative

models. In *International Conference on Machine Learning*, pages 50–59. PMLR, 2018.

[6] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 34, 2021.

[7] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, 2019.

[8] Michael Ahn, Henry Zhu, Kristian Hartikainen, Hugo Ponte, Abhishek Gupta, Sergey Levine, and Vikash Kumar. Robel: Robotics benchmarks for learning with low-cost robots. In *Conference on Robot Learning*, pages 1300–1313. PMLR, 2020.

[9] Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. Opal: Offline primitive discovery for accelerating offline reinforcement learning. *ArXiv*, abs/2010.13611, 2020.

[10] Lucas Nunes Alegre, Ana Bazzan, and Bruno C Da Silva. Optimistic linear support and successor features as a basis for optimal policy transfer. In *International Conference on Machine Learning*, pages 394–413. PMLR, 2022.

[11] Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. Deep variational information bottleneck. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

[12] Safa Alver and Doina Precup. Constructing a good behavior basis for

transfer using generalized policy updates. In *10th International Conference on Learning Representations, ICLR 2022*, 2022.

[13] Rana Ali Amjad and Bernhard Claus Geiger. Learning representations for neural network-based classification using the information bottleneck principle. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[14] André Barreto, Will Dabney, Rémi Munos, Jonathan J. Hunt, Tom Schaul, David Silver, and Hado van Hasselt. Successor features for transfer in reinforcement learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4055–4065, 2017.

[15] André Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel J. Mankowitz, Augustin Zídek, and Rémi Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 510–519. PMLR, 2018.

[16] André Barreto, Diana Borsa, Shaobo Hou, Gheorghe Comanici, Eser Aygün, Philippe Hamel, Daniel Toyama, Jonathan J. Hunt, Shibl Mourad, David Silver, and Doina Precup. The option keyboard: Combining skills in reinforcement learning. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 13031–13041, 2019.

[17] André Barreto, Shaobo Hou, Diana Borsa, David Silver, and Doina Precup. Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, (48):30079–30087, 2020.

[18] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

[19] Richard E. Bellman. Dynamic programming. *Princeton University Press*, 1957.

[20] Christopher Berner, G. Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, D. Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *ArXiv*, abs/1912.06680, 2019.

[21] Diana Borsa, André Barreto, John Quan, Daniel J. Mankowitz, Hado van Hasselt, Rémi Munos, David Silver, and Tom Schaul. Universal successor features approximators. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[22] G. Brockman, Vicki Cheung, Ludwig Pettersson, J. Schneider, John Schulman, Jie Tang, and W. Zaremba. Openai gym. *ArXiv*, abs/1606.01540, 2016.

[23] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

[24] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

[25] Víctor Campos Camúñez, Alex Trott, Caiming Xiong, Richard Socher, Xavier Giró Nieto, and Jordi Torres Viñals. Explore, discover and learn: unsupervised discovery of state-covering skills. In *ICML 2020, Thirty-seventh International Conference on Machine Learning:[posters]*, pages 1–17, 2020.

[26] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE, 2017.

[27] Matthew Chalk, Olivier Marre, and Gasper Tkacik. Relevant sparse codes with variational information bottleneck. In *Advances in Neural Information Processing Systems*, pages 1957–1965, 2016.

[28] Ricky TQ Chen, Xuechen Li, Roger B Grosse, and David K Duvenaud. Isolating sources of disentanglement in variational autoencoders. In *Advances in neural information processing systems*, pages 2610–2620, 2018.

[29] John D. Co-Reyes, Yuxuan Liu, A. Gupta, Benjamin Eysenbach, P. Abbeel, and Sergey Levine. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In *ICML*, 2018.

[30] Bin Dai, Chen Zhu, and David Wipf. Compressing neural networks using the variational information bottleneck. *arXiv preprint arXiv:1802.10399*, 2018.

[31] Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, (4):613–624, 1993.

[32] Kien Do and Truyen Tran. Theory and evaluation metrics for learning disentangled representations. In *International Conference on Learning Representations*, 2020.

[33] Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

[34] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*, 2019.

[35] Bradley Efron. Bootstrap methods: another look at the jackknife. In *Breakthroughs in statistics*, pages 569–593. Springer, 1992.

[36] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. 2019.

[37] Fernando Fernández and Manuela Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 720–727, 2006.

[38] Ian Fischer. The conditional entropy bottleneck. *arXiv preprint arXiv:2002.05379*, 2020.

