# Temporally Abstract Model-Based Reinforcement Learning with Variable-Length Skills

Thomas Wei

CMU-RI-TR-23-80

August 2023

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

**Thesis Committee:**

Howie Choset (Chair)
Jeff Schneider
Ben Freed

*Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.*

*For my family*

# Abstract

Building agents that can reason and plan at various levels of temporal abstraction has long been a goal of artificial intelligence research. In the context of reinforcement learning (RL), performing temporal abstraction typically implies the use of behavioral primitives that are executed over extended periods of time. These behavioral primitives are sometimes known as *options*, but following more recent literature, we refer to them as *skills*. Many existing model-free and all model-based methods for temporal abstraction in RL use skills that execute for a fixed length of time, constraining an agent to a single level of temporal abstraction throughout the entirety of a task. Variable-length skills allow an agent to flexibly change its level of abstraction, which is desirable as many interesting tasks are composed of sub-tasks that can take vastly different amounts of time. Furthermore, state-dependent termination conditions allow a skill to end exactly once its sub-task has been completed and not simply after a predetermined amount of time.

This thesis presents a novel approach to temporal abstraction in model-based RL that allows agents to learn a set of variable-length skills, state-dependent termination conditions, and dynamics on the temporally coarsened timescale induced by those skills. We demonstrate in long-horizon tasks from Datasets for Deep Data-Driven Reinforcement Learning (D4RL) that variable-length skills induce more repeatable state transitions, enabling higher downstream task performance compared to fixed-length skills. Furthermore, our approach is competitive with existing offline RL algorithms while naturally enabling knowledge transfer to novel goals.

# Contents


**1 Introduction** **1**

**2 Background** **5**

2.1 Reinforcement Learning . . . . . 5

2.1.1 Markov Decision Processes and Partial Observability . . . . . 5

2.1.2 Online and Offline Reinforcement Learning . . . . . 7

2.1.3 The Options Framework . . . . . 7

2.2 Latent Variable Models . . . . . 7

2.2.1 Variational Autoencoders . . . . . 8

2.2.2 Sequential Variational Autoencoders . . . . . 9

**3 Learning Temporal Structure in Data** **11**

3.1 Previous Literature . . . . . 11

3.2 Preliminaries . . . . . 12

3.2.1 Attention and Transformers . . . . . 12

3.2.2 Concrete (Gumbel-Softmax) Random Variables . . . . . 13

3.3 Method . . . . . 14

3.3.1 Temporally Abstract Data-Generating Process . . . . . 14

3.3.2 Continuous Relaxation and Architecture . . . . . 18

3.4 Evaluation . . . . . 20

3.4.1 Learned Segmentations . . . . . 20

3.4.2 Skill Length Distributions . . . . . 22

**4 Learning and Planning Over Variable Length Skills** **25**

4.1 Previous Literature . . . . . 25

4.2 Preliminaries . . . . . 26

4.2.1 Learning Fixed-Length Skills from Offline Data . . . . . 26

4.2.2 Planning with the Cross-Entropy Method . . . . . 27
4.3 Method . . . . . 28
4.3.1 Training Data Preprocessing . . . . . 29
4.3.2 Learning Variable-Length Skills from Offline Data . . . . . 29
4.4 Evaluation . . . . . 31
4.4.1 Quality of Learned Skills . . . . . 31
4.4.2 Skill Planning on Downstream Tasks . . . . . 32

**5 Conclusion and Future Work** **35**

**Bibliography** **39**

# List of Figures

1.1 Overview of our approach to model-based temporal abstraction with variable-length skills. Our approach proceeds in three stages: 1) a transformer-based segmentation model is learned using a variational objective. The segmentation model splits trajectories into segments corresponding to single skills. 2) Trajectory segments are fed into a skill model, which learns a low-level policy (action decoder), a termination condition, and an abstract dynamics model. 3) The abstract dynamics model is used to plan skill sequences in an online fashion to accomplish a downstream task. . . . . 3

2.1 The probabilistic graphical models for the dynamics of a fully observable MDP (top) and a POMDP (bottom). Shaded variables are observed, while white are unobserved. . . . . 6

2.2 The probabilistic graphical models that are assumed by the variational autoencoder (left), an unforced sequential variational autoencoder (middle), and a forced sequential variational autoencoder (left). Shaded variables are observed, while unshaded are unobserved. . . . . 8

3.1 Probabilistic graphical model for learning a segmentation model. Here, $\mathbf{o}_t$ and $\mathbf{a}_t$ represent the observation and action associated with timestep $t$, respectively. $\mathbf{s}_t$, $\mathbf{z}_t$, and $\mathbf{m}_t$ are latent variables. $\mathbf{m}_t$ specifically denotes whether the current skill (represented by $\mathbf{z}_t$) terminates at timestep $t$. If not, then the skill variable remains constant in the next timestep, *i.e.*, $\mathbf{z}_t = \mathbf{z}_{t+1}$. The left side shows the full PGM with all latent variables and particular values of $\mathbf{m}_t$ displayed next to their respective nodes. The right side shows the simplification of the PGM that occurs once the skills that take the same value due to the assignment of all $\mathbf{m}_t$ are grouped together. . . . . 15

3.2 A depiction of our skill encoding scheme. The dotted vertical lines are segmentation variables $\mathbf{m}_t$ that take values greater than 0. The length of the dots represents the continuous value taken by these $\mathbf{m}_t$, longer dots implying a higher value. The $\tilde{\tau}_t$ represent our original sequential data, $o_t, a_t$ that have been preprocessed before the skill encoding step. Each $\mathbf{z}_i$ is encoded from the $\tilde{\tau}_t$ with its corresponding attention mask, depicted in the same color. These $\mathbf{z}_i$ are then used to condition the corresponding low-level latent variables $\mathbf{s}_t$. . . . . 18
3.3 Frames taken from segmentation points discovered by our segmentation model. These frames occur at semantically meaningful points in the trajectory and thus can be easily labeled with human-interpretable skills. Many of these frames also occur at points where trajectories deviate from one another as they accomplish different subsets of kitchen tasks. . . . . 18
3.4 Average number of timesteps per trajectory segment vs. epochs for different values of the abstraction bonus coefficient. Our method (top) finds similar segmentation schemes for many different coefficient values, resulting in a bimodal distribution. VTA (bottom), which has been modified to use an abstraction bonus rather than a prior over skill lengths, learns segmentations that are much more sensitive to the abstraction bonus coefficient. . . . . 21
3.5 A comparison of action reconstructions and segmentations learned by our model and a number of baselines. . . . . 23
3.6 The empirical distributions over skill length learned by our approach in the Maze2d and Kitchen environments. . . . . 24

