Learning Options via Compression

Anonymous Author(s)
Affiliation
Address
email

Abstract

Identifying statistical regularities in solutions to some tasks in multi-task reinforcement learning can accelerate learning new tasks. Skill learning offers one way of extracting these regularities by decomposing pre-collected experience into a sequence of skills. A popular approach to skill learning is maximizing the likelihood of the pre-collected experience with latent variable models. However, there are often many different solutions that maximize the likelihood equally well, including degenerate solutions. To address this underspecification, we propose a new objective that combines the maximum likelihood objective with a penalty on the description length of the skills. This penalty incentivizes the skills to maximally identify and extract common structure from the experiences. We demonstrate the effectiveness of our method on a multi-task benchmark from prior work. Further, while most prior works in the off-line multi-task setting focus on low-dimensional tasks, we demonstrate that our method can scale to challenging tasks with image observations. Additionally, the acquired skills can be used to solve downstream tasks with up to $8\times$ fewer samples compared to skills from maximizing likelihood.

1 Introduction

Reinforcement learning (RL) algorithms for solving tasks *tabula rasa* are sample inefficient [27, 8]. However, many domains, such as autonomous driving [10] and robotics [15] contain a wealth of pre-collected experience for various tasks, which can be leveraged to quickly solve new tasks. This pre-collected experience can be used in numerous ways, including learning a dynamics model [18, 35], learning compact representations of observations [80], learning behavioral priors [72], and extracting meaningful skills or, more specifically, options [74, 39, 2, 55] for hierarchical reinforcement learning (HRL) [6]. Our work studies this last approach.

Given the experience, skill learning methods extract commonly occurring behaviors. For example, in an indoor navigation task, skills may correspond to navigating to different rooms. Once learned, these skills can be used in place of low-level actions to quickly solve new tasks via planning [65, 47] or RL [51, 44, 48].

Prior methods learn skills by maximizing the likelihood of the pre-collected experience [40, 39, 2, 83]. However, this maximum likelihood objective (or the lower bounds on it) is *underspecified*: it often admits many solutions, only some of which would help learning new tasks (see Figure 1). For example, one degenerate solution is to learn a single skill for each entire trajectory; another degenerate solution learns a separate skill for each action. Both of these decompositions can perfectly reconstruct the pre-collected experiences (and, hence, maximize likelihood), but they are of limited use for learning to solve new tasks. Empirically, we observe the underspecification problem can occur even on simple tasks: maximizing likelihood yields unnatural decompositions that achieve the same likelihood as semantically meaningful decompositions.

We will address this underspecification problem by introducing an additional objective which biases skill learning to acquire skills that can accelerate the learning of new tasks. To construct such an objective, we make two key observations:

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

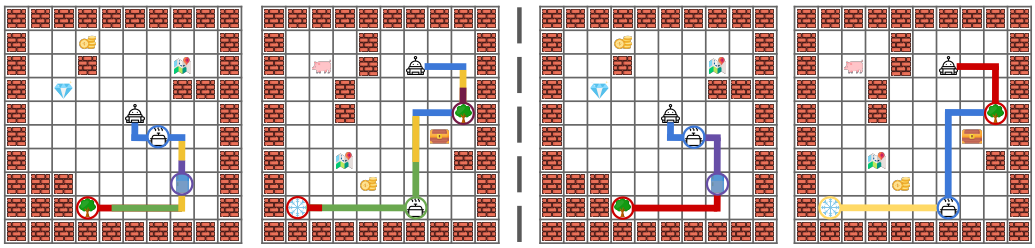

Figure 1: The maximum likelihood objective is *underspecified*: the skills on both the left and the right maximize the likelihood (i.e., encode the data), but the skills on the right are more likely to be useful for a new task. *Compressing* the data requires factoring out common temporal structures, like the skills on the right. The visualized tasks are from Kipf et al. [39].

- Skills (i.e., options) must extract reusable temporally extended structure from the pre-collected experiences to accelerate learning new tasks.
- *Compressing* data also requires identifying and extracting common structure.

Hence, we hypothesize that compression is a good objective for skill learning. We formalize this notion of compression with the information-theoretic principle of *minimum description length* (MDL) [64]. Concretely, we combine the maximum likelihood objective from prior work with a new term to minimize the number of bits required to represent the pre-collected experience with skills. Maximizing likelihood forces the skills to encode the experience. Then our compression objective forces the skills to find common structure by our second observation. Crucially, while methods related to compression typically involve discrete optimization and hence are not differentiable, we also provide a fully differentiable model built on top of variational temporal abstraction (VTA) [36], which optimizes our objective (Section 4).

Overall, our main contribution is an objective for extracting skills from offline experience, which addresses the underspecification problem with the maximum likelihood objective. We call the resulting approach LOVE (**L**earning **O**ptions **V**ia compr**E**ssion). Using the standard multi-task benchmark from prior work [39], we demonstrate that LOVE learns skills that enable faster RL and are more semantically meaningful compared to prior methods. While prior works focus on tasks with low-dimensional observations, we demonstrate that our regularizer allows LOVE to scale to tasks with image observations.

2 Related Works

We study the problem of using offline experience from one set of tasks to quickly learn new tasks. While we use the experience to learn skills, prior works also consider other approaches of leveraging the offline experience, including to learn a dynamics model [35], to learn a compact representation of observations [80], and to learn behavioral priors for exploring new tasks [72].

We build on a rich literature on extracting skills [74] from *offline* experience [58, 21, 40, 69, 39, 7, 62, 85, 84, 43, 2, 67, 66, 55, 83, 49, 86, 63, 75, 81]. These works predominantly learn skills using a latent variable model, where the latent variables partition the experience into skills, and the overall model is learned by maximizing (a lower bound on) the likelihood of the experiences. This approach is structurally similar to a long line of work in hierarchical Bayesian modeling [22, 9, 20, 46, 34, 14, 45, 36, 25, 16]. We also follow this approach and build off of VTA [36] for the latent variable model, but differ by introducing a new compression term to address the underspecification problem with maximizing likelihood. Zhang et al. [83] show that maximizing likelihood with variational inference can itself be seen as a form of compression, but their proposed approach compresses the latent variables at each time step, which does in general yield optimal compression as our compression objective does, and further only learns open-loop policies that do not condition on the state.

Beyond learning skills from offline experience, prior works also consider learning skills with additional supervision [3, 59, 70, 32, 73], from online interaction without reward labels [23, 19, 17, 68, 5], and from online interaction with reward labels [41, 4, 78, 26, 54].

3 Preliminaries

Problem setting. We consider the problem of using an offline dataset of experience to quickly solve new RL tasks, where each task is identified by a reward function. The offline dataset is a set of trajectories $\mathcal{D} = \{\tau_i\}_{i=1}^N$, where each trajectory is a sequence of states $\boldsymbol{x} \in \mathcal{X}$ and actions $\boldsymbol{a} \in \mathcal{A}$: $\tau_i = \{\boldsymbol{x}_0, \boldsymbol{a}_0, \boldsymbol{x}_1, \dots\}$. Each trajectory is collected by some unknown policy interacting with a Markov decision process (MDP) with dynamics $\mathcal{P}(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t, \boldsymbol{a}_t)$. Following prior work [18, 1, 15], we do not assume access to the rewards (i.e., the task) of the offline trajectories.

Using this dataset, our aim is to quickly solve a new task. The new task involves interacting with the same MDP as the data collection policy in the offline dataset, except with a new reward function $\mathcal{R}(\boldsymbol{x}, \boldsymbol{a})$. The objective is to maximize the expected returns in as few time steps of interaction as possible. Importantly, this reward function is only provided *after* the method has extracted skills from the offline dataset.

Variational Temporal Abstraction. Our method builds upon variational temporal abstraction (VTA) [36], which is a method for decomposing non-control sequential data (i.e., data without actions) into subsequences. We overview VTA below and extend it to handle actions it Section 4.1.

VTA assumes that each observation $\boldsymbol{x}_t$ is associated with an abstract representation $\boldsymbol{s}_t$ and a subsequence descriptor $\boldsymbol{z}_t$. An additional, binary random variable $\boldsymbol{m}_t$ indicates whether a new subsequence begins at the current observation. VTA uses the following generative model:

1. Determine whether to begin a new subsequence: $\boldsymbol{m}_t \sim p(\boldsymbol{m}_t \mid \boldsymbol{s}_{t-1})$. A new subsequence always begins on the first time step, i.e., $\boldsymbol{m}_1 = 1$.
2. Sample the descriptor: $\boldsymbol{z}_t \sim p(\boldsymbol{z}_t \mid \boldsymbol{z}_{<t}, \boldsymbol{m}_t)$. If $\boldsymbol{m}_t = 0$, the previous descriptor is copied, i.e., $\boldsymbol{z}_t = \boldsymbol{z}_{t-1}$.
3. Sample abstract representation: $\boldsymbol{s}_t \sim p(\boldsymbol{s}_{<t} \mid \boldsymbol{z}_t, \boldsymbol{m}_t)$.
4. Sample the observation: $\boldsymbol{x}_t \sim p(\boldsymbol{x}_t \mid \boldsymbol{s}_t)$.

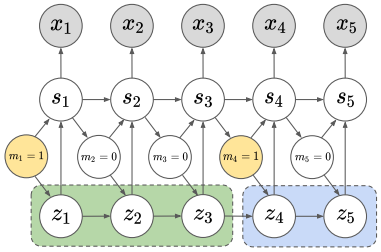

Figure 2: The VTA graphical model (adapted from Kim et al. [36]). The model decomposes the data $x_{1:T}$ into subsequences demarcated by when the boundary variables $m_t = 1$ (highlighted in yellow). Each subsequence is assigned a descriptor $\boldsymbol{z}$, where all $\boldsymbol{z}_t$ within a subsequence are the same (outlined in dashed lines).

We visualize this sampling procedure in Figure 2. Formally, we can write this generative model as:

$$p(\boldsymbol{x}_{1:T}, \boldsymbol{z}_{1:T}, \boldsymbol{s}_{1:T}, \boldsymbol{m}_{1:T}) = \prod_{t=1}^{T} p(\boldsymbol{x}_t \mid \boldsymbol{s}_t) p(\boldsymbol{m}_t \mid \boldsymbol{s}_{t-1}) p(\boldsymbol{s}_t \mid \boldsymbol{s}_{<t}, \boldsymbol{z}_t, \boldsymbol{m}_t) p(\boldsymbol{z}_t \mid \boldsymbol{z}_{<t}, \boldsymbol{m}_t).$$

VTA maximizes the likelihood of the observed data under this latent variable model using the standard evidence lower bound (ELBO):

$$\log p(\boldsymbol{x}_{1:T}) \geq \sum_{\boldsymbol{m}_{1:T}} \int_{\substack{\boldsymbol{s}_{1:T}, \\ \boldsymbol{z}_{1:T}}} q_{\boldsymbol{\theta}}(\boldsymbol{z}_{1:T}, \boldsymbol{s}_{1:T}, \boldsymbol{m}_{1:T} \mid \boldsymbol{x}_{1:T}) \cdot \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}_{1:T}, \boldsymbol{z}_{1:T}, \boldsymbol{s}_{1:T}, \boldsymbol{m}_{1:T})}{q_{\boldsymbol{\theta}}(\boldsymbol{z}_{1:T}, \boldsymbol{s}_{1:T}, \boldsymbol{m}_{1:T} \mid \boldsymbol{x}_{1:T})}$$

This lower bound holds for any choice of $q_{\boldsymbol{\theta}}$. VTA chooses to factor this distribution as:

$$q_{\boldsymbol{\theta}}(\boldsymbol{z}_{1:T}, \boldsymbol{s}_{1:T}, \boldsymbol{m}_{1:T} \mid \boldsymbol{x}_{1:T}) = \prod_{t=1}^{T} q_{\boldsymbol{\theta}}(\boldsymbol{m}_t \mid \boldsymbol{x}_{1:t})\, q_{\boldsymbol{\theta}}(\boldsymbol{z}_t \mid \boldsymbol{z}_{t-1}, \boldsymbol{m}_t, \boldsymbol{x}_{1:T})\, q_{\boldsymbol{\theta}}(\boldsymbol{s}_t \mid \boldsymbol{z}_t, \boldsymbol{m}_t, \boldsymbol{x}_{1:t}).$$

As discussed above and as we observe in Section 6, the maximum likelihood objective is underspecified and may yield degenerate or unhelpful solutions. We address this in the next section with a new objective that compresses the latent variables $\boldsymbol{z}$, while also maximizing the ELBO.