[39] Ian Fischer and Alexander A Alemi. Ceb improves model robustness. *arXiv preprint arXiv:2002.05380*, 2020.

[40] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. In *ICLR*, 2016.

[41] The garage contributors. Garage: A toolkit for reproducible reinforcement learning research. `https://github.com/rlworkgroup/garage`, 2019.

[42] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15, pages 315–323, 2011.

[43] Anirudh Goyal, Riashat Islam, Daniel Strouse, Zafarali Ahmed, Matthew M Botvinick, Hugo Larochelle, Sergey Levine, and Yoshua Bengio. Infobot: Transfer and exploration via the information bottleneck. *ArXiv*, abs/1901.10902, 2019.

[44] Anirudh Goyal, Yoshua Bengio, Matthew M Botvinick, and Sergey Levine. The variational bandwidth bottleneck: Stochastic evaluation on an information budget. *ArXiv*, abs/2004.11935, 2020.

[45] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

[46] T. Haarnoja, Aurick Zhou, Kristian Hartikainen, G. Tucker, Sehoon Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications. *ArXiv*, abs/1812.05905, 2018.

[47] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[48] Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart Russell, and Anca Dragan. Inverse reward design. In *Advances in Neural Information Processing Systems*, 2017.

[49] I. Higgins, Loïc Matthey, A. Pal, C. Burgess, Xavier Glorot, M. Botvinick, S. Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017.

[50] R. Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

[51] Zhang-Wei Hong, Ge Yang, and Pulkit Agrawal. Bilinear value networks. In *10th International Conference on Learning Representations, ICLR 2022*, 2022.

[52] Jonathan J. Hunt, André Barreto, Timothy P. Lillicrap, and Nicolas Heess. Composing entropic policies using divergence correction. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 2911–2920. PMLR, 2019.

[53] Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cynthia H Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. In *NeurIPS*, 2019.

[54] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating

deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

[55] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, 2016.

[56] Michael Janner, Igor Mordatch, and Sergey Levine. Generative temporal difference learning for infinite-horizon prediction. *arXiv preprint arXiv:2010.14496*, 2020.

[57] Gregory Kahn, Adam Villaflor, Bosen Ding, Pieter Abbeel, and Sergey Levine. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[58] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *ArXiv*, abs/1806.10293, 2018.

[59] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2016.

[60] Hyunjik Kim and A. Mnih. Disentangling by factorising. In *ICML*, 2018.

[61] Jaekyeom Kim, Minjung Kim, Dongyeon Woo, and Gunhee Kim. Drop-bottleneck: Learning discrete compressed representation for noise-robust

exploration. In *International Conference on Learning Representations*, 2021.

[62] Jaekyeom Kim, Seohong Park, and Gunhee Kim. Unsupervised skill discovery with bottleneck option learning. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

[63] Jaekyeom Kim, Seohong Park, and Gunhee Kim. Constrained gpi for zero-shot transfer in reinforcement learning. In *Advances in Neural Information Processing Systems*, 2022.

[64] Youngjin Kim, Wontae Nam, Hyunwoo Kim, Ji-Hoon Kim, and Gunhee Kim. Curiosity-bottleneck: Exploration by distilling task-specific novelty. In *International Conference on Machine Learning*, pages 3379–3388, 2019.

[65] Diederik P Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[66] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, 2014.

[67] Artemy Kolchinsky, Brendan D Tracey, and David H Wolpert. Nonlinear information bottleneck. *Entropy*, 21(12):1181, 2019.

[68] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical review E*, 69(6):066138, 2004.

[69] Alex Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.

[70] Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. *ArXiv*, abs/2107.04034, 2021.

[71] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database, 2010.

[72] Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Raetsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. In *international conference on machine learning*, pages 4114–4124. PMLR, 2019.

[73] Chen Ma, Dylan R Ashley, Junfeng Wen, and Yoshua Bengio. Universal successor features for transfer reinforcement learning. *arXiv preprint arXiv:2001.04025*, 2020.

[74] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

[75] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

[76] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[77] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K

Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[78] Daniel Moyer, Shuyang Gao, Rob Brekelmans, Aram Galstyan, and Greg Ver Steeg. Invariant representations without adversarial training. In *Advances in Neural Information Processing Systems*, pages 9084–9093, 2018.

[79] Ofir Nachum, Shixiang Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. In *NeurIPS*, 2018.

[80] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-optimal representation learning for hierarchical reinforcement learning. In *International Conference on Learning Representations*, 2019.