4.1 The various components of our skill model. Here $L(\tau)$ is the length of the particular trajectory segment $\tau$. . . . . 30
4.2 The empirical distributions of the error between the induced terminal state from rolling out a learned skill-conditioned policy and the estimated terminal state from skill-conditioned abstract dynamics for both the Maze2d and Kitchen environments. Outliers beyond the range of the graph are clamped to within the range (right-hand side of both graphs). . . . . 31

# List of Tables

4.1 Maze2d and Kitchen D4RL task scores with the default goal specified by the original task. We evaluate Maze2d environments by the percent of episodes where the goal is reached and the Kitchen environments with normalized scores. . 32
4.2 Scores for the Maze2d and Kitchen D4RL tasks with randomized goals. . . . . 32
4.3 Scores for the Maze2d Medium environment for variable length skills with segmentation points learned by our model, fixed length skills (OPOSM), and variable length skills learned by VTA with an informative prior. Results for both the default goal specified by the task and randomly sampled goals. . . . . 33

# Chapter 1

# Introduction

Temporal abstraction refers to the process of reasoning or decision-making on coarse-grained timescales. Agents that can leverage temporal abstraction are better able to deal with long-time-horizon tasks because they can more effectively capture long-time-horizon dependencies and ignore irrelevant fine-grained details [30]. It has, therefore, long been a goal of artificial intelligence research to build agents that can automatically learn and make effective use of their own temporal abstractions when solving tasks [1, 2, 6, 7, 10, 24, 25, 27, 28, 30].

A key challenge when learning temporally abstract representations of a task is finding the "correct" level of abstraction for a particular situation. We, as humans, are able to dynamically vary the coarseness of the abstractions we use, allowing us to flexibly adjust the level of our abstraction to suit our needs. For example, when planning a route for a road trip, we may abstract away extended periods of monotonous highway driving where few decisions are necessary while preserving detail during periods of rapid decision-making, such as when driving through a busy city. The flexibility to dynamically alter our level of abstraction allows us to maximize task performance while minimizing the cognitive load of planning. If we were instead constrained to plan on one particular timescale, say 10-minute increments, we would waste huge amounts of effort in constructing needlessly detailed plans while simultaneously failing to capture sufficient detail when rapid decision-making is necessary. Despite the obvious suboptimality of such a fixed-timescale approach, this is precisely what is done by many recent approaches to temporal abstraction in reinforcement learning (RL). Here, we address this shortcoming by developing what is, to our knowledge, the first model-based temporal abstraction approach capable of adjusting the level of abstraction in a reactive manner.

Many recent approaches to temporal abstraction in RL are built upon the *options* framework, introduced in Sutton et al. 1999. Options are temporally extended behavioral primitives, often referred to as *skills*, and are comprised of a policy, an initiation set, and a termination condi-

tion. Here, a *policy* refers to a mapping from states to actions that is associated with a particular skill, an *initiation set* is a set of states from which the corresponding skill can be activated, and a *termination condition* indicates when the current skill should be terminated, given the current state. Because options are temporally extended, agents that use options can model state transitions and make decisions at a coarser time granularity [30]. Many recent temporally abstract RL algorithms use some form of variational objective to discover options and then use either model-free [1, 2, 6, 10] or model-based RL [7, 24, 25, 27] to select skills for a downstream task. In this thesis, we focus on model-based temporal abstraction because models enable agents to plan for counterfactual situations, making them more data efficient. Additionally, models naturally enable knowledge transfer to novel tasks that share the same state-transition dynamics [7].

Many of the model-free approaches to temporal abstraction in RL, and all model-based approaches that we are aware of, are constrained to learn skills that terminate after a fixed amount of time. As mentioned earlier, this fixed-length skill constraint is problematic because it locks the agent into a fixed level of temporal abstraction, which must be hand-selected by the algorithm designer. Perhaps even more importantly, the fixed-length constraint does not allow skills to terminate in a *state-dependent* manner. It is natural to assume that we may want to learn skills that accomplish an intended *goal* in the environment, *i.e.,* take the agent to a particular region of the state space. Particularly in stochastic environments, the agent may have no way of determining how much time the skill will require to accomplish its goal at its outset, causing the agent to potentially overshoot or undershoot its goal. This is particularly detrimental in a model-based setting, where the agent uses the skill-conditioned state-transition dynamics to plan [7, 24, 25, 27]. Because fixed-length skills do not result in precise state transitions, the agent cannot possibly learn a temporally abstract dynamics model that is both precise and accurate. If, on the other hand, skills only terminate once the agent reaches a narrow region of the state space $G$, it becomes possible to precisely predict state transitions, as the temporally abstract dynamics model need only have support in $G$.

In this work we propose a novel method for performing model-based temporal abstraction with variable-length skills from offline data. Our approach proceeds in three stages. First, a novel attention-based *segmentation model* is learned, which segments state-action trajectories such that the demonstrated behavior in each segment is well-described as belonging to a single skill. Second, trajectory segments are used to train a VAE-based *skill model*, which contains a low-level policy that maps the current state and skill to actions, an abstract dynamics model that predicts the terminal state in a segment given the initial state and skill, and a state-dependent termination condition, which specifies whether a skill should terminate at each timestep given the current state. Unlike past approaches that sampled *fixed-length* trajectory segments *uniformly at*