4 LOVE: Learning Options via Compression

In this section, we describe our method for learning skills from pre-collected experience. We first extend VTA to handle experience labeled with actions, and then introduce a compression objective that encourages extracting common structure from the experience.

4.1 A Model for Interaction Data

We now extend the VTA model [36] to handle sequential data labeled with actions, where descriptors $\boldsymbol{z}$ now represent skills. From a high level, the model partitions a trajectory into subsequences with the boundary variables $\boldsymbol{m}$ and labels each subsequence as a skill $\boldsymbol{z}$.

We wish to learn a state-conditional policy rather than the joint distribution of the whole trajectory. To do this, we write a new generative model for the actions $\boldsymbol{a}_{1:T}$ conditional on the state $\boldsymbol{x}_{1:T}$:

$$p(\boldsymbol{z}_{1:T}, \boldsymbol{s}_{1:T}, \boldsymbol{m}_{1:T}, \boldsymbol{a}_{1:T} \mid \boldsymbol{x}_{1:T}) = \prod_{t=1}^{T} p(\boldsymbol{a}_t \mid \boldsymbol{s}_t) p(\boldsymbol{m}_t \mid \boldsymbol{s}_{t-1}) p(\boldsymbol{s}_t \mid \boldsymbol{x}_t, \boldsymbol{z}_t) p(\boldsymbol{z}_t \mid \boldsymbol{x}_t, \boldsymbol{z}_{t-1}, \boldsymbol{m}_{t-1}).$$

This differs from the original VTA generative model in two ways: (1) we introduce a $p(\boldsymbol{a}_t \mid \boldsymbol{s}_t)$ term that indicates how actions are sampled; and (2) the distributions over $\boldsymbol{z}_t$ and $\boldsymbol{s}_t$ do not depend on all previous $\boldsymbol{z}_{<t}$ and $\boldsymbol{s}_{<t}$ to encode the Markov property. We then augment the variational distribution such that the posterior over skills $\boldsymbol{z}$ also depends on actions:

$$q_{\boldsymbol{\theta}}(\boldsymbol{z}_{1:T}, \boldsymbol{s}_{1:T}, \boldsymbol{m}_{1:T} \mid \boldsymbol{x}_{1:T}, \boldsymbol{a}_{1:T}) = \prod_{t=1}^{T} \underbrace{q_{\boldsymbol{\theta}}(\boldsymbol{m}_t \mid \boldsymbol{x}_{1:t})}_{\substack{\text{termination}\\ \text{policy}}} \underbrace{q_{\boldsymbol{\theta}}(\boldsymbol{z}_t \mid \boldsymbol{z}_{t-1}, \boldsymbol{m}_t, \boldsymbol{x}_{1:T}, \boldsymbol{a}_{1:T})}_{\text{skill posterior}} \underbrace{q_{\boldsymbol{\theta}}(\boldsymbol{s}_t \mid \boldsymbol{z}_t, \boldsymbol{x}_t)}_{\text{state abstraction posterior}} .$$

Overall, this yields a model with 3 learned components:

- A *state abstraction posterior* $q_{\boldsymbol{\theta}}(\boldsymbol{s}_t \mid \boldsymbol{z}_t, \boldsymbol{x}_t)$ and *decoder* $p_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t)$, which together form a distribution over actions conditioned on the skill $\boldsymbol{z}_t$ and current state $\boldsymbol{x}_t$. We refer to these jointly as the *skill policy* $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{z}_t, \boldsymbol{x}_t) = \mathbb{E}_{\boldsymbol{s}_t \sim q_{\boldsymbol{\theta}}(\boldsymbol{s}_t \mid \boldsymbol{z}_t, \boldsymbol{x}_t)} [p_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t)]$.
- A *termination policy* $q_{\boldsymbol{\theta}}(\boldsymbol{m}_t \mid \boldsymbol{x}_{1:t})$, which determines whether the previous skill terminates or not.
- A *skill posterior* $q_{\boldsymbol{\theta}}(\boldsymbol{z}_t \mid \boldsymbol{z}_{t-1}, \boldsymbol{m}_t, \boldsymbol{x}_{1:T}, \boldsymbol{a}_{1:T})$, which outputs the current skill $\boldsymbol{z}_t$ conditioned on all states $\boldsymbol{x}_{1:T}$ and actions $\boldsymbol{a}_{1:T}$. This depends on $\boldsymbol{m}_t$ and $\boldsymbol{z}_{t-1}$, since the boundary variables determine whether the previous skill $\boldsymbol{z}_{t-1}$ terminates: when $\boldsymbol{m}_t = 0$, the previous skill does not terminate and $\boldsymbol{z}_t = \boldsymbol{z}_{t-1}$.

Once this model is learned, the skill policy and termination policy represent the skills, without a need for the skill posterior: Given a skill variable $\boldsymbol{z}$, the skill policy encodes what actions the skill takes, and the termination policy determines when the skill stops. In the next section, we describe our objective for learning this model. Then, in Section 5, we describe how we use the skill policy and termination policy to learn new tasks.

4.2 Discovering Structure via Compression

As previously discussed, the maximum likelihood objective is underspecified for skill learning, because many skills can maximize likelihood, independent of whether they are useful for learning new tasks. In this section, we address this with a new objective that attempts to measure how useful skills will be for learning new tasks in terms of compression. Our objective is based on the intuition that effectively compressing a sequence of data requires factoring out common structure, and factoring out common structure is critical for learning useful skills.

We measure the complexity of a learned skill decomposition as the amount of information required to communicate the sequence of skills that encode the pre-collected experience. The latent variable model introduced in the previous section encodes each trajectory as a sequence of skills $\boldsymbol{z}_{1:T}$ and boundaries $\boldsymbol{m}_{1:T}$. However, using the skill and termination policies, it is possible to recover each trajectory from only the skill variables at the boundary points: i.e., at the time steps $\{t \in [T] \mid \boldsymbol{m}_t = 1\}$. For a prior on skills $p_{\boldsymbol{z}}$, an optimal code requires $-\log p_{\boldsymbol{z}}(\boldsymbol{z}_t)$ bits to send a skill $\boldsymbol{z}_t$ [13, Chapter 5.2]. Hence, the expected code length of communicating a trajectory $\boldsymbol{x}_{1:T}$ is:

$$\textsc{InfoCost}(\boldsymbol{\theta}; p_{\boldsymbol{z}}) = -\mathbb{E}_{\substack{\boldsymbol{x}_{1:T},\\ \boldsymbol{m}_{1:T},\\ \boldsymbol{z}_{1:T}}} \left[\sum_{t=1}^{T} \log p_{\boldsymbol{z}}(\boldsymbol{z}_t) \boldsymbol{m}_t \right],$$

where the expectation is under trajectories $(\boldsymbol{x}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{x}_T)$ from the pre-collected experience and sampling $\boldsymbol{z}_{1:T}$ from the skill posterior and $\boldsymbol{m}_{1:T}$ from the termination policy. The choice of prior that minimizes the average code length is one that equals the empirical distribution of skills under the pre-collected experience [13, Chapter 5.3]:

$$p^{\star}_{\boldsymbol{z}} \triangleq \arg\min_{p_{\boldsymbol{z}}} \textsc{InfoCost}(\boldsymbol{\theta}; p_{\boldsymbol{z}}), \quad \text{where } p^{\star}_{\boldsymbol{z}}(\boldsymbol{z}) = \frac{1}{n_{\mathrm{s}}} \mathbb{E}_{\substack{\boldsymbol{x}_{1:T},\\ \boldsymbol{m}_{1:T},\\ \boldsymbol{z}_{1:T}}} \left[\sum_{t=1}^{T} \delta(\boldsymbol{z}_t = \boldsymbol{z}) \boldsymbol{m}_t \right],$$

and $n_{\mathrm{s}} \triangleq \mathbb{E}_{\boldsymbol{x}_{1:T}, \boldsymbol{m}_{1:T}} \left[\sum_{t=1}^{T} \boldsymbol{m}_t \right]$. In Appendix A.1, we show that this marginal $p^{\star}_{\boldsymbol{z}}$ is a proper density for both continuous and discrete $\boldsymbol{z}$. Substituting in this optimal choice of prior, we can show that the code length can be expressed as the marginal entropy $\mathcal{H}_{p^{\star}_{\boldsymbol{z}}}[\boldsymbol{z}]$ times the number of skills per trajectory n_{s} (see Appendix A.2 for proof):

$$\mathcal{L}_{\mathrm{CL}}(\boldsymbol{\theta}) \triangleq \min_{p(\boldsymbol{z}_t)} \textsc{InfoCost}(\boldsymbol{\theta}; p(\boldsymbol{z}_t)) = -\mathbb{E}_{\substack{\boldsymbol{x}_{1:T},\\ \boldsymbol{m}_{1:T},\\ \boldsymbol{z}_{1:T}}} \left[\sum_{t=1}^{T} \log p^{\star}_{\boldsymbol{z}}(\boldsymbol{z}_t) \boldsymbol{m}_t \right] = n_{\mathrm{s}} \mathcal{H}_{p^{\star}_{\boldsymbol{z}}}[\boldsymbol{z}]. \tag{1}$$

Intuitively, this is equal to the average code length of a skill multiplied by the the average number of skills per trajectory. Note that compression is only beneficial if the model achieves high likelihood of the data. We capture this by solving the following constrained optimization problem:

$$\min_{\boldsymbol{\theta}} \; \mathcal{L}_{\text{CL}}(\boldsymbol{\theta}) \quad \text{s.t. } \mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}) \leq C, \tag{2}$$

where $\mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta})$ a negated evidence lower bound on the log-likelihood (detailed in Appendix B).

Remarks. We discuss a connection between this objective and variational inference in Appendix A.3. Additionally, while this work focuses on the RL setting, our objective generally applies to sequential modeling problems.

4.3 Connections to Minimal Description Length

Our approach closely relates to the minimum description length (MDL) principle [64]. This principle equates *learning* with *finding regularity* in data, which can be used to *compress* the data. Informally, the best model is the one that encodes the data with the lowest description length. Given a model $\boldsymbol{\theta}$ that encodes the data $\mathcal{D}$ into some message, one way to formalize the description length of the data $L(\mathcal{D})$ is with a crude two-part code [24]. This decomposes the description length of the data as the length of the message plus the length of the model:

$$L(\mathcal{D}) = \underbrace{L(\mathcal{D} \mid \boldsymbol{\theta})}_{\text{message length}} + \underbrace{L(\boldsymbol{\theta})}_{\text{model length}} . \tag{3}$$

In our case, the message is the latent variables $\boldsymbol{z}_t$ at the boundary points $\{t \in [T] \mid \boldsymbol{m}_t = 1\}$, which can be decoded into the data $\mathcal{D}$ with the skill and termination policies (representing the model). Hence, our approach can be seen as an instance of minimizing the description length $L(\mathcal{D})$. Optimizing our objective in Equation 1 is equivalent to minimizing the message length $L(\mathcal{D} \mid \boldsymbol{\theta})$. While we do not directly attempt to minimize the model length term $L(\boldsymbol{\theta})$, many works [56, 76] indicate that deep learning has *implicit regularization*, which biases optimization toward low complexity solutions without an explicit regularizer. In general, computing the true description length or appropriate notion of complexity for neural networks is a tall order [57, 82, 33]. However, there is a rich space of methods to be explored here and we therefore leave the extension of our approach to directly minimizing the model length term for future work.

4.4 A Practical Implementation

Model. We instantiate our model by defining discrete skills $\boldsymbol{z} \in [K]$, state abstractions $\boldsymbol{s} \in \mathbb{R}^d$, and binary boundary variables $\boldsymbol{m} \in \{0, 1\}$. We parameterize all components of our model as neural networks. See Appendix D for architecture details.