[81] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, pages 807–814, 2010.

[82] Mark W. Nemecek and Ron Parr. Policy caches with successor features. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, pages 8025–8033. PMLR, 2021.

[83] Chris Nota. The autonomous learning library. `https://github.com/cpnota/autonomous-learning-library`, 2020.

[84] Seohong Park, Jongwook Choi, Jaekyeom Kim, Honglak Lee, and Gunhee Kim. Lipschitz-constrained unsupervised skill discovery. In *International Conference on Learning Representations*, 2022.

[85] Seohong Park, Kimin Lee, Youngwoon Lee, and Pieter Abbeel. Controllability-aware unsupervised skill discovery. *arXiv preprint arXiv:2302.05103*, 2023.

[86] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[87] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, 2017.

[88] Xue Bin Peng, Angjoo Kanazawa, Sam Toyer, Pieter Abbeel, and Sergey Levine. Variational discriminator bottleneck: Improving imitation learning, inverse rl, and gans by constraining information flow. *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

[89] Diederik M Roijers. *Multi-objective decision-theoretic planning.* PhD thesis, University of Amsterdam, 2016.

[90] Brian C Ross. Mutual information between discrete and continuous data sets. *PloS one*, 9(2):e87357, 2014.

[91] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[92] Nikolay Savinov, Anton Raichuk, Raphaël Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. Episodic curiosity through reachability. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

[93] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1312–1320. JMLR.org, 2015.

[94] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, L. Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588 7839:604–609, 2020.

[95] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[96] Karl Schulz, Leon Sixt, Federico Tombari, and Tim Landgraf. Restricting the flow: Information bottlenecks for attribution. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020.

[97] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations

from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[98] Archit Sharma, Michael Ahn, Sergey Levine, Vikash Kumar, Karol Hausman, and Shixiang Gu. Emergent real-world robotic skills via unsupervised off-policy reinforcement learning. In *Robotics: Science and Systems (RSS)*, 2020.

[99] Archit Sharma, Shixiang Gu, Sergey Levine, V. Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020.

[100] Pranav Shyam, Wojciech Jaśkowski, and Faustino Gomez. Model-based active exploration. In *International Conference on Machine Learning*, pages 5779–5788. PMLR, 2019.

[101] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[102] DJ Strouse and David J Schwab. The deterministic information bottleneck. *Neural computation*, 29(6):1611–1630, 2017.

[103] Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. In *International Conference on Learning Representations*, 2018.

[104] R. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112:181–211, 1999.

[105] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2015.

[106] Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

[107] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[108] A. S. Vezhnevets, Simon Osindero, T. Schaul, N. Heess, Max Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *ICML*, 2017.

[109] Xiaofei Wang, Kimin Lee, Kourosh Hakhamaneshi, Pieter Abbeel, and Michael Laskin. Skill preferences: Learning to extract and execute robotic skills from human feedback. In *Conference on Robot Learning*, pages 1259–1268. PMLR, 2022.

[110] Jesse Zhang, Haonan Yu, and W. Xu. Hierarchical reinforcement learning by discovering intrinsic options. *ArXiv*, abs/2101.06521, 2021.

Acknowledgements

I extend my heartfelt appreciation to my advisor Prof. Gunhee Kim for the valuable guidance and for being a great mentor throughout my Ph.D. study. I am also sincerely grateful to my thesis committee members – Prof. Sungjoo Yoo, Prof. Hyun Oh Song, Prof. Joonseok Lee and Dr. Kimin Lee – for their insightful and helpful advice on this thesis.

I was lucky enough to have worked with my great collaborators: Seohong Park, Jongwook Choi, Hyoungseok Kim, Minjung Kim, Dongyeon Woo, Yeonwoo Jeong, Yeda Song and Julian Paquerot. I also want to appreciate all my colleagues at SNU's Vision and Learning Laboratory: Junhyug Noh, Youngjae Yu, Byeongchang Kim, Soochan Lee, Sangho Lee, Wonhee Lee, Insu Jeon, Minui Hong, Joonil Na, Jongseok Kim, Wonkwang Lee, Taeyoung Hahn, Junsoo Ha, Chris Dongjoo Kim, Jiwan Chung, Hyunwoo Kim, Myeongjang Pyeon, Jinseo Jeong, Jaewoo Ahn, Heeseung Yun, Sungeun Kim, Sangwoo Moon, Jihwan Moon, Seokhee Hong, Eunkyu Park, Dohyun Kim, Junseo Koo, Sehun Lee, Fatemeh Pesaran, Keighley Overbay, Seungyoon Woo, Hyungyu Seo, Dayoon Ko and Taehyun Lee. It was so much fun to hang out with all of them, and I will always cherish the memories of the time we spent together.