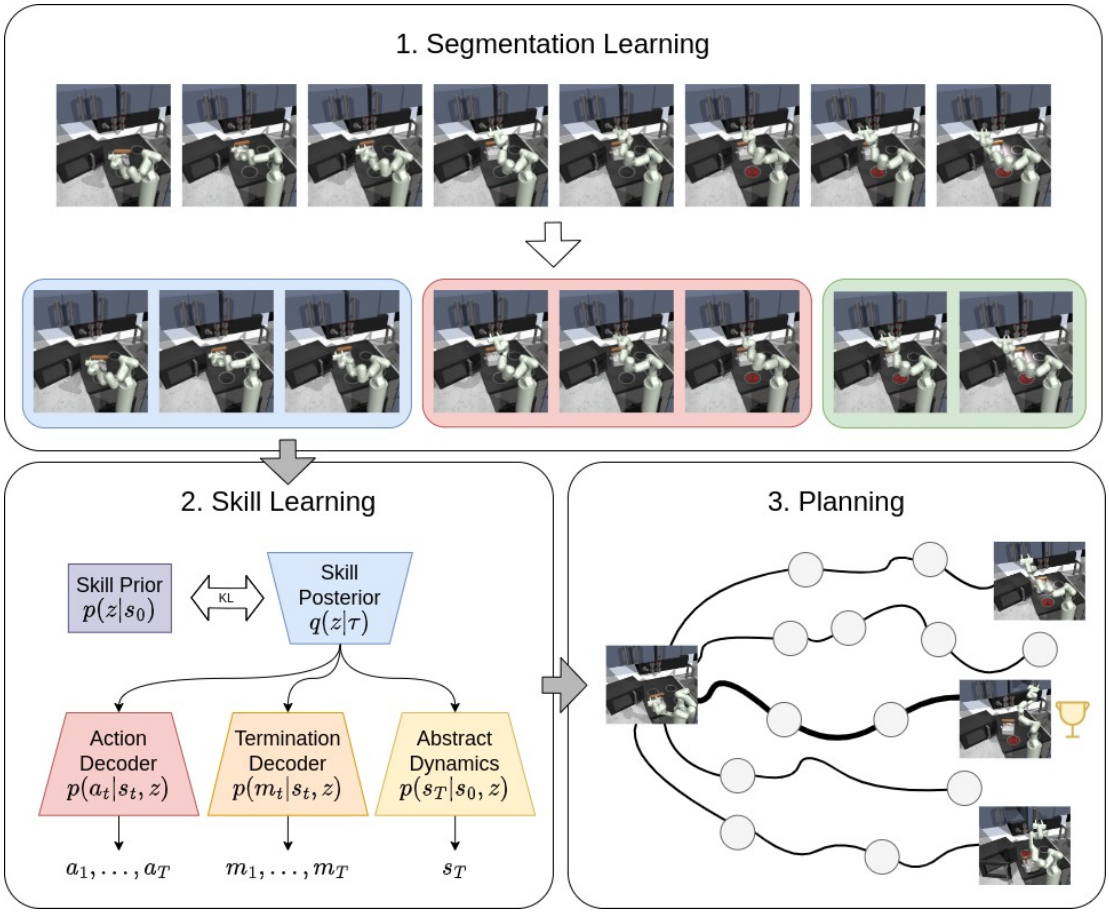


Figure 1.1: Overview of our approach to model-based temporal abstraction with variable-length skills. Our approach proceeds in three stages: 1) a transformer-based segmentation model is learned using a variational objective. The segmentation model splits trajectories into segments corresponding to single skills. 2) Trajectory segments are fed into a skill model, which learns a low-level policy (action decoder), a termination condition, and an abstract dynamics model. 3) The abstract dynamics model is used to plan skill sequences in an online fashion to accomplish a downstream task.

*random* from the dataset, we train our skill model on *variable-length* segments, whose endpoints were identified as corresponding to the endpoints of skills by the segmentation model. Finally, we use our temporally abstract dynamics model to iteratively re-plan skill sequences in an online fashion for downstream tasks.

We find that our trajectory segmentation model is capable of identifying semantically meaningful skills in the D4RL kitchen tasks, which involve a simulated Franka robot arm tasked with reconfiguring objects in a kitchen. Furthermore, we show that our skill model learns state transition dynamics that are more accurate, compared to a fixed skill-length approach such as OPOSM [7]. Finally, we find that our approach is competitive with existing offline RL algorithms in the Kitchen and Maze2d D4RL tasks, including other approaches that utilize temporal abstraction.

# Chapter 2

# Background

## 2.1 Reinforcement Learning

Reinforcement learning (RL) is the process of an agent learning a mapping from situations to actions in order to maximize cumulative rewards over time [29]. The agent does not have direct access to the dynamics of the environment. Instead, the dynamics must be inferred from trajectories of observations and actions generated through interaction with the environment.

Many variants of the RL problem exist. Some assume full observability of the environmental state, while others do not. In some cases, the agent is required to learn from scratch; in others, access to trajectories generated by a demonstrator is provided to the agent beforehand. This section will provide background on all variants of the RL problem that are relevant for understanding the work presented in this thesis.

### 2.1.1 Markov Decision Processes and Partial Observability

The simplest and most well-studied RL setting is the fully observable Markov Decision Process (MDP). At every timestep, the agent takes as input the environment state $\mathbf{s}_t \in S$ and produces a distribution over actions from which a particular action $\mathbf{a}_t \in A$ is sampled. Given $\mathbf{a}_t$, the environment samples the next state $\mathbf{s}_{t+1}$ from the state transition distribution $P(s_{t+1}|s_t, a_t)$ and a scalar reward from the reward distribution $\mathbf{r}_t \sim R(r_t|s_t, a_t, s_{t+1})$.

The RL objective is to find the policy $\pi^*$ that maximizes the expected cumulative sum of rewards

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{\vec{\mathbf{s}},\vec{\mathbf{a}},\vec{\mathbf{r}}\sim p^\pi(\vec{s},\vec{a},\vec{r})} \left[ \sum_{t=1}^{T} \mathbf{r}_t \right] \tag{2.1}$$

where $p^\pi(\vec{s}, \vec{a}, \vec{r})$ is the distribution over sequences of states, actions, and rewards induced by the