Optimization. We apply Gumbel-softmax [31, 50] to optimize over the discrete random variables $\boldsymbol{z}$ and $\boldsymbol{m}$. We rewrite the compression objective $\mathcal{L}_{\text{CL}}$ as a product of n_{s} and $\mathcal{H}_{p^\star_{\boldsymbol{z}}}[\boldsymbol{z}_t]$ in Equation 1 to improve the stability of optimization. This rewriting allows computing a gradient in terms of a distribution over the skill variables $p^\star_{\boldsymbol{z}}$, rather than samples $\boldsymbol{z}_t$, which yields a more accurate finite sample gradient. Empirically, this leads to stabler optimization and convergence to better solutions.

We approximate the optimal skill prior $p^\star_{\boldsymbol{z}}$ using our skill posterior as:

$$p^\star_{\boldsymbol{z}}(\boldsymbol{z}) \approx \frac{1}{\widehat{\mathbb{E}}_{\substack{\boldsymbol{x}_{1:T},\\ \boldsymbol{m}_{1:T}}}\left[\sum_{t=1}^{T} \boldsymbol{m}_t\right]} \widehat{\mathbb{E}}_{\substack{\boldsymbol{x}_{1:T},\\ \boldsymbol{m}_{1:T}}}\left[\sum_{t=1}^{T} q_{\boldsymbol{\theta}}(\boldsymbol{z}_t = \boldsymbol{z} \mid \boldsymbol{z}_{t-1}, \boldsymbol{m}_t, \boldsymbol{x}_{1:T}, \boldsymbol{a}_{1:T})\, \boldsymbol{m}_t\right],$$

where $\widehat{\mathbb{E}}$ denotes the empirical expectation over minibatches of $\boldsymbol{x}_{1:T}, \boldsymbol{a}_{1:T}$ from $\mathcal{D}$ and sampling $\boldsymbol{m}_{1:T}$ from the termination policy. We solve the constrained optimization problem (Equation 2) with dual gradient descent on the Lagrangian. In Appendix G, we summarize the overall training procedure in Algorithm 1 and report details about the Lagrangian.

Enforcing a minimum skill length. Though degenerate solutions such as skills that only take a single action achieve a low value in our compression objective, empirically, such solutions create local optima that are difficult to escape. To avoid this, we mask the boundary variables during training to ensure that each skill z_t operates for at least $T_{\text{min}} = 3$ time steps. We find that when the skills are at least minimally temporally extended, optimization of the compression objective appears to be stabler and achieves better values. We remove these masks at test time, when we learn a task with our skills.

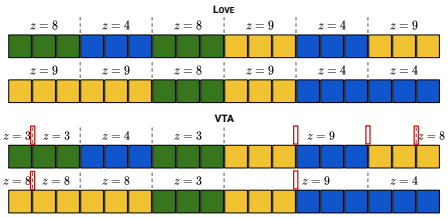

Figure 3: Learned boundaries and descriptors on *Simple Colors*. LOVE recovers the patterns, while VTA learns boundaries that break up patterns or span multiple patterns (outlined in red).

	Simple Colors		*Conditional Colors*	
	VTA	**LOVE**	**VTA**	**LOVE**
$\mathcal{L}_{\text{ELBO}}$	2868 ± 43	2838 ± 19	2832 ± 7.7	2827 ± 1.9
Precision	0.87 ± 0.19	$\mathbf{0.99} \pm 0.01$	0.84 ± 0.22	$\mathbf{0.99} \pm 0.01$
Recall	0.79 ± 0.13	$\mathbf{0.85} \pm 0.03$	$\mathbf{0.82} \pm 0.16$	$\mathbf{0.83} \pm 0.06$
F1	0.82 ± 0.13	$\mathbf{0.91} \pm 0.02$	0.83 ± 0.19	$\mathbf{0.90} \pm 0.03$
Code Length	7.58 ± 1.3	$\mathbf{6.34} \pm 0.75$	9.17 ± 1.76	$\mathbf{6.83} \pm 0.51$

Table 1: Effect of LOVE on the *Simple Colors* and *Conditional Colors* datasets (5 seeds). Indicative of the underspecification problem, all approaches achieve similar values for the likelihood objective, but LOVE better recovers the correct boundaries.

5 Using the Learned Skills for HRL

We now describe how we quickly learn new tasks, given the skills learned from the pre-collected experience. Overall, we simply augment the agent's action space with the learned skills and learn a new policy that can take either low-level actions or learned skills, similar to Kipf et al. [39]. However, our compression objective may result in unused skills, since this decreases the encoding cost of the used skills. Therefore, we first filter down to only the skills that are used to compress the pre-collected experience by selecting the skills where the marginal is over some threshold α:

$$\mathcal{Z} = \left\{ k \in [K] \;\middle|\; p^{\star}_{\boldsymbol{z}}(k) > \alpha \right\}.$$

Then, on a new task with action space $\mathcal{A}$, we train an agent using the augmented action space $\mathcal{A}^{+} = \mathcal{A} \cup \mathcal{Z}$. When the agent selects a skill $z \in \mathcal{Z}$, we follow the procedure in Algorithm 2 in the Appendix. We take actions following the skill policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{z}, \boldsymbol{x}_t)$ (lines 2–3), until the termination policy $q_{\boldsymbol{\theta}}(\boldsymbol{m}_t \mid \boldsymbol{x}_{1:t})$ outputs a termination $\boldsymbol{m}_t = 1$ (line 5). At that point, the agent observes the next state $\boldsymbol{x}_{t+1}$ and selects the next action or skill.

6 A Didactic Experiment: Frame Prediction

Before studying the RL setting, we first illustrate the effect of LOVE in the simpler setting of sequential data without actions. In this setting, we use the original VTA model, which partitions a sequence $\boldsymbol{x}_{1:T}$ into contiguous subsequences with the boundary variables $\boldsymbol{m}$ and labels each subsequence with a descriptor $\boldsymbol{z}$, as described in Section 3. Here, VTA's objective is to maximize the likelihood of the sequence $\boldsymbol{x}_{1:T}$, and our compression objective applies to this setting completely unmodified. We compare LOVE and only maximizing likelihood, i.e., VTA [36], by measuring how well the learned subsequences correspond to underlying patterns in the data. To isolate the effect of the compression objective, we do not enforce a minimum skill length.

Dataset. We consider two simple datasets *Simple Colors* and *Conditional Colors*, which consist of sequences of 32×32 monochromatic frames, with repeating underlying patterns. *Simple Colors* consists of 3 patterns: 3 consecutive yellow frames, 3 consecutive blue frames and 3 consecutive green frames. The dataset is generated by sampling these patterns with probability 0.4, 0.4, and 0.2 respectively and concatenating the results. Each pattern is sampled independent of history. The dataset is then divided into sequences of 6 patterns, equal to $6 \times 3 = 18$ frames.

In contrast to *Simple Colors*, *Conditional Colors* tests learning patterns that depend on history. It consists of 4 patterns: the 3 patterns from *Simple Colors* with an additional pattern of 3 consecutive purple frames. The dataset is generated by uniformly sampling the patterns from *Simple Colors*. To make the current timestep pattern dependent on the previous timestep, the yellow frames are re-colored to purple, if the previous pattern was blue or yellow. As in *Simple Colors*, the dataset is divided into sequences of 6 patterns.

In both datasets, the optimal encoding strategy is to learn 3 subsequence descriptors (i.e., skills), one for each pattern in *Simple Colors*. In *Conditional Colors*, the descriptor corresponding to yellow in *Simple Colors* either outputs yellow or purple, depending on the history. This encoding strategy achieves an expected code length of 6.32 nats in *Simple Colors* and 6.59 nats in *Conditional Colors*.

Results. We visualize the learned descriptors and boundaries in Figure 3. LOVE successfully segments the sequences into the patterns and assigns a consistent descriptor $\boldsymbol{z}$ to each pattern. In contrast, despite the simple structure of this problem, VTA learns subsequences that span over multiple patterns or last for fewer timesteps than a single pattern.

Quantitatively, we measure the (1) precision, recall, and F1 scores of the boundary prediction, (2) the ELBO of the maximum likelihood objective and (3) the average code length $\mathcal{L}_{\text{CL}}$ in Table 1. While both VTA and LOVE achieve approximately the same negated ELBO (lower is better), LOVE recovers the correct boundaries with much higher precision, recall, and F1 scores. This illustrates that the underspecification problem can occur even in simple sequential data. Additionally, we find that LOVE achieves an encoding cost close to the optimal value. One interesting, though rare, failure mode is that LOVE sometimes even achieves a lower encoding cost than the optimal value by over-weighting the compression term and imperfectly reconstructing the data. In Appendix C, we conduct a ablation study on the weights of $\mathcal{L}_{\text{CL}}$ in optimization.

7 Experiments

We aim to answer three questions: (1) Does LOVE learn semantically meaningful skills? (2) Do skills learned with LOVE accelerate learning for new tasks? (3) Can LOVE scale to high-dim pixel observations? To answer these questions, we learn skills from demonstrations and use these skills to learn new tasks, first on a 2D multi-task domain, and later on a 3D variant.

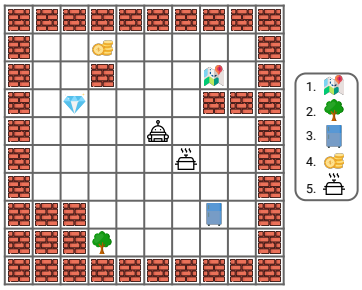

Figure 4: Multi-task grid world instance from Kipf et al. [39]. The agent must pick up a sequence of objects in a specified order.

Multi-task domain. We consider the multi-task 10×10 grid world introduced by Kipf et al. [39], a representative skill learning approach (Figure 4). This domain features a challenging control problem that requires the agent to collect up to 5 objects, and we later add a perception challenge to it with the 3D variant. We use our own custom implementation, since the code from Kipf et al. [39] is not publicly available. In each task, 6 objects are randomly sampled from a set of $N_{\text{obj}} = 10$ different objects and placed at random positions in the grid. Additionally, impassable walls are randomly added to the grid, and the agent is also placed at a random initial grid cell. Each task also includes an instruction list of $N_{\text{pick}} = 3$ or 5 objects that the agent must pick up in order.

Within each task, the agent's actions are to move up, down, left, right, or to pick up an item at the agent's grid cell. The state consists of two parts: (1) a $10 \times 10 \times (N_{\text{obj}} + 2)$ grid observation, indicating if the agent, a wall, or any of the N_{obj} different object types is present at each of the 10×10 grid cells; (2) the instruction list encoded as a list of ordered one-hot integers. Following Kipf et al. [39], we consider two reward variants: In the *sparse reward* variant, the agent only receives $+1$ reward after picking up all N_{pick} objects in the correct order. In the *dense reward* variant, the agent receives $+1$ reward after picking up each specified object in the correct order. Our dense reward variant is slightly harder than the variant in Kipf et al. [39], which gives the agent $+1$ reward for picking up objects in *any* order. The agent receives 0 reward in all other time steps. The episode ends when the agent has picked up all the objects in the correct order, or after 50 timesteps.

Pre-collected experience. We follow the setting in Kipf et al. [39]. We set the pre-collected experience to be 2000 demonstrations generated via breadth-first search on randomly generated tasks with only $N_{\text{pick}} = 3$ and test if the agent can generalize to $N_{\text{pick}} = 5$ when learning a new task. These demonstrations are not labeled with rewards and also *do not* contain the instruction list observations.

Points of comparison. To study our compression objective, we compare with two representatives of learning skills via the maximum likelihood objective:

- VTA [36], modified to handle interaction data as in Section 4.1.
- Discovery of deep options (DDO) [21]. Since the original implementation is unavailable, we implement DDO's maximum likelihood objective and graphical model on top of VTA's variational inference optimization procedure, instead of expectation-gradients used in the original paper.

For fairness, we also compare with variants of these that implement the minimum skill length constraint from Section 4.4. CompILE [39] is another approach that learns skill by maximizing likelihood and was introduced in the same paper as this grid world. Because the implementation is unavailable, we do not compare with it. However, we note that CompILE requires additional supervision that LOVE, VTA, and DDO do not: namely, it requires knowing how many skills each demonstration is composed of. This supervision can be challenging to obtain without already knowing what the skills should be.