Last but not least, I want to deeply thank my family for the love, encouragement and support they have given me.

요약

딥러닝 분야의 많은 연구자들은 다양한 작업을 수행할 수 있는 에이전트(agent)를 만들고자 해왔다. 가능한 모든 작업에 대해 학습을 수행하는 것은 보통 불가능하기 때문에, 에이전트가 학습용 작업에서 배운 것을 기반으로 새로운 작업에 대해 더 잘 일반화하도록 하는 것은 딥러닝의 중요한 과제 중 하나이다. 효과적인 일반화에는, 서로 다른 조건 하에서 사용될 수 있는 추상화(abstraction)를 학습하는 것과 그러한 추상화를 새로운 작업에 잘 활용하는 것, 두 가지 모두가 중요하다. 본 학위논문에서는 이러한 일반화 과제를 위와 같이 추상화(abstraction)와 전이(transfer), 이 두 가지 관점에서 다룬다.

먼저 입력 데이터를 추상화하고 잡음(noise)에 강건한 특징(features)을 학습하는 것에 대해 다룬다. 추론 시점에 주어지는, 작업과 무관한 정보는 학습된 모델과 에이전트의 성능에 큰 영향을 미칠 수 있기 때문에, 그러한 잡음에 대한 강건성을 갖도록 하는 것은 일반화에서 중요한 문제 중 하나이다. 이러한 문제를 해결하기 위해, 작업 변수와 무관한 특징을 이산적으로(discretely) 제거하고 원하는 특징을 남기는, *Drop-Bottleneck*이라는 이산적 정보 병목(discrete information bottleneck) 방법론을 제안한다. 이 방법론은 단순한 정보 압축 목표(objective)를 가지며 결정론적인 압축된 표현 또한 제공하는데, 이는 온전한 일관성 및 줄어든 특징 수에 따른 향상된 효율성을 필요로 하는 추론에 유용하다.

또한, 주어진 환경에서 에이전트가 지도(supervision) 없이 가능한 행동을 발견하고 더 재사용 가능한 형태의 스킬(skill)로 추상화하는 것을 다룬다. 비지도적 스킬 발견은 외부적 보상 없이 환경과 상호작용하며 유용한 행동을 찾고 학습하는 것을 목표로 한다. 이는 학습된 스킬의 지식을 재사용하고 새로운 작업을 더 효율적이고 효과적으로 수행할 수 있도록 하기 때문에, 강화학습에서 시간적 추상화의

중요한 과제 중 하나이다. 해당 목표를 위해 본 논문에서는 *Information Bottleneck Option Learning (IBOL)*이라는 비지도적 스킬 발견 방법론을 제시한다. 이것은 환경을 선형화하여 상태 공간에서 더 광범위한 행동을 찾고, 스킬의 재사용성을 향상시키기 위해 정보 병목을 통해 해당 행동들의 얽히지 않은(disentangled) 추상화를 학습한다.

마지막으로, 원천 작업에서 학습한 지식을, 추가적인 학습 없이 대상 작업에서의 성능을 향상시키는데 활용하는 방법을 다룬다. 강화학습에서 작업들 사이에 보상 함수가 달라지는 조건에서의 제로샷(zero-shot) 전이에는 후속 특징(successor features) 프레임워크가 많이 사용된다. 본 학위논문에서는 후속 특징을 이용하는 학습된 가치(value) 근사기의 새로운 작업으로의 전이를, 해당 작업에서의 오류를 제한함으로써 향상시키는 것을 목적으로 한다. 원천 작업 및 해당 작업들에서 학습된 후속 특징들을 이용해, 원천 작업 벡터들의 선형 결합으로 표현 가능한 새로운 작업 벡터에서의 최적 가치에 대한 하한 및 상한을 제시한다. 그리고 *constrained GPI*라는, 새로운 대상 작업에서의 가치 근삿값을 제한하여 전이를 향상시키는 단순한 시험 시점 방법론을 제시한다.

주요어: 딥러닝, 심층 강화학습, 스킬 발견, 시간적 추상화, 전이 학습
학번: 2018-28413