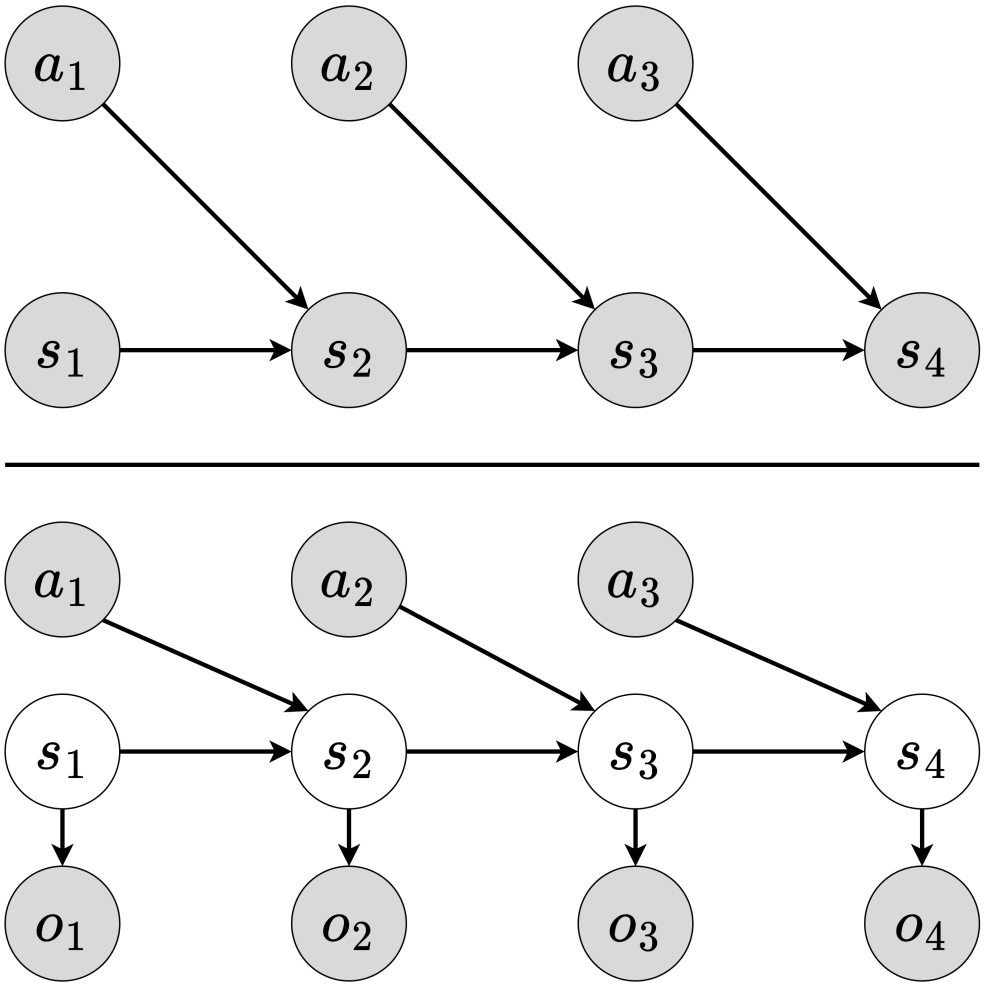


Figure 2.1: The probabilistic graphical models for the dynamics of a fully observable MDP (top) and a POMDP (bottom). Shaded variables are observed, while white are unobserved.

interactions of policy $\pi(a_t|s_t)$ with the environment.

Another RL setting of interest is the Partially Observable MDP (POMDP). In a POMDP, the agent does not get direct access to the environment state $\mathbf{s}_t \in S$ but rather receives a sample from an observation distribution $\mathbf{o}_t \in O$ generated by the environment state. The fully observable MDP can be seen as a special case of a POMDP where $O = S$ and $p(o_t|s_t) = \delta(o_t - s_t)$. However, in general, the observation $\mathbf{o}_t$ can contain less information about the environment than the state $\mathbf{s}_t$, which fully describes the environment at time $t$. Common examples of this include the case where observations are noisy measurements of the true state and the case of image observations, where the information contained in a single image may be sufficient to identify the position of bodies of interest, but velocity information requires multiple frames to accurately infer.

### 2.1.2 Online and Offline Reinforcement Learning

In the original RL formulation, the agent begins with no information about the environment and must improve its policy via its own interactions with the environment. This has since come to be known as online reinforcement learning. In contrast, offline RL, a.k.a. batch RL, initially gives the agent access to trajectories generated by a demonstrator but does not allow the agent to improve via trial and error in the environment. A common example of an application for offline RL is autonomous driving. In the autonomous driving domain, high-quality demonstration data can be easily gathered from human drivers, whereas the actions of a partially trained agent could have extremely costly consequences. There also exist hybrid variations on the spectrum between strictly online RL and strictly offline RL, where both demonstration trajectories are provided, and the agent is allowed to improve via online interaction with the environment. The work in this thesis focuses on the strictly offline RL problem setting.

### 2.1.3 The Options Framework

The options framework [30] introduced a general formulation for temporally abstract subroutines in MDPs. Dubbed options, these temporally abstract subroutines consist of three components: a policy distribution over actions $\pi(a_t|s_t)$, a Bernoulli termination distribution $\beta(s_t)$, and a subset of states that the option can be initiated from. When an option is initiated, its corresponding policy executes in the environment until a termination event is sampled from the termination distribution.

When agents have access to executing meaningful options, either alongside or instead of base-level actions, planning can be greatly accelerated, especially on problems of long time horizon [30]. More recently, OPAL [2] and OPOSM [7] have shown that semi-Markov, fixed-length options learned from the demonstrations in an offline dataset can also improve performance in offline policy learning and planning over a learned model, respectively.

## 2.2 Latent Variable Models

Latent variable models are a class of models that assume there exist hidden or unobserved variables that are related to the observable data of interest. The specific manner in which the joint distribution over observed and unobserved variables is composed of simpler distributions characterizes different families of latent variable models. This compositional probabilistic structure can be represented as a probabilistic graphical model (PGM) such as those in Figure 2.2. Latent variable models aim to encode underlying sources of variation into the latent variables in a

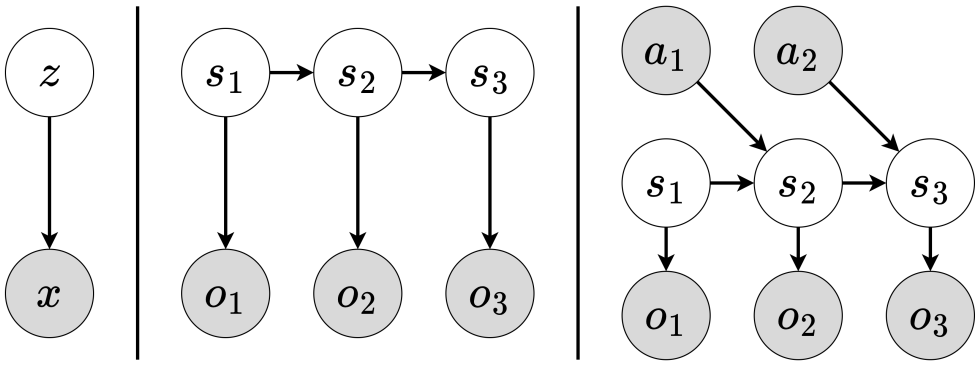


Figure 2.2: The probabilistic graphical models that are assumed by the variational autoencoder (left), an unforced sequential variational autoencoder (middle), and a forced sequential variational autoencoder (left). Shaded variables are observed, while unshaded are unobserved.

manner consistent with the assumed probabilistic structure.