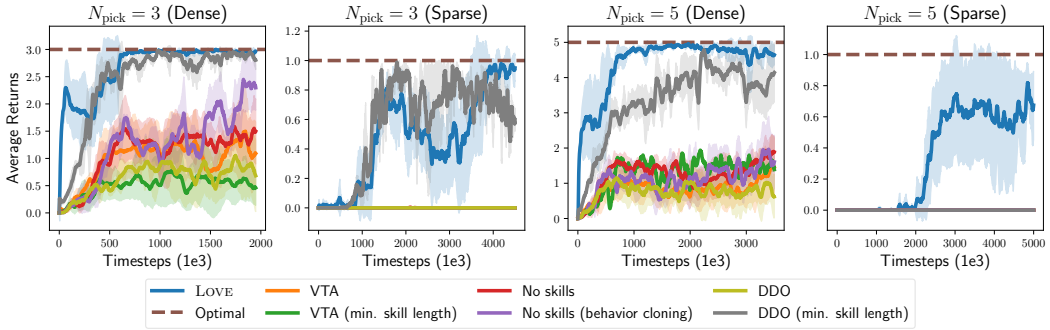

Figure 5: Returns vs. timesteps of environment interactions for 4 settings in the grid world with 1-stddev error bars (5 seeds). Only LOVE achieves high returns across all 4 settings.

Since latent variable models are prone to local optima [42], it is common to learn such models with multiple restarts [53]. We therefore run each method with 3 random initializations and pick the best model according to the compression objective $\mathcal{L}_{\text{CL}}$ for LOVE and according to the ELBO of the maximum likelihood objective $\mathcal{L}_{\text{ELBO}}$ for the others. Notably, this does not require any additional demonstrations or experience.

We begin by analyzing the skills learned from the pre-collected experience before analyzing the utility of these skills for learning downstream tasks in the next sections.

7.1 Analyzing the Learned Skills

We analyze the learned skills by comparing them to a natural decomposition that partitions the demonstration in N_{pick} sequences of moving to and picking up an object. Specifically, we measure the precision and recall of each method in outputting the correct boundaries of these N_{pick} sequences.

	Precision	Recall	F1
DDO [21]	0.19	**1**	0.32
DDO + min. skill length	0.26	0.53	0.35
VTA [36]	0.19	**0.99**	0.32
VTA + min. skill length	0.27	0.53	0.36
LOVE (ours)	**0.90**	0.94	**0.92**

Table 2: Precision and recall of outputting the boundaries between picking up different objects. Only LOVE learns skills that move to and pick up an object.

We find that only LOVE recovers the correct boundaries with both high precision and recall (Table 2). Visually examining the skills learned with LOVE shows that each skill moves to an object and picks it up. We visualize LOVE's skills in Appendix H. In contrast, maximizing likelihood with either DDO or VTA learns degenerate skills that output only a single action, leading to high recall, but low precision. Adding the minimum skill length constraint to these approaches prevents such degenerate solutions, but still does not lead to the correct boundaries. Skills learned by VTA with the minimum skill length constraint exhibit no apparent structure, but skills learned by DDO with the minimum skill length constraint appear to move toward objects, though they terminate at seemingly random places. These results further illustrate the underspecification problem. All approaches successfully learn to reconstruct the demonstrations and achieve high likelihood, but only LOVE and DDO with the minimum skill length constraint learn skills that appear to be semantically meaningful.

7.2 Learning New Tasks

Next, we evaluate the utility of the skills for learning new tasks. To evaluate the skills, we sample a new task in each of 4 settings: with sparse or dense rewards and with $N_{\text{pick}} = 3$ or $N_{\text{pick}} = 5$. Then we learn the new tasks following the procedure in Section 5: we augment the action space with the skills and train an agent over the augmented action space. To understand the impact of skills, we also compare with a low-level policy that learns these tasks using only the original action space, and the same low-level policy that incorporates the demonstrations by pre-training with behavioral cloning. We parametrize the policy for all approaches with dueling double deep Q-networks [52, 79, 77] with ϵ-greedy exploration. We report the returns of evaluation episodes with $\epsilon = 0$, which are run every 10 episodes. We report returns averaged over 5 seeds. See Appendix F.3 for full details.

Results. Overall, LOVE learns new tasks across all 4 settings comparably to or faster than both skill methods based on maximizing likelihood and incorporating demonstrations via behavior cloning (Figure 5). More specifically, when $N_{\text{pick}} = 3$, DDO with the min. skill length constraint performs comparably to LOVE, despite not segmenting the demonstrations into sequences of moving to and picking up objects. We observe that while a single DDO skill does not move to and pick up an

object, the skills consistently move toward objects, which likely helps with exploration. Imitating the demonstrations with behavior cloning also accelerates learning when $N_{\text{pick}} = 3$ with dense rewards, though not as much as LOVE or DDO, and yields little benefit in all other settings. Skill learning with VTA yields no benefit over no skill learning at all.

Recall that $N_{\text{pick}} = 3$ in all of the demonstrations. LOVE learns new tasks in the generalization setting of $N_{\text{pick}} = 5$ much faster than the other methods. With dense rewards, DDO (min. skill length) also eventually solves the task, but requires over $8\times$ more timesteps. The other approaches learn to pick up some objects, but never achieve optimal returns of 5. With sparse rewards, LOVE is the only approach to achieve high returns, while all other approaches achieve 0 returns. This likely occurs because this setting creates a challenging exploration problem, so exploring with low-level actions or skills learned with VTA or DDO rarely achieves reward. By contrast, exploring with skills learned with LOVE achieves rewards much more frequently, which enables LOVE to ultimately solve the task.

7.3 Scaling to High-dimensional Observations

Prior works in the offline multi-task setting have learned skills from low-dim states and then transferred them to high-dim pixel observations [63], or have learned hierarchical models from pixel observations [36, 21]. However, to the best of our knowledge, prior works do not directly learn skills from high-dim pixel observations, only from low-dim states. Hence, in this section, we test if LOVE can scale up to high-dim pixel observations by considering a challenging 3D image-based variant of the multi-task domain above, illustrated in Figure 6. The goal is still to pick up N_{pick} objects in the correct order, where objects are blocks of different colors. Each task includes 3 objects randomly sampled from a set of $N_{\text{obj}} = 6$ different colors. The observations are $400 \times 60 \times 3$ egocentric panoramic RGB arrays. The actions are to move forward / backward, turn left / right, and pick up.

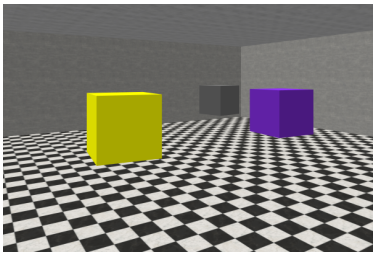

Figure 6: 3D visual multi-task domain.

We consider only the harder *sparse reward* and generalization setting, where the agent must pick up $N_{\text{pick}} = 3$ objects in the correct order, after receiving 2000 pre-collected demonstrations of the approximate shortest path of picking up $N_{\text{pick}} = 2$ objects in the correct order. We use the same hyperparameters as in multi-task domain and only change the observation encoder and action decoder (full details in Appendix D.4). Figure 7 shows the results. Again, LOVE learns skills that each navigate to and pick up an object, which quickly achieves optimal returns, indicating its ability to scale to high-dim observations. In contrast, the other approaches perform much worse on pixel observations than the previous low-dim observations. DDO also makes progress on the task, but requires far more samples, as it again learns skills that only operate for a few timesteps, though they move toward the objects. All other approaches fail to learn at all, including the variants of VTA and DDO without the min. skill length, which are not plotted.

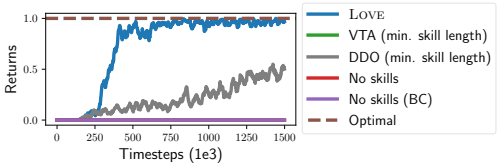

Figure 7: Returns on the 3D visual multi-task domain. Only LOVE achieves near-optimal returns.

8 Conclusion

We started by highlighting the underspecification problem: maximizing likelihood does not necessarily yield skills that are useful for learning new tasks. To overcome this problem, we drew a connection between skill learning and compression and proposed a new objective for skill learning via compression and a differentiable model for optimizing our objective. Empirically, we found that the underspecification problem occurs even on simple tasks and learning skills with our objective allows learning new tasks much faster than learning skills by maximizing likelihood.

Still, important future work remains. LOVE applies when there are useful structures that can be extracted from multiple trajectories. This is often present in multi-task demonstrations, which solve related tasks in similar ways. However, an open future challenge for adapting to general offline data is to ensure that the learned skills do not overfit to potentially noisy or unhelpful behaviors often present in offline data. In addition, We showed a connection between skill learning and compression by minimizing the description length of a crude two-part code. However, we only proposed a way to optimize the message length term and not the model length. Completing this connection by accounting for model length may therefore be an exciting direction for future work.

References

[1] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *arXiv preprint arXiv:1606.07419*, 2016.

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

[3] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, pages 166–175. PMLR, 2017.

[4] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[5] Akhil Bagaria, Jason K Senthil, and George Konidaris. Skill discovery for exploration and planning using deep skill graphs. In *International Conference on Machine Learning*, pages 521–531. PMLR, 2021.

[6] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1):41–77, 2003.

[7] Ritwik Bera, Vinicius G Goecks, Gregory M Gremillion, John Valasek, and Nicholas R Waytowich. Podnet: A neural network for discovery of plannable options. *arXiv preprint arXiv:1911.00171*, 2019.

[8] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[9] David M Blei and Pedro J Moreno. Topic segmentation with an aspect hidden markov model. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 343–348, 2001.

[10] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020.

[11] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[12] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[13] Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

[14] Hanjun Dai, Bo Dai, Yan-Ming Zhang, Shuang Li, and Le Song. Recurrent hidden semi-markov model. 2016.

[15] Sudeep Dasari, Frederik Ebert, Stephen Tian, Suraj Nair, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, Sergey Levine, and Chelsea Finn. Robonet: Large-scale multi-robot learning. *arXiv preprint arXiv:1910.11215*, 2019.

[16] Zhe Dong, Bryan Seybold, Kevin Murphy, and Hung Bui. Collapsed amortized variational inference for switching nonlinear dynamical systems. In *International Conference on Machine Learning*, pages 2638–2647. PMLR, 2020.

[17] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=SJx63jRqFm`.

[18] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. *Advances in neural information processing systems*, 29:64–72, 2016.

[19] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*, 2017.

[20] Emily B Fox, Erik B Sudderth, Michael I Jordan, and Alan S Willsky. A sticky hdp-hmm with application to speaker diarization. *The Annals of Applied Statistics*, pages 1020–1056, 2011.

[21] Roy Fox, Sanjay Krishnan, Ion Stoica, and Ken Goldberg. Multi-level discovery of deep options. *arXiv preprint arXiv:1703.08294*, 2017.

[22] Zoubin Ghahramani and Geoffrey E Hinton. Variational learning for switching state-space models. *Neural computation*, 12(4):831–864, 2000.

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

[24] Peter Grunwald. A tutorial introduction to the minimum description length principle. *arXiv preprint math/0406077*, 2004.

[25] James Harrison, Apoorva Sharma, Chelsea Finn, and Marco Pavone. Continuous meta-learning without tasks. *arXiv preprint arXiv:1912.08866*, 2019.

[26] Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rk07ZXZRb`.

[27] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[28] Geoffrey E Hinton and Richard S Zemel. Autoencoders, minimum description length, and helmholtz free energy. *Advances in neural information processing systems*, 6:3–10, 1994.

[29] Antti Honkela and Harri Valpola. Variational learning and bits-back coding: an information-theoretic view to bayesian learning. *IEEE transactions on Neural Networks*, 15(4):800–810, 2004.

[30] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[31] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[32] Yiding Jiang, Shixiang Gu, Kevin Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning. *arXiv preprint arXiv:1906.07343*, 2019.

[33] Yiding Jiang*, Behnam Neyshabur*, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=SJgIPJBFvH`.

[34] Matthew J Johnson, David K Duvenaud, Alex Wiltschko, Ryan P Adams, and Sandeep R Datta. Composing graphical models with neural networks for structured representations and fast inference. *Advances in neural information processing systems*, 29:2946–2954, 2016.

[35] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. *arXiv preprint arXiv:2005.05951*, 2020.

[36] Taesup Kim, Sungjin Ahn, and Yoshua Bengio. Variational temporal abstraction. *Advances in Neural Information Processing Systems*, 32:11570–11579, 2019.

[37] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[38] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[39] Thomas Kipf, Yujia Li, Hanjun Dai, Vinicius Zambaldi, Alvaro Sanchez-Gonzalez, Edward Grefenstette, Pushmeet Kohli, and Peter Battaglia. Compile: Compositional imitation learning and execution. In *International Conference on Machine Learning*, pages 3418–3428. PMLR, 2019.

[40] Sanjay Krishnan, Roy Fox, Ion Stoica, and Ken Goldberg. Ddco: Discovery of deep continuous options for robot learning from demonstrations. In *Conference on robot learning*, pages 418–437. PMLR, 2017.

[41] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29:3675–3683, 2016.

[42] M Pawan Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In *NIPS*, volume 1, page 2, 2010.

[43] Sang-Hyun Lee and Seung-Woo Seo. Learning compound tasks without task-specific knowledge via imitation and self-supervised learning. In *International Conference on Machine Learning*, pages 5747–5756. PMLR, 2020.

[44] Youngwoon Lee, Shao-Hua Sun, Sriram Somasundaram, Edward S Hu, and Joseph J Lim. Composing complex skills by learning transition policies. In *International Conference on Learning Representations*, 2018.

[45] Scott Linderman, Matthew Johnson, Andrew Miller, Ryan Adams, David Blei, and Liam Paninski. Bayesian learning and inference in recurrent switching linear dynamical systems. In *Artificial Intelligence and Statistics*, pages 914–922. PMLR, 2017.

[46] Scott W Linderman, Andrew C Miller, Ryan P Adams, David M Blei, Liam Paninski, and Matthew J Johnson. Recurrent switching linear dynamical systems. *arXiv preprint arXiv:1610.08466*, 2016.

[47] Evan Zheran Liu, Ramtin Keramati, Sudarshan Seshadri, Kelvin Guu, Panupong Pasupat, Emma Brunskill, and Percy Liang. Learning abstract models for strategic exploration and fast reward transfer. *arXiv preprint arXiv:2007.05896*, 2020.

[48] Siqi Liu, Guy Lever, Zhe Wang, Josh Merel, SM Eslami, Daniel Hennes, Wojciech M Czarnecki, Yuval Tassa, Shayegan Omidshafiei, Abbas Abdolmaleki, et al. From motor control to team play in simulated humanoid football. *arXiv preprint arXiv:2105.12196*, 2021.

[49] Yuchen Lu, Yikang Shen, Siyuan Zhou, Aaron Courville, Joshua B Tenenbaum, and Chuang Gan. Learning task decomposition with ordered memory policy network. *arXiv preprint arXiv:2103.10972*, 2021.

[50] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[51] Amy McGovern and Richard S Sutton. Macro-actions in reinforcement learning: An empirical analysis. *Computer Science Department Faculty Publication Series*, page 15, 1998.

[52] 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.

[53] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[54] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *arXiv preprint arXiv:1805.08296*, 2018.

[55] Taewook Nam, Shao-Hua Sun, Karl Pertsch, Sung Ju Hwang, and Joseph J Lim. Skill-based meta-reinforcement learning. In *Fifth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=jsV1-AQVEFY`.

[56] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. *arXiv preprint arXiv:1412.6614*, 2014.

[57] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro. Exploring generalization in deep learning. *arXiv preprint arXiv:1706.08947*, 2017.

[58] Scott Niekum, Sachin Chitta, Andrew G Barto, Bhaskara Marthi, and Sarah Osentoski. Incremental semantically grounded learning from demonstration. In *Robotics: Science and Systems*, volume 9, pages 10–15607. Berlin, Germany, 2013.

[59] Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *International Conference on Machine Learning*, pages 2661–2670. PMLR, 2017.

[60] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[61] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. *arXiv preprint arXiv:1711.00937*, 2017.

[62] Karl Pertsch, Youngwoon Lee, and Joseph J Lim. Accelerating reinforcement learning with learned skill priors. *arXiv preprint arXiv:2010.11944*, 2020.

[63] Dushyant Rao, Fereshteh Sadeghi, Leonard Hasenclever, Markus Wulfmeier, Martina Zambelli, Giulia Vezzani, Dhruva Tirumala, Yusuf Aytar, Josh Merel, Nicolas Heess, et al. Learning transferable motor skills with hierarchical latent mixture policies. *arXiv preprint arXiv:2112.05062*, 2021.

[64] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[65] Melrose Roderick, Christopher Grimm, and Stefanie Tellex. Deep abstract q-networks. *arXiv preprint arXiv:1710.00459*, 2017.

[66] Tanmay Shankar and Abhinav Gupta. Learning robot skills with temporal variational inference. In *International Conference on Machine Learning*, pages 8624–8633. PMLR, 2020.

[67] Tanmay Shankar, Shubham Tulsiani, Lerrel Pinto, and Abhinav Gupta. Discovering motor programs by recomposing demonstrations. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rkgHY0NYwr`.

[68] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=HJgLZR4KvH`.

[69] Arjun Sharma, Mohit Sharma, Nicholas Rhinehart, and Kris M Kitani. Directed-info gail: Learning hierarchical policies from unsegmented demonstrations using directed information. *arXiv preprint arXiv:1810.01266*, 2018.

[70] Kyriacos Shiarlis, Markus Wulfmeier, Sasha Salter, Shimon Whiteson, and Ingmar Posner. Taco: Learning task decomposition via temporal alignment for control. In *International Conference on Machine Learning*, pages 4654–4663. PMLR, 2018.

[71] Yuri M Shtarkov. Univresal sequential coding of single messages. *Prob. Pered. Inf.*, 23:175–186, 1987.

[72] Avi Singh, Huihan Liu, Gaoyue Zhou, Albert Yu, Nicholas Rhinehart, and Sergey Levine. Parrot: Data-driven behavioral priors for reinforcement learning. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=Ysuv-WOFeKR`.

[73] Sumedh A Sontakke, Sumegh Roychowdhury, Mausoom Sarkar, Nikaash Puri, Balaji Krishnamurthy, and Laurent Itti. Video2skill: Adapting events in demonstration videos to skills in an environment using cyclic mdp homomorphisms. *arXiv preprint arXiv:2109.03813*, 2021.

[74] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

[75] Daniel Tanneberg, Kai Ploeger, Elmar Rueckert, and Jan Peters. Skid raw: Skill discovery from raw trajectories. *IEEE Robotics and Automation Letters*, 6(3):4696–4703, 2021.

[76] Guillermo Valle-Perez, Chico Q Camargo, and Ard A Louis. Deep learning generalizes because the parameter-function map is biased towards simple functions. *arXiv preprint arXiv:1805.08522*, 2018.

[77] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[78] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 3540–3549. PMLR, 2017.

[79] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[80] Mengjiao Yang and Ofir Nachum. Representation matters: Offline pretraining for sequential decision making. *arXiv preprint arXiv:2102.05815*, 2021.

[81] Mengjiao Yang, Sergey Levine, and Ofir Nachum. Trail: Near-optimal imitation learning with suboptimal data. *arXiv preprint arXiv:2110.14770*, 2021.

[82] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3): 107–115, 2021.

[83] Jesse Zhang, Karl Pertsch, Jiefan Yang, and Joseph J Lim. Minimum description length skills for accelerated reinforcement learning. In *Self-Supervision for Reinforcement Learning Workshop - ICLR 2021*, 2021. URL `https://openreview.net/forum?id=r4XxtrIo1m9`.

[84] Zelin Zhao, Chuang Gan, Jiajun Wu, Xiaoxiao Guo, and Joshua B Tenenbaum. Augmenting policy learning with routines discovered from a demonstration. *arXiv preprint arXiv:2012.12469*, 2020.

[85] Wenxuan Zhou, Sujay Bajracharya, and David Held. Plas: Latent action space for offline reinforcement learning. *arXiv preprint arXiv:2011.07213*, 2020.

[86] Yifeng Zhu, Peter Stone, and Yuke Zhu. Bottom-up skill discovery from unsegmented demonstrations for long-horizon robot manipulation. *arXiv preprint arXiv:2109.13841*, 2021.

A Derivations

A.1 $p^\star_{\boldsymbol{z}}$ is a valid density

Proposition A.1. *$p^\star_{\boldsymbol{z}}(\xi)$ is a proper probability density function.*

Proof. Recall that

$$p^\star_{\boldsymbol{z}}(\xi) = \frac{\mathbb{E}_{\substack{\boldsymbol{x}_{1:T},\boldsymbol{m}_{1:T},\\ \boldsymbol{z}_{1:T}}}\left[\sum_{t=1}^T \delta(\boldsymbol{z}_t = \xi)\boldsymbol{m}_t\right]}{\mathbb{E}_{\boldsymbol{x}_{1:T},\boldsymbol{m}_{1:T}}\left[\sum_{t=1}^T \boldsymbol{m}_t\right]}.$$

A function is a probability density function if it is non-negative, continuous, and integrates to 1. By construction, $p^\star_{\boldsymbol{z}}(\xi)$ is non-negative since $\boldsymbol{m}_t \geq 0$ and $\delta(\boldsymbol{z}_t = \xi) \geq 0$. Further, $\delta(\boldsymbol{z}_t = \xi)$ is continuous in ξ so the numerator $\mathbb{E}_{\substack{\boldsymbol{x}_{1:T},\boldsymbol{m}_{1:T},\\ \boldsymbol{z}_{1:T}}}\left[\sum_{t=1}^T \delta(\boldsymbol{z}_t = \xi)\boldsymbol{m}_t\right]$, an expectation containing $\delta(\boldsymbol{z}_t = \xi)$, is also continuous in ξ. Finally,

$$\begin{aligned}
\int_\xi p^\star_{\boldsymbol{z}}(\xi)\, d\xi &= \frac{1}{\mathbb{E}_{\boldsymbol{x}_{1:T},\boldsymbol{m}_{1:T}}\left[\sum_{t=1}^T \boldsymbol{m}_t\right]} \int_\xi \mathbb{E}_{\boldsymbol{x}_{1:T},\boldsymbol{m}_{1:T},\boldsymbol{z}_{1:T}}\left[\sum_{t=1}^T \delta(\boldsymbol{z}_t = \xi)\boldsymbol{m}_t\right] d\xi \\
&= \frac{1}{\mathbb{E}_{\boldsymbol{x}_{1:T},\boldsymbol{m}_{1:T}}\left[\sum_{t=1}^T \boldsymbol{m}_t\right]} \mathbb{E}_{\boldsymbol{x}_{1:T},\boldsymbol{m}_{1:T},\boldsymbol{z}_{1:T}}\left[\sum_{t=1}^T \left(\int_\xi \delta(\boldsymbol{z}_t = \xi) d\xi\right)\boldsymbol{m}_t\right] \quad \text{(Fubini's theorem)} \\
&= \frac{1}{\mathbb{E}_{\boldsymbol{x}_{1:T},\boldsymbol{m}_{1:T}}\left[\sum_{t=1}^T \boldsymbol{m}_t\right]} \mathbb{E}_{\boldsymbol{x}_{1:T},\boldsymbol{m}_{1:T},\boldsymbol{z}_{1:T}}\left[\sum_{t=1}^T 1 \cdot \boldsymbol{m}_t\right] \\
&= 1.
\end{aligned}$$

Therefore, $p^\star_{\boldsymbol{z}}(\xi)$ satisfies all 3 properties of a probability density function. □

Remarks. This result generalizes to the case for discrete $\boldsymbol{z}$ by replacing the delta with an indicator function and integration with summation.