### 2.2.1 Variational Autoencoders

One of the simplest latent variable models is the variational autoencoder (VAE) [16]. The probabilistic graphical model assumed by the VAE can be seen in the left side of Figure 2.2. A VAE assumes that a sample of observed data $\mathbf{x}$ is sampled from a distribution conditioned on a corresponding latent random variable $\mathbf{z}$. Assuming such a probabilistic structure allows us to write the joint distribution as

$$p(z, x) = p(z)p_\theta(x|z) \tag{2.2}$$

where $p_\theta(x|z)$ is parameterized by a neural network with parameters $\theta$. We want to learn the parameters $\theta^*$ under which the marginal probability of our data is maximized:

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, p(x) = \underset{\theta}{\operatorname{argmax}} \int p(z)p_\theta(x|z)dz. \tag{2.3}$$

However, the integration required to evaluate the marginal probability is typically intractable. Therefore, Kingma and Welling adopt a variational approach and derive an evidence lower bound (ELBO) on the marginal log probability of the observed data.

In order to do so, they introduce the approximate posterior $q_\theta(z|x)$, which is also parameterized by a neural network. We will group all optimizable neural network parameters into $\theta$ for

simplicity. Then we can derive the ELBO as

$$\log p(x) = \log \int p(z) p_\theta(x|z) dz \tag{2.4}$$

$$= \log \int p(z) p_\theta(x|z) \frac{q_\theta(z|x)}{q_\theta(z|x)} dz \tag{2.5}$$

$$= \log \mathbb{E}_{\mathbf{z} \sim q_\theta(z|x)} \left[ \frac{p(\mathbf{z}) p_\theta(x|\mathbf{z})}{q_\theta(\mathbf{z}|x)} \right] \tag{2.6}$$

$$\geq \mathbb{E}_{\mathbf{z} \sim q_\theta(z|x)} \left[ \log \frac{p(\mathbf{z}) p_\theta(x|\mathbf{z})}{q_\theta(\mathbf{z}|x)} \right] \tag{2.7}$$

$$= \mathbb{E}_{\mathbf{z} \sim q_\theta(z|x)} \left[ \log p_\theta(x|\mathbf{z}) \right] - D_{KL}(q_\theta(z|x)||p(z)), \tag{2.8}$$

where the inequality is due to the concavity of the log function and Jensen's inequality. The original VAE [16] used a standard normal distribution for $p(z)$ and enforced that $q_\theta(z|x)$ was also a normal distribution whose parameters were output by a neural network. In order to make the computation compatible with standard autodifferentiation software used to optimize neural networks, the reparameterization trick for sampling normal random variables was used to turn the expectation over $\mathbf{z}$ into an expectation over noise from a standard normal distribution. Noise from a standard normal distribution can be seen as another input to the computation graph rather than an intermediate node, as would be the case if $\mathbf{z}$ was sampled directly from the posterior. This allows backpropagation to proceed without the need for any modifications for propagating through a stochastic computation tree node.

### 2.2.2 Sequential Variational Autoencoders

The standard VAE can be extended to the sequential PGM depicted in the middle of Figure 2.2 to get a sequential VAE. In this PGM, the latent variable $\mathbf{s}_t$ both conditions the distribution from which observation $\mathbf{o}_t$ is sampled and also affects the distribution over the next latent variable $\mathbf{s}_{t+1}$. We can write the joint distribution over a length $T$ sequence as

$$p(\vec{s}, \vec{o}) = \prod_{t=1}^{T} p_\theta(o_t|s_t) p_\theta(s_t|s_{t-1}). \tag{2.9}$$

Through a derivation similar to that of the standard VAE objective, we can derive an ELBO on the log probability of our observed sequences of observations of the form

$$\log p(\vec{o}) \geq \sum_{t=1}^{T} \left( \mathbb{E}_{\mathbf{s}_t \sim q_\theta(s_t|o_{\leq t})} \left[ \log p_\theta(o_t|\mathbf{s}_t) \right] - D_{KL}(q_\theta(s_t|o_{\leq t})||p_\theta(s_t|s_{t-1})) \right). \tag{2.10}$$

The sequential latent variable setting can be further extended to the case where actions influence the evolution of $s_t$ over time [11], as depicted in the right side of Figure 2.2. The modification to the PGM results in the following ELBO on the conditional probability of the observations given the actions taken:

$$\log p(\vec{o}|\vec{a}) \geq \sum_{t=1}^{T} \left( \mathbb{E}_{\mathbf{s}_t \sim q_\theta(s_t|o_{\leq t}, a_{<t})} \left[ \log p_\theta(o_t|\mathbf{s}_t) \right] - D_{KL}(q_\theta(s_t|o_{\leq t}, a_{<t}) || p(s_t|s_{t-1}, a_{t-1})) \right). \tag{2.11}$$

Hafner et al. enforce that the dimensionality of the latent variables learned is far lower than the dimensionality of the observation space, allowing for more efficient policy rollouts in latent space.

Hafner et al. also introduce the RSSM which adds a deterministic, RNN-computed representation component to each latent variable. This can be thought of as simply enforcing the prior assumption that some elements of the latent variable be computed deterministically (have a Dirac delta distribution) with the added benefit that the deterministic pathway for gradient computation allows for long-term dependencies to be more easily learned than a purely stochastic model would be. More recently, Yan et al. introduced a transformer-based [32] variant of the RSSM that retains the same original benefits with the added computational and optimization improvements that transformers provide.

# Chapter 3

# Learning Temporal Structure in Data

## 3.1 Previous Literature

Extracting temporal structure from sequential data in an unsupervised manner has long been a topic of interest for its potential applications to reinforcement learning, long-sequence data generation, and many other problems. As such, a number of prior works have proposed their own solutions to this problem, the most relevant of which are detailed in this section. At a high level, all of these approaches fundamentally aim to group consecutive data points together into segments when they share similar latent generative factors.

Variational Temporal Abstraction (VTA) [15] combines a variational inference approach to sequence modeling with a learnable, hierarchical temporal structure, enabling variable-length temporal abstractions, or segments, to be learned. Learning Options Via Compression (LOVE) [14] extends this approach to the RL setting by converting the evidence lower bound (ELBO) used in VTA to an ELBO on selected actions present in an offline RL dataset of demonstrations, given the states. LOVE also introduces a compression term that is added to the ELBO objective to incentivize learning temporal abstractions that more effectively compress trajectories. These approaches are built on hierarchical variants of the RSSM introduced in [11]. Temporal Variational Inference (TVI) [23] also takes a variational approach to the problem of discovering temporal abstractions of demonstration data but assumes a fully observable environment and Markovian demonstration policy. Our approach also takes a hierarchical variational inference approach to the problem of discovering temporal abstractions from data.

Both VTA and LOVE rely on a variant of the straight-through estimator [5] in order to make the process of sampling termination points of segments differentiable by standard autodifferentiation libraries. In contrast to VTA and LOVE, Compositional Imitation Learning and Execution (CompILE) [17] uses the Concrete random variable [20] to sample its termination points between

segments. Because Concrete random variables take continuous values, this results in a continuous relaxation to the problem of dividing a trajectory into discrete segments. Our method uses a different continuous relaxation of the trajectory segmentation problem, which is made possible by a novel, attention-based encoding method.

While all the previously discussed approaches separately predict termination events and skill transitions, Directed-info GAIL [26] and Lee and Seo 2020 unify these operations by predicting a skill, or segment, transition at every point.

## 3.2 Preliminaries

In this section, we review attention, transformers, and Concrete random variables. Together, along with sequential variational autoencoders detailed in Chapter 2, these form the building blocks of the unsupervised learning algorithm that allows us to discover temporal structure in data.

### 3.2.1 Attention and Transformers

Attention [4] is a method of aggregating multiple potential input vectors into a single output vector. The amount each input vector contributes to the composition depends on some importance to, or alignment with, a given context vector. More formally, given input vectors $h_1, \ldots, h_n \in \mathbb{R}^d$, context vector $g$, and alignment function $f$, the aggregated input vector $a$ is computed as

$$a = \sum_{i=1}^{n} \alpha_i h_i, \tag{3.1}$$

where $\alpha_i$ are weights normalized via the softmax operation

$$\alpha_i = \frac{\exp\left(f\left(h_i, g\right)\right)}{\sum_{j=1}^{n} \exp\left(f\left(h_j, g\right)\right)}. \tag{3.2}$$

The alignment function $f$ is usually parameterized as a neural network whose weights can be jointly learned with other components of a larger model via backpropagation.

The Transformer [32] is a sequence processing model that uses attention as its primary mode of computation. The type of attention used in a Transformer is self-attention, which uses a mapping of $h_i$, usually linear, as the context vector (also known as the query vector) to produce aggregated output $a_i$. This process is performed simultaneously for every $i$ via the following

matrix operations:

$$A = (W^V H)\,\mathrm{softmax}\left(\frac{(W^K H)^T (W^Q H)}{\sqrt{d}}\right) \tag{3.3}$$

where $A$ is the matrix formed by column vectors $\begin{bmatrix} a_1 & \dots & a_n \end{bmatrix}$ and $H$ is the matrix formed by $\begin{bmatrix} h_1 & \dots & h_n \end{bmatrix}$. $W^Q, W^K, W^V$ are $d \times d$ learnable parameter matrices and the softmax operation is performed column-wise. As seen above, the attention affinity function in the transformer has been chosen to be linear mappings followed by a scaled dot product. Transformers also employ several of these self-attention mechanisms in parallel, the results of which are aggregated by a subsequent linear map.

### 3.2.2 Concrete (Gumbel-Softmax) Random Variables

Concrete random variables [20], concurrently discovered under the name Gumbel-Softmax random variables [13], are continuous relaxations of discrete random variables that can be sampled via the reparameterization trick. This is important because it allows for gradients to be backpropagated through discrete stochastic nodes in a computation tree via standard automatic differentiation techniques.

The $n$-way concrete random variable is parameterized by a location vector $\alpha \in (0, \infty)^n$ and temperature $\lambda \in (0, \infty)$. It has a density over the $n-1$ simplex of the form

$$p_{\alpha,\lambda}(x) = (n-1)!\lambda^{n-1} \prod_{k=1}^{n} \left(\frac{\alpha_k x_k^{-\lambda-1}}{\sum_{i=1}^{n} \alpha_i x_i^{-\lambda}}\right). \tag{3.4}$$

As alluded to, a random variable $\mathbf{X}$ distributed according to $\mathrm{Concrete}(\alpha, \lambda)$ can be sampled via the reparameterization trick

$$\mathbf{X}_k = \frac{\exp\left(\left(\log \alpha_k + \mathbf{G}_k\right)/\lambda\right)}{\sum_{i=1}^{n} \exp\left(\left(\log \alpha_i + \mathbf{G}_i\right)/\lambda\right)} \tag{3.5}$$

where $\mathbf{G}_k$ are i.i.d according to $\mathrm{Gumbel}(0, 1)$.

Though the Concrete distribution has non-zero density in the interior of the simplex and thus produces continuous rather than discrete samples, with a sufficiently low temperature parameter $\lambda$ there are guaranteed not to be any modes in the interior of the simplex [20]. This leads models with Concrete latent random variables to concentrate mass near the vertices of the simplex rather than in the interior.

A fully discrete variant of the Concrete random variable was also introduced in [13]. In this variant, an $\mathrm{argmax}$ is taken over a vector sampled from the Concrete distribution to get a discrete

value. Then in the backward pass, the gradient computation proceeds as if the differentiable, continuous sample was used instead, similar to the straight-through estimator [5].

## 3.3 Method

In this section, we describe our novel approach to learning variable-length temporal abstractions in an unsupervised manner. In particular, we seek to learn a decomposition of demonstration trajectories from an offline RL dataset into segments where each segment represents the execution of a single skill. Note that although we apply this approach to discover skills for use in the offline RL setting, the ideas presented in this section could easily be applied to other problems where temporal abstraction is desirable, such as the generation of long-form text or video.