A.2 Two expressions of $\mathcal{L}_{\text{CL}}$

Proposition A.2. *The expected coding length of a trajectory is equal to the expected number skills multiplied by the marginal entropy:*

$$\mathcal{L}_{CL}(\boldsymbol{\theta}) = -\mathbb{E}_{\substack{\boldsymbol{x}_{1:T},\boldsymbol{m}_{1:T},\\ \boldsymbol{z}_{1:T}}}\left[\sum_{t=1}^T \log p^\star_{\boldsymbol{z}}(\boldsymbol{z}_t)\boldsymbol{m}_t\right] = n_s \mathcal{H}_{p^\star_{\boldsymbol{z}}}[\boldsymbol{z}].$$

Proof.

$$\begin{aligned}
&-\mathbb{E}_{\boldsymbol{x}_{1:T},\boldsymbol{m}_{1:T},\boldsymbol{z}_{1:T}}\left[\sum_{t=1}^T \log p^\star_{\boldsymbol{z}}(\boldsymbol{z}_t)\boldsymbol{m}_t\right] \\
&= -\mathbb{E}_{\boldsymbol{x}_{1:T},\boldsymbol{m}_{1:T},\boldsymbol{z}_{1:T}}\left[\sum_{t=1}^T \left(\int_\xi \delta(\boldsymbol{z}_t = \xi) \log p^\star_{\boldsymbol{z}}(\xi) d\xi\right)\boldsymbol{m}_t\right] \\
&= -\mathbb{E}_{\boldsymbol{x}_{1:T},\boldsymbol{m}_{1:T},\boldsymbol{z}_{1:T}}\left[\int_\xi \sum_{t=1}^T \boldsymbol{m}_t \delta(\boldsymbol{z}_t = \xi) \log p^\star_{\boldsymbol{z}}(\xi) d\xi\right] && \text{(Fubini's theorem)} \\
&= -\int_\xi \mathbb{E}_{\boldsymbol{x}_{1:T},\boldsymbol{m}_{1:T},\boldsymbol{z}_{1:T}}\left[\sum_{t=1}^T \boldsymbol{m}_t \delta(\boldsymbol{z}_t = \xi)\right] \log p^\star_{\boldsymbol{z}}(\xi) d\xi && \text{(Linearity of expectation)} \\
&= -\mathbb{E}_{\boldsymbol{x}_{1:T},\boldsymbol{m}_{1:T}}\left[\sum_{t=1}^T \boldsymbol{m}_t\right] \int_\xi \frac{\mathbb{E}_{\boldsymbol{x}_{1:T},\boldsymbol{m}_{1:T},\boldsymbol{z}_{1:T}}\left[\sum_{t=1}^T \boldsymbol{m}_t \delta(\boldsymbol{z}_t = \xi)\right]}{\mathbb{E}_{\boldsymbol{x}_{1:T},\boldsymbol{m}_{1:T}}\left[\sum_{t=1}^T \boldsymbol{m}_t\right]} \log p^\star_{\boldsymbol{z}}(\xi) d\xi
\end{aligned}$$

$$= \underbrace{\mathbb{E}_{\boldsymbol{x}_{1:T}, \boldsymbol{m}_{1:T}, \boldsymbol{z}_{1:T}} \left[\sum_{t=1}^{T} \boldsymbol{m}_t \right]}_{n_{\mathrm{s}}} \underbrace{\int_{\xi} -p^{\star}_{\boldsymbol{z}}(\xi) \log p^{\star}_{\boldsymbol{z}}(\xi) d\xi}_{\mathcal{H}_{p^{\star}_{\boldsymbol{z}}}[\boldsymbol{z}]}$$

$$= n_{\mathrm{s}} \mathcal{H}_{p^{\star}_{\boldsymbol{z}}}[\boldsymbol{z}]$$

□

Remarks. This result generalizes to the case for discrete $\boldsymbol{z}$ by replacing the delta with an indicator function and integration with summation. Further, this rewriting not only offers an insightful interpretation of the objective but also has *different* finite-sample gradient compared to the original form. We discuss its implication further in Section 4.4.

A.3 But you just picked the wrong prior

Is the compression objective a prior? Yes and no. A bridge that connects variational inference and source coding is the bits-back coding argument [28, 29]. Under this scheme, the cost of communicating a message is equal to the KL divergence between the approximate posterior and the chosen prior distribution. This is because the posterior often does not match the prior and the sender must use more bits to send the message compared to using the prior exactly. In modern deep latent variable models such as VAE [38], the prior is in general chosen to be an isotropic Gaussian distribution or a sequence of isotropic Gaussian distributions that decomposes over the time steps. The bit-back coding argument shows that variational inference as a coding scheme is asymptotically optimal (in this limit of sending large numbers of messages), but it does not immediately guarantee good *representation learning*. As a thought experiments, imagine we choose the dimension of the latent variable in a VAE to be equal to the dimension of the data itself. It is clear that the model is not incentivized to perform good representation learning, but the bit-backing coding scheme still applies. Instead, good representation can only emerge *if* the prior is chosen properly.

Contrary to this prevailing design choice, our latent variables $\boldsymbol{z}_{1:T}$ and $\boldsymbol{m}_{1:T}$ are not independent from each other and the timesteps are not independent from each other. In fact, the interaction between the latent variables is crucial for compressing the trajectory optimally in our framework. Therefore, our objective takes into account the *global* structure of the latent variables. Note that $\mathcal{L}_{\mathrm{CL}}$ also has the natural interpretation of measuring the average number of bits that is required to send a trajectory under the coding scheme of this work. As both quantities measure the cost of the communication, the philosophical parallel between KL divergence and $\mathcal{L}_{\mathrm{CL}}$ is likely not a coincidence. In practice, we observe that $\mathcal{L}_{\mathrm{CL}}$ also has comparable regularization effect on the latent code, and in some cases it is possible to learn a good model *without* any KL divergence if $\mathcal{L}_{\mathrm{CL}}$ is present. From this perspective, our compression objective is indeed a "prior" insofar as it specifies the desired properties of the posterior.

Nonetheless, it is an open question whether $\mathcal{L}_{\mathrm{CL}}$ has a precise counterpart in Bayesian inference, that is, it is not immediately obvious if it can be written as the KL divergence between the posterior $q(\boldsymbol{z}_{1:T}, \boldsymbol{m}_{1:T} \mid \boldsymbol{x}_{1:T})$ and some prior distribution $p(\boldsymbol{z}_{1:T}, \boldsymbol{m}_{1:T})$. One complication, for example, is the fact that the log density of $\boldsymbol{m}_{1:T}$ does not participate in the computation of $\mathcal{L}_{\mathrm{CL}}$. It may be possible to construct some form of hierarchical priors that enables a fully Bayesian interpretation of $\mathcal{L}_{\mathrm{CL}}$, but it is also well-known that the MDL framework can accommodate codes that are not Bayesian (e.g., the Shtarkov normalized maximum likelihood code [71]). We do not offer a definitive answer to this question in this work, but the connection between Bayesian inference and $\mathcal{L}_{\mathrm{CL}}$ is certainly an interesting direction for future works.

B Evidence Lower Bound

B.1 VTA

We reproduce the data generating process of Kim et al. [36] here:

$$p(\boldsymbol{x}_{1:T}, \boldsymbol{z}_{1:T}, \boldsymbol{s}_{1:T}, \boldsymbol{m}_{1:T}) = \prod_{t=1}^{T} p(\boldsymbol{x}_t \mid \boldsymbol{s}_t) p(m_t \mid \boldsymbol{s}_t) p(\boldsymbol{s}_t \mid \boldsymbol{s}_{<t}, \boldsymbol{z}_t, m_{t-1}) p(\boldsymbol{z}_t \mid \boldsymbol{z}_{<t}, m_{t-1}).$$

VTA [36] shows that the ELBO can be written as:

$$\mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\phi}) \approx \sum_{t=1}^{T} \log p_{\boldsymbol{\phi}}(\boldsymbol{x}_t \mid \boldsymbol{s}_t) + D_{\text{KL}}\left(q_{\boldsymbol{\theta}}(m_t \mid \boldsymbol{x}_{1:T}) \,\|\, p_{\boldsymbol{\phi}}(m_t \mid \boldsymbol{s}_t)\right)$$
$$+ D_{\text{KL}}\left(q_{\boldsymbol{\theta}}(\boldsymbol{z}_t \mid \boldsymbol{s}_t, \boldsymbol{m}_{1:T}, \boldsymbol{x}_{1:T}) \,\|\, p_{\boldsymbol{\phi}}(\boldsymbol{z}_t \mid \boldsymbol{s}_{t-1}, m_{t-1})\right)$$
$$+ D_{\text{KL}}\left(q_{\boldsymbol{\theta}}(\boldsymbol{s}_t \mid \boldsymbol{s}_{t-1}, \boldsymbol{m}_{1:T}, \boldsymbol{x}_{1:T}) \,\|\, p_{\boldsymbol{\phi}}(\boldsymbol{s}_t \mid \boldsymbol{s}_{t-1}, \boldsymbol{z}_t, m_{t-1})\right).$$

Like most ELBO, this ELBO can be deconstructed into a reconstruction term:

$$\mathcal{L}_{\text{rec}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_{t=1}^{T} \log p_{\boldsymbol{\phi}}(\boldsymbol{x}_t \mid \boldsymbol{s}_t),$$

and the remaining KL term $\mathcal{L}_{\text{KL}}(\boldsymbol{\theta}, \boldsymbol{\phi})$.

B.2 LOVE

We reproduce the data generating process of LOVE here. Instead of using $\boldsymbol{\phi}$ for the parameters of priors and decoder, we will merge all parameters into $\boldsymbol{\theta}$:

$$p(\boldsymbol{a}_{1:T}, \boldsymbol{z}_{1:T}, \boldsymbol{s}_{1:T}, \boldsymbol{m}_{1:T} \mid \boldsymbol{x}_{1:T}) = \prod_{t=1}^{T} p(\boldsymbol{a}_t \mid \boldsymbol{s}_t) p(m_t \mid \boldsymbol{s}_t) p(\boldsymbol{s}_t \mid \boldsymbol{x}_t, \boldsymbol{z}_t, m_{t-1}) p(\boldsymbol{z}_t \mid \boldsymbol{x}_t, \boldsymbol{z}_{<t}, m_{t-1}).$$

For LOVE, the ELBO is re-written as:

$$\mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}) \approx \sum_{t=1}^{T} \log p_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t) + D_{\text{KL}}\left(q_{\boldsymbol{\theta}}(m_t \mid \boldsymbol{x}_{1:t}) \,\|\, p_{\boldsymbol{\theta}}(m_t \mid \boldsymbol{s}_t)\right) + D_{\text{KL}}\left(q_{\boldsymbol{\theta}}(z_t \mid m_t, \boldsymbol{x}_{1:T}, \boldsymbol{a}_{1:T}) \,\|\, p(z_t)\right)$$
$$+ D_{\text{KL}}\left(q_{\boldsymbol{\theta}}(\boldsymbol{s}_t \mid z_t, \boldsymbol{x}_t) \,\|\, p_{\boldsymbol{\theta}}(\boldsymbol{s}_t \mid \boldsymbol{s}_{t-1}, m_{t-1})\right). \tag{4}$$

Since z_t is a categorical distribution, we chose the uniform prior $p(z_t)$ similar to Oord et al. [61]. Similar to VTA, the loss is also decomposed into a reconstruction term and a KL term.

C Sensitivity of λ

Here we study how sensitive LOVE is to the choice of λ, the coefficient of $\mathcal{L}_{\text{CL}}$. We tested 5 values of $\lambda \in [0.01, 1]$ on *Simple* and *Cond. Colors* and applied dual gradient descent to automatically tune λ.

	$\lambda = 0.01$	$\lambda = 0.03$	$\lambda = 0.1$	$\lambda = 0.3$	$\lambda = 1.0$	λ (dual GD)	VTA
Simple (F1)	$.67 \pm .27$	$.80 \pm .21$	$.91 \pm .02$	$.95 \pm .06$	$.74 \pm .08$	$.99 \pm .00$	$.82 \pm .13$
Cond. (F1)	$.84 \pm .06$	$.92 \pm .02$	$.90 \pm .03$	$.82 \pm .08$	$.87 \pm .08$	$.99 \pm .00$	$.83 \pm .19$

Table 3: The effect of different values of λ and dual GD on F1 scores of *Simple Colors* and *Conditional Colors* over 5 seeds.

We see that while λ can change the quality of solutions found, the performance is stable within a $\pm 3\times$ regions around $\lambda = 0.1$. Further, using dual GD *consistently* outperforms handpicked λ with an ELBO threshold C slightly (e.g., 5%) higher than the $\mathcal{L}_{\text{ELBO}}$ achieved without compression, which can be obtained with minimal prior knowledge about the task. Hence λ likely can be tuned without strong prior knowledge. This contrasts with the other methods such as Kipf et al. [39] which require knowing the number of skills in each demonstration to perform well while achieving the similar or better performance.