A primary challenge to learning segmentations is the fact that assigning individual timesteps to segments is a combinatorial optimization problem. Combinatorial optimization problems have discrete solution sets and, as such, are not naturally compatible with deep learning approaches. Therefore, we instead optimize a continuous relaxation of the problem, using Concrete random variables [13, 20]. Our architecture relies heavily on an attention mechanism [4] for encoding segments and transformers [32] for decoding, the latter being chosen due to their compatibility with soft masks produced by the former.

In the next section, we describe how we model the data-generation processes and the evidence lower bound that we use to train our segmentation model. Subsequently, we describe in detail our segmentation model architecture and the continuous relaxation that allows it to be effectively trained.

### 3.3.1 Temporally Abstract Data-Generating Process

We start by assuming that there exist high-level, temporally extended skills that are responsible for generating the actions, and thus trajectories, that we see in demonstrations. Because the points where one skill ends and the next begins are unknown, we adopt the probabilistic graphical model (PGM) in Fig. 3.1, which allows any possible segmentation of a trajectory to be represented by a particular assignment of the latent binary random variables $\vec{\mathbf{m}}$. The vector symbol denotes the sequence of variables corresponding to all timesteps in an episode, *i.e.*, $\vec{\mathbf{m}} = \{\mathbf{m}_1, ..., \mathbf{m}_T\}$. Similar to VTA [15], when a particular $\mathbf{m}_t$ takes the value $0$, we enforce that the skill $\mathbf{z}_{t+1}$ is equal to $\mathbf{z}_t$, whereas when $\mathbf{m}_t = 1$, then $\mathbf{z}_{t+1}$ will take on new value, distinct from $\mathbf{z}_t$. For our particular example in Figure 3.1, this means that skills $\mathbf{z}_1$ and $\mathbf{z}_2$ represent a single skill that is responsible for generating actions $\mathbf{a}_1$ and $\mathbf{a}_2$, and skills $\mathbf{z}_3$, $\mathbf{z}_4$, and $\mathbf{z}_5$ similarly represent a single

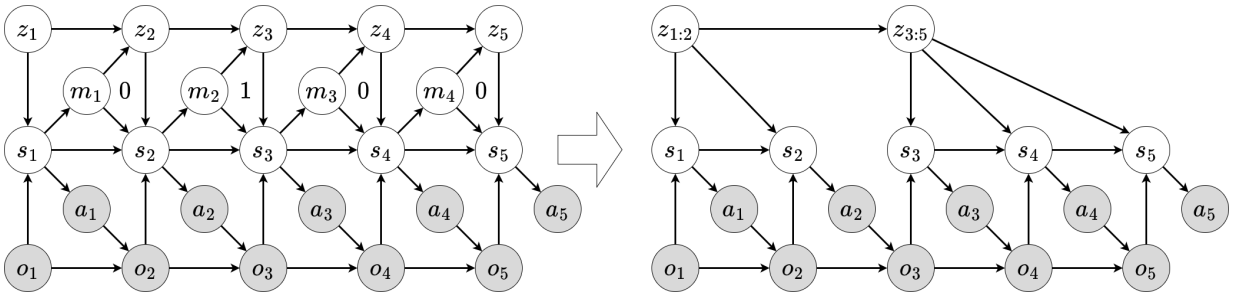


Figure 3.1: Probabilistic graphical model for learning a segmentation model. Here, $\mathbf{o}_t$ and $\mathbf{a}_t$ represent the observation and action associated with timestep $t$, respectively. $\mathbf{s}_t$, $\mathbf{z}_t$, and $\mathbf{m}_t$ are latent variables. $\mathbf{m}_t$ specifically denotes whether the current skill (represented by $\mathbf{z}_t$) terminates at timestep $t$. If not, then the skill variable remains constant in the next timestep, *i.e.*, $\mathbf{z}_t = \mathbf{z}_{t+1}$. The left side shows the full PGM with all latent variables and particular values of $\mathbf{m}_t$ displayed next to their respective nodes. The right side shows the simplification of the PGM that occurs once the skills that take the same value due to the assignment of all $\mathbf{m}_t$ are grouped together.

skill that generates $\mathbf{a}_3$, $\mathbf{a}_4$, and $\mathbf{a}_5$.

We follow prior work in this area [14, 15] in using a PGM with both high-level latent variables ($\vec{\mathbf{z}}$) and low-level latent variables ($\vec{\mathbf{s}}$), in addition to $\vec{\mathbf{m}}$. The low-level latent variables allow us to handle the partial observability of states and represent non-Markovian policies in the computation of segmentations and the reconstruction of actions. Though the environments we deploy our model on in this work do not require a non-Markovian treatment, we found it unnecessary to simplify our segmentation model as full observable MDPs are a special case of POMDPs.

The joint distribution of our assumed data generation process is given by

$$p(\tau, \vec{s}, \vec{z}, \vec{m}) = \prod_{t=1}^{T} p(o_t|o_{t-1}, a_{t-1}) p_\theta(a_t|s_t) p_\theta(s_t|o_t, s_{t-1}, z_t, m_{t-1}) p_\theta(m_t|s_t) p_\theta(z_t|z_{t-1}, m_{t-1}), \tag{3.6}$$

where for brevity, we let $\tau = (\vec{o}, \vec{a})$. In addition, the conditional distributions with the subscript $\theta$ are components of the data-generating process that we model with neural network parameterized distributions. Here, $\theta$ represents the parameters of those neural networks.

We aim to learn a segmentation model that maximizes the marginal probability of the observed data, $p(\tau)$; however, the marginalization process over latent random variables $\vec{\mathbf{s}}, \vec{\mathbf{z}}, \vec{\mathbf{m}}$ is intractable in general [16]. Therefore, we optimize a lower bound on the log of our desired objective instead. To encourage temporally extended skills, we also add a term corresponding to the average number of skills used in a trajectory.