D Architecture

D.1 Frame Prediction

Our base architecture is similar to Kim et al. [36][1] and we highlight the difference blue. The parameters of the posteriors are implemented with neural networks. First, the observation $\boldsymbol{x}_{1:T}$ are encoded into a lower dimension embedding $\hat{\boldsymbol{x}}_{1:T}$ with a `observation encoder`. The embedding is

[1] `https://github.com/taesupkim/vta`

then passed through a `boundary posterior decoder` that outputs the parameters of $\boldsymbol{m}_{1:T}$. The embedding is further passed through a GRU [11], $f_{\text{s-rnn-fwd}}$ which runs on $\hat{\boldsymbol{x}}_{1:T}$ and give the cell state $\mathtt{h}_{1:T}$. For $\boldsymbol{z}_{1:T}$, we have two GRU's with cell, $f_{\text{z-rnn-bwd}}$ and $f_{\text{z-rnn-bwd}}$, which run on $\hat{\boldsymbol{x}}_{1:T}$ in different temporal direction (i.e., $f_{\text{z-rnn-bwd}}$ sees the future) to generate cell states $\mathtt{c}^{\text{fwd}}_{1:T}$ and $\mathtt{c}^{\text{bwd}}_{1:T}$. For each time step t, a candidate $\boldsymbol{z}'_t$ is sampled from a n_z-dimensional categorical distribution[2] whose log probability is a linear projection of $\mathtt{c}^{\text{fwd}}_t \| \mathtt{c}^{\text{bwd}}_t$. The sampled $\boldsymbol{z}'_t$ are embedded linearly into a vector of size 128. If $\boldsymbol{m}_1 = 1$, the $\boldsymbol{z}_t = \boldsymbol{z}'_t$; otherwise, $\boldsymbol{z}_t = \boldsymbol{z}_{t-1}$ [3]. Then we concatenate $\mathtt{h}_{t-1} \| \boldsymbol{z}_t$ and linearly project the vector into the mean and standard deviation of a isotropic Gaussian in $\mathbb{R}^8$. A $\hat{\boldsymbol{s}}_t$ is sampled from this Gaussian.

Th model also keeps two "belief" vectors that keep track of the history of the computation. These vectors are modulated by two RNN, $f_{\text{h-rnn}}$ and $f_{\text{c-rnn}}$. The abstraction belief $\boldsymbol{c}_t = \boldsymbol{m}_t \cdot f_{\text{c-rnn}}(\boldsymbol{z}_t, \boldsymbol{c}_{t-1}) + (1 - \boldsymbol{m}_t) \cdot \boldsymbol{c}_{t-1}$. The observation belief $\boldsymbol{h}_t = \boldsymbol{m}_t \boldsymbol{c}_{t-1} + (1 - \boldsymbol{m}_t) \cdot f_{\text{h-rnn}}(\hat{\boldsymbol{s}}_t \| \boldsymbol{c}_t \| \boldsymbol{z}_t, \boldsymbol{h}_{t-1})$. Finally the state abstraction is computed as the linear projection of $\boldsymbol{s}_t = \text{proj}(\boldsymbol{h}_t \| \hat{\boldsymbol{s}}_t)$. The projection layer is 256×128. Finally, $\boldsymbol{s}_t$ is passed through a `observation decoder` and decode into the reconstruction of $\boldsymbol{x}_t$. Refer to Kim et al. [36] for more details and an illustration of the computation graph.

For each convolutional layer or transposed convolution layer, the tuple's values correspond to input channel, output channel, kernel size, stride, padding. With that notation in mind, the specific hyperparameters for each components are:

- `observation encoder`: 4-layer convolutional neural network with {(3, 128, 4, 2, 1), (128, 128, 4, 2, 1), (128, 128, 4, 2, 1), (128, 128, 4, 1, 0)} with ELU activation [12] and batch normalization [30]. The output is flattened linearly projected to vector of size 128.
- `boundary posterior decoder` is a 5-layer 1-D causal temporal convolution [60] with {(128, 128, 5, 1, 2) $\times$ 5, (128, 2, 5, 1, 2)} with ELU activation and batch normalization. The output is the logits for the $\boldsymbol{m}_{1:T}$.
- $f_{\text{z-rnn-bwd}}, f_{\text{z-rnn-fwd}}, f_{\text{s-rnn-fwd}}, f_{\text{c-rnn}}, f_{\text{h-rnn}}$: GRU with cell size 128
- `observation decoder`: 4-layer transposed convolutional layers {(128, 128, 4, 1, 0), (128, 128, 4, 2, 1), (128, 128, 4, 2, 1), (128, 3, 4, 2, 1)} with ELU activation and batch normalization on all but the last layer. The last layer has no normalization and has `tanh` activation.

Removing Boundary Regularization. An important difference from Kim et al. [36] is that we do not enforce maximum number of subsequence N_{max} and the maximum length of subsequence l_{max} through the boundary prior (This effectively amounts to setting both to a value larger than the sequence length). These two hyperparameters' effect are similar to that of picking the number of segments in CompILE [39] and provide strong training signal. This assumption is in general problematic since we cannot expect to know a priori what the good values are for these hyperparameters. Kipf et al. [39] demonstrates that the performance can suffer if this kind of supervision is wrong. On the other hand, we do not assume anything about the the duration of the subsequence or how many subsequences there are in each demonstration.

We do, however, enforce a minimum length on the skill. While it is largely an optimization choice, we believe this prior is significantly weaker since useful options are almost always temporally extended. Indeed, with only the minimum length constraint, VTA is unable to learn good skills conducive for downstream tasks.

D.2 Interaction Data

For trajectories, we make some larger changes to the architecture to encode properties we want in a model for learning options, e.g., Markov property. Though similar in spirit, a large portion of the architecture is different from the base VTA model, so the difference will not be highlighted.

For the posterior, we have an `observation encoder` and an `action encoder` that embeds both the state observation $\boldsymbol{x}_{1:T}$ and actions $\boldsymbol{a}_{1:T}$ to get embedding $\mathtt{a}_{1:T}$ and $\mathtt{x}_{1:T}$ each of size 128. The concatenation $\mathtt{a}_t \| \mathtt{x}_t$ linear projected down to a vector of size 128. The embedding is then passed through a `boundary posterior decoder` that outputs the parameters of $\boldsymbol{m}_{1:T}$. The embedding is further passed through a GRU [11], $f_{\text{s-rnn-fwd}}$ which runs on $\hat{\boldsymbol{x}}_{1:T}$ and give the cell state $\mathtt{h}_{1:T}$. For

[2] Kim et al. [36] uses a isotropic Gaussian.

[3] This is the COPY action in Kim et al. [36].

$\boldsymbol{z}_{1:T}$, we have two GRU's with cell, $f_{\text{z-rnn-bwd}}$ and $f_{\text{z-rnn-bwd}}$, which run on $\hat{\boldsymbol{x}}_{1:T}$ in different temporal direction (i.e., $f_{\text{z-rnn-bwd}}$ sees the future) to generate cell states $\mathtt{c}^{\text{fwd}}_{1:T}$ and $\mathtt{c}^{\text{bwd}}_{1:T}$.

For sampling $\boldsymbol{z}$ is done differently for interaction data for better gradient and representation learning. Instead of doing straight-through estimator for the categorical random variable, we opt to use vector-quantization [61] with straight-through estimators. First, $\mathtt{c}^{\text{fwd}}_t \| \mathtt{c}^{\text{bwd}}_t$ is linearly projected into dimension 128. Then a ReLU is applied and another 128 by 128 linear layer is applied. The VQ codebook is of size $n_{\boldsymbol{z}} \times 128$. Instead of taking argmax of the codebook, we approximate the distribution to be proportional to the softmax over the distance over a temperature t_{VQ}. We can sample from this distribution and apply the straight-through gradient on the sampled code candidate $\boldsymbol{z}'_{1:T}$. If $\boldsymbol{m}_1 = 1$, the $\boldsymbol{z}_t = \boldsymbol{z}'_t$; otherwise, $\boldsymbol{z}_t = \boldsymbol{z}_{t-1}$.

We construct directly $\boldsymbol{s}_t = \boldsymbol{z}_t \| \mathtt{x}_t$. and linearly project the vector into the mean and standard deviation of a isotropic Gaussian $\mathbb{R}^8$. $\boldsymbol{s}_t$ is sampled from this Gaussian and then decoded with the `action decoder` into the reconstructed action in $\mathcal{A}$.

For each convolutional layer or transposed convolution layer, the tuple's values correspond to input channel, output channel, kernel size, stride, padding. With that notation in mind, the specific hyperparameters for each components are:

- `observation encoder`: This takes as input the $10 \times 10 \times N_{\text{obj}} + 2$ and is implemented as a two 2D convolutional layers with ReLU activations, followed by a linear layer with a ReLU activation and a linear layer with no activation. The convolutional layers have hyperparameters $(12, 64, 3, 1, 0)$ and $(64, 64, 3, 1, 0)$ respectively. The first linear has input dimension $6 \times 6 \times 64$ and output dimension 128. The second linear layer has input and output dimensions 128.
- `action encoder`: The actions are embedded with an embedding matrix with embedding dimension 128.
- `boundary posterior decoder` is a 5-layer 1-D causal temporal convolution [60] with {(128, 128, 5, 1, 2) $\times$ 5, (128, 2, 5, 1, 2)} with ELU activation and batch normalization. The output is the logits for the $\boldsymbol{m}_{1:T}$.
- $f_{\text{z-rnn-bwd}}, f_{\text{z-rnn-fwd}}, f_{\text{s-rnn-fwd}}$: GRU with cell size 128
- `action decoder`: The decoder is implemented as three linear layers with ReLU activations, followed by a linear layer with no activation. The (input, output) dimensions of these layers are as follows: $(128, 128), (128, 128), (128, 128), (128, 5)$, where the last layer outputs logits over the actions.

D.3 3D Navigation Environment

The architecture we use here is almost identical to that of multi-task environment D.2, but we change the output of the `action decoder` to be 4 and the `observation encoder` to be a 3-layer convolutional neural networks with $(3, 32, 5, 2, 0), (32, 32, 5, 2, 0), (32, 32, 4, 2, 0)$, followed two linear layers $(7520, 128), (128, 128)$ with a ReLU activation in between.

The same encoder is used for learning new tasks with HRL, but with an object embedding concatenated to the output (details in Appendix D.4).

D.4 Hierarchical Reinforcement Learning

For all approaches, we parametrize the policy as a double dueling deep Q-network [52, 79, 77]. The parametrization of the Q-function consists of a state embedding followed by two linear layer heads for the state-value $V_\theta(s)$ function and the advantage $A_\theta(s, a)$ function. Then the Q-function is computed as $Q_\theta(s, a) = V_\theta(s) + A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} A(s, a)$.

The state embedding is computed as embedding the grid state observation g as $e(g)$, embedding the instruction state observation i as $e(i)$, and applying a final linear layer with output dimension 128 to a ReLU on their concatenation: `Linear(ReLU([e(g); e(i)]))`. The grid embedding is computed using the same architecture as the observation encoder in the previous section. The instruction embedding is computed with a 16-dimensional embedding matrix.

E Hardware

All our experiments are conducted on a GeForce RTX 2080 Ti or a GeForce RTX A6000.

F Hyperparameters

F.1 Frame Prediction

For these set of experiments, we use the same architecture and hyperparameters as Kim et al. [36]. The hyperparameters used for both *Simple Colors* and *Conditional Colors* are:

- β : 1.0
- λ (fixed) : 0.1
- mini-batch size: 512
- training iteration: 30000
- n_z: 10

F.2 Interaction Data

For training the interaction data model, we use the following hyperparameters:

- β : 0
  - We found that the compression objective readily provides strong and sufficient regularization for the latent code. Adding additional KL divergence often results in the model not being able to reconstruct the actions properly.
- λ:
  - We use a adaptive scheduling that approximate dual gradient descent. λ is initialized to 0. After every gradient step, λ is increased by 2.0×10^{-5} if $\mathcal{L}_{\text{ELBO}} \leq 0.05$ (Since $\beta = 0$, this is effectively $\mathcal{L}_{\text{rec}}$); otherwise, λ is decreased by 2.0×10^{-5}. After each update, the value of λ is clipped to $[0, 0.05]$. We find this setting provides the most stable training.
- mini-batch size: 64
- training iteration: 20000
- n_z: 10
- t_{VQ}: 0.1

F.3 Hierarchical Reinforcement Learning

We use the same hyperparameters for training the policy for all approaches. Specifically, these hyperparameter values are as follows:

- ϵ-greedy schedule: We linearly decay ϵ from 1 to 0.01 over $500K$ timesteps for dense reward settings and over $5M$ timesteps for sparse reward settings.
- Discount factor γ: We use $\gamma = 0.99$ in all of the DQN updates.
- Maximum replay buffer size: 1M
- Minimum replay buffer size before updating: 500
- Learning rate: 0.0001 with the Adam optimizer [37]
- Batch size: 32
- Update frequency: every 4 timesteps
- Target syncing frequency: every 50K updates
- Gradient ℓ_2-norm clipping: 10
- Marginal threshold α: 0.001

Since the demonstrations do not contain the instruction list observations, we set those to 0 in the demonstrations for the behavior cloning approach.

G Algorithm

The Lagrangian for the unconstrained optimization problem is:

$$\min_{\boldsymbol{\theta}} \max_{\lambda \geq 0} \; \lambda \mathcal{L}_{\text{CL}}(\boldsymbol{\theta}) + (\mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}) - C)\,.$$

In standard KKT condition, the dual variable is introduced on the ELBO, but the two problems are equivalent by inverting the dual variable. Since the only constraint on λ is that it is non-negative, this transformation does not change the optimal solution. Further note that we find that in many cases, choosing a fixed constant value for λ is sufficient for solving the problem.

Algorithm 1 Learning Options via Compression (LOVE): We highlight differences between our method and prior work (VTA) in blue.

Initialize model parameters $\boldsymbol{\theta}$
while not converged **do**
  Sample a trajectory $(\boldsymbol{x}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{x}_T)$ from the pre-collected experience $\mathcal{D}$
  for $t = 1, 2, \ldots, T$ **do**
    Sample boundary conditioned on entire trajectory $m_t \sim q_{\boldsymbol{\theta}}(m_t \mid \boldsymbol{x}_{1:t})$
    Sample skill from skill posterior $z_t \sim q_{\boldsymbol{\theta}}(z_t \mid m_t, \boldsymbol{x}_{1:t}, \boldsymbol{a}_{1:t})$
    Sample abstraction from abstraction posterior $\boldsymbol{s}_t \sim q_{\boldsymbol{\theta}}(\boldsymbol{s}_t \mid z_t, \boldsymbol{x}_t)$
    Compute probability of the correct action from the decoder $p_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t)$
  end for
  Compute lower bound on maximum likelihood objective $\mathcal{L}_{\text{ELBO}}$ according to Equation 4
  Compute compression objective $\mathcal{L}_{\text{CL}}$ according to Equation 1
  Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla(\mathcal{L}_{\text{ELBO}} + \lambda \mathcal{L}_{\text{CL}})$ and λ
end while

Algorithm 2 Executing skill $\boldsymbol{z}$

while skill has not terminated **do**
  Compute state abstraction $\mu_{\boldsymbol{s}_t} = \mathbb{E}_{s \sim q_{\boldsymbol{\theta}}(\boldsymbol{s}_t \mid \boldsymbol{z}, \boldsymbol{x}_t)}[s]$
  Take action $\boldsymbol{a}_t = \arg\max_{\boldsymbol{a}} \; p_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \mu_{\boldsymbol{s}_t})$
  Observe next state $\boldsymbol{x}_{t+1}$
  Terminate if $\mathbb{E}_{\boldsymbol{m}_{t+1} \sim q_{\boldsymbol{\theta}}(\boldsymbol{m}_{t+1} \mid \boldsymbol{x}_{1:t+1})}[\boldsymbol{m}_{t+1}] > 0.5$
end while

H Visualization of Learned Skills

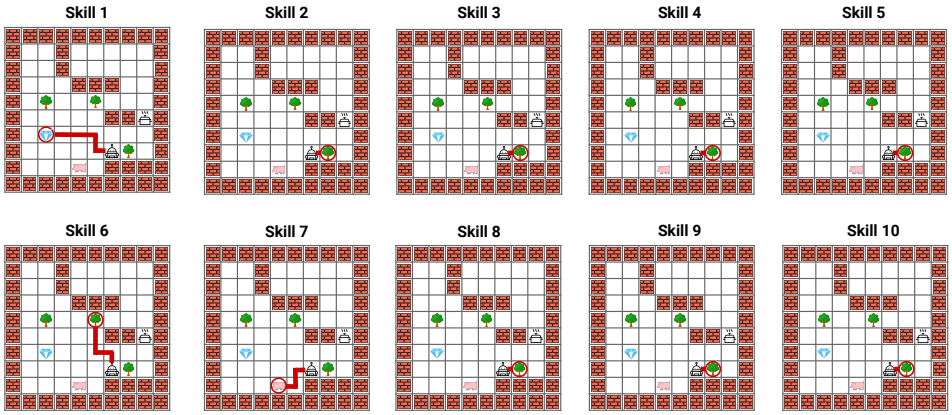

Figure 8: Skill visualization for all 10 skills of LOVE.

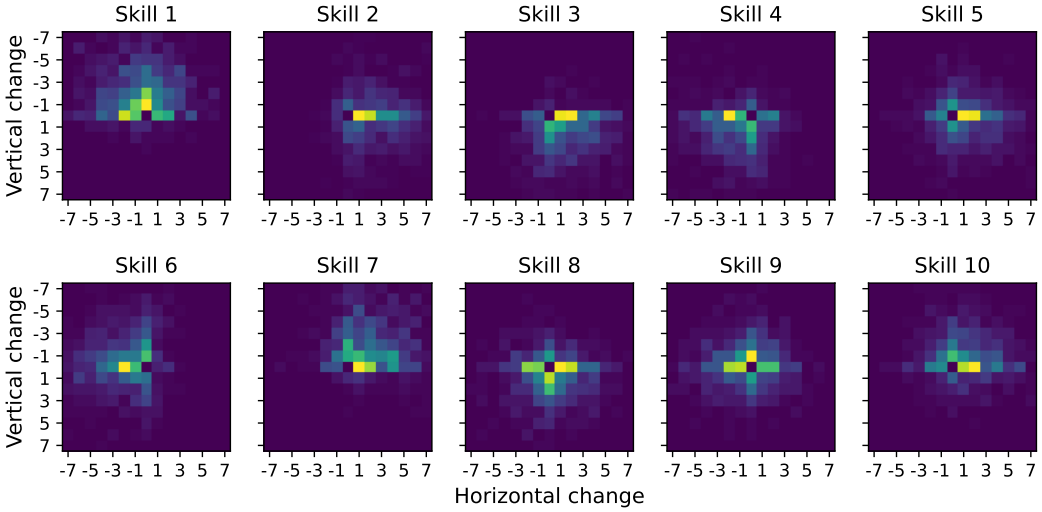

Figure 9: Heat map the agent's end position minus the agent's start position after applying each skill. In other words, this shows what direction the agent moves in after following each skill.

In Figure 8, we visualize the behavior of following each of LOVE's 10 skills on an example task. Each skill moves to and picks up an object, and the total set of skills covers 3 of the 4 object types in the task. These skills pick up both nearby objects, such as the tree in skill 2, as well as more distant objects, such as the diamond in skill 1. This contrasts DDO and VTA, which do not learn skills that move to and pick up objects.

To further understand what each of LOVE's skills does, we analyze whether there is a correlation between each skill and either the type of object it picks up, or the location of the object it picks up. We find that there appears to be little correlation between each skill and the type of object it picks up. Plotting the frequency of picking up each object type by skill shows that each skill picks up each object type with roughly the same frequency. Instead, there appears to be a correlation between each skill and the location of the object it picks up, illustrated in Figure 9. For example, skill 1 appears to specialize in moving to and picking up objects that are above the agent, while skill 7 tends to pick up objects that are up and to the right of the agent. It is also interesting to note that some skills are much more biased to move in certain direction (e.g., skill 1, 6, 7) while some appear to be more general (e.g., 4, 5, 8, 9) and move in any direction. Note in Figure 8, it may seem LOVE's skills seem short / redundant. This is due to the fact that LOVE's skills each pick up an object, so they naturally appear short when the agent is close to the objects, as in Figure 8. However, when the

agent is far away from the objects, the skills still pick up the objects and are much longer. From Figure 9, we can see from the heatmap that even the skills that appear to do same thing in Figure 8 have very distinct behaviors depending on the state they are in. Also, recall LOVE imposes a sparse distribution over skills (Section 5). After filtering, skill 3 and 7 in Figure 8 are dropped since they have low marginal probability and are not used to describe a significant part of the trajectories. Figure 8 includes redundant skills LOVE does not use; the used skills in the sparse distribution have little redundancy.

I Option Critic Results

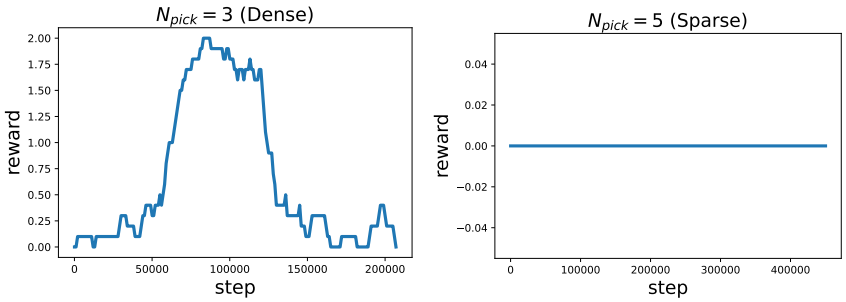

Figure 10: Results of running option critic [4] on 2 of the settings we considered in Figure 5.

Option critic [4] is a classical online HRL algorithm. We show the results of running option critic[4] on two of the four RL tasks we considered in Figure 10. The best return achieved over the training is shown in Table 4. We set the number of option to 8 following Bacon et al. [4] and leave other hyperparameters untouched. $N_{\text{pick}} = 3$ with dense reward is the easily setting and $N_{\text{pick}} = 5$ with sparse reward is the hardest setting. We see that in $N_{\text{pick}} = 3$, option critic is able to reach reward of 2 (maximum possible is 3) at around 1M environment steps but the performance deterioates afterwards. In $N_{\text{pick}} = 5$, the algorithm fails to make any meaningful progress. These observations may suggest online HRL algorithms like option critic may be insufficient for solving these tasks.

	$N_{\text{pick}} = 3$ (Dense)	$N_{\text{pick}} = 5$ (Sparse)
Option-Critic	2.0	0.0
LOVE (Ours)	**3.0**	**0.7**

Table 4: Comparison between LOVE and Bacon et al. [4] on the two RL tasks we consider in this work. Each entry is the maximum average return achieved during the course of training for both algorithms. For $N_{\text{pick}} = 3$ (Dense), 3.0 is the maximum possible return. For $N_{\text{pick}} = 5$ (Sparse), 1.0 is the maximum possible return.

J Comparison to Zhang et al. [83]

	Simple Colors		*Conditional Colors*		*Navigation*	
	MDL	**LOVE**	**MDL**	**LOVE**	**MDL**	**LOVE**
Precision	0.87	**0.99**	0.84	**0.99**	0.79	**0.90**
Recall	0.78	**0.85**	0.82	**0.83**	0.34	**0.94**
F1	0.82	**0.91**	0.83	**0.90**	0.48	**0.92**

Table 5: Comparison between LOVE and Zhang et al. [83] on the three segmentation tasks we consider in this work. We refer to Zhang et al. [83] as **MDL** in the table.

Zhang et al. [83] learn open-loop skills, which do not condition on the state and therefore cannot adapt to different states. Further, the MDL used in Zhang et al. [83] is equivalent to the variational inference used by VTA (on a different graphical model), which can be seen as greedily compressing

[4] `https://github.com/lweitkamp/option-critic-pytorch`

each skill independently, rather than compressing a whole trajectory as LOVE does. Table 5 reports segmentation results on the Color domain and grid world, akin to Tables 2 and 3. Zhang et al. [83] performs the same as VTA on the Color domains – as they both use variational inference and there is no state – which achieves lower precision / recall than LOVE. On the grid world navigation, Zhang et al. [83] fails to recover the boundaries, unlike LOVE, because skills that navigate to objects require observing the state. This problem is shared by all open-loop approaches.