**ELBO** As with other variational inference approaches, we introduce an approximate posterior $q_\theta(\vec{s}, \vec{z}, \vec{m}|\vec{o}, \vec{a})$, which we choose to factorize in the following manner:

$$q(\vec{s}, \vec{z}, \vec{m}|\tau) = \prod_{t=1}^{T} q_\theta(m_t|m_{<t}, \tau) q_\theta(z_t|\vec{m}, \tau) q_\theta(s_t|\vec{z}, \vec{m}, \tau). \tag{3.7}$$

In order to encode the previously discussed temporal structure, we enforce that if $m_{t-1}$ is $0$, then the posterior $q_\theta(z_t|\vec{m}, \tau)$ and prior $p_\theta(z_t|z_{t-1}, m_{t-1})$ are both set to a Dirac delta distribution at the previous skill, *i.e.,* $p(z) = \delta(z - z_{t-1})$.

We can then write the log of our objective as the log of an expectation with respect to our approximate posterior

$$\log p(\tau) = \log \iiint p(\tau, \vec{s}, \vec{z}, \vec{m}) \frac{q(\vec{s}, \vec{z}, \vec{m}|\tau)}{q(\vec{s}, \vec{z}, \vec{m}|\tau)} d\vec{s} d\vec{z} d\vec{m} = \log \mathbb{E}_{\vec{\mathbf{s}}, \vec{\mathbf{z}}, \vec{\mathbf{m}} \sim q} \left[ \frac{p(\tau, \vec{\mathbf{s}}, \vec{\mathbf{z}}, \vec{\mathbf{m}})}{q(\vec{\mathbf{s}}, \vec{\mathbf{z}}, \vec{\mathbf{m}}|\tau)} \right]. \tag{3.8}$$

Since log is a concave function, we can invoke Jensen's inequality to give us a lower bound on our objective:

$$\log p(\tau) = \log \mathbb{E}_{\vec{\mathbf{s}}, \vec{\mathbf{z}}, \vec{\mathbf{m}} \sim q} \left[ \frac{p(\tau, \vec{\mathbf{s}}, \vec{\mathbf{z}}, \vec{\mathbf{m}})}{q(\vec{\mathbf{s}}, \vec{\mathbf{z}}, \vec{\mathbf{m}}|\tau)} \right] \geq \mathbb{E}_{\vec{\mathbf{s}}, \vec{\mathbf{z}}, \vec{\mathbf{m}} \sim q} \left[ \log \frac{p(\tau, \vec{\mathbf{s}}, \vec{\mathbf{z}}, \vec{\mathbf{m}})}{q(\vec{\mathbf{s}}, \vec{\mathbf{z}}, \vec{\mathbf{m}}|\tau)} \right]. \tag{3.9}$$

Then, we need only expand both the joint distribution and our approximate posterior into their factorized forms and simplify

$$\begin{aligned}
&\mathbb{E}_{\vec{\mathbf{s}}, \vec{\mathbf{z}}, \vec{\mathbf{m}} \sim q} \left[ \log \frac{\prod_{t=1}^{T} p(o_t|o_{t-1}, a_{t-1}) p_\theta(a_t|\mathbf{s}_t) p_\theta(\mathbf{s}_t|o_t, \mathbf{s}_{t-1}, \mathbf{z}_t, \mathbf{m}_{t-1}) p_\theta(\mathbf{m}_t|\mathbf{s}_t) p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{m}_{t-1})}{\prod_{t=1}^{T} q_\theta(\mathbf{m}_t|\mathbf{m}_{<t}, \tau) q_\theta(\mathbf{z}_t|\vec{\mathbf{m}}_t, \tau) q_\theta(\mathbf{s}_t|\vec{\mathbf{z}}, \vec{\mathbf{m}}, \tau)} \right] \\
&= \sum_{t=1}^{T} \Bigg( \log p(o_t|o_{t-1}, a_{t-1}) + \mathbb{E}_{\mathbf{s}_t \sim q} \left[ \log p_\theta(a_t|\mathbf{s}_t) \right] - D_{\mathrm{KL}} \left( q_\theta(m_t|m_{<t}, \tau) \| p_\theta(m_t|s_t) \right) \\
&\quad - D_{\mathrm{KL}} \left( q_\theta(z_t|\vec{m}, \tau) \| p_\theta(z_t|z_{t-1}, m_{t-1}) \right) - D_{\mathrm{KL}} \left( q_\theta(s_t|\vec{z}, \vec{m}, \tau) \| p_\theta(s_t|o_t, s_{t-1}, z_t, m_{t-1}) \right) \Bigg)
\end{aligned} \tag{3.10}$$

to get an ELBO for $\log p(\tau)$. The loss component function $\mathcal{L}_{\mathrm{ELBO}}(\tau, \theta)$ is simply the negation of

this ELBO with the environmental dynamics term $\sum_{t=1}^{T} \log p(o_t|o_{t-1}, a_{t-1})$ removed:

$$\begin{aligned}\mathcal{L}_{\text{ELBO}}(\tau, \theta) = \sum_{t=1}^{T} \Big[ &\mathbb{E}_{\mathbf{s}_t \sim q_\theta} \left[ -\log p_\theta(a_t|\mathbf{s}_t) \right] + D_{\text{KL}} \left( q_\theta(m_t, m_{<t}|\tau) \| p_\theta(m_t|s_t) \right) \\ &+ D_{\text{KL}} \left( q_\theta(z_t|\vec{m}, \tau) \| p_\theta(z_t|z_{t-1}, m_{t-1}) \right) + D_{\text{KL}} \left( q_\theta(s_t|\vec{z}, \vec{m}, \tau) \| p_\theta(s_t|o_t, s_{t-1}, z_t, m_{t-1}) \right) \Big] \\ &\geq -log(p(\tau)) + \sum_{t=1}^{T} \log p(o_t|o_{t-1}, a_{t-1}).\end{aligned} \tag{3.11}$$

We can safely ignore the environmental dynamics term because it is unaffected by the parameters $\theta$ we optimize over. Following Kingma and Welling, we approximate the expectation with respect to our approximate posterior with one-sample Monte Carlo estimates.

**Abstraction Bonus** With just our ELBO objective it is possible for our model to learn the degenerate solution where all $\mathbf{m}_t$ take the value 1, resulting in a separate skill for each time step. Learning a space of skills with this degenerate segmentation would essentially be learning an embedding of the original action space and we would not be able to gain the benefits of temporally extended skills. Therefore, we add the following abstraction bonus term to our loss :

$$\mathcal{L}_{\text{A}}(\tau, \theta) = \sum_{t=1}^{T} \mathbb{E}_{\mathbf{m}_t \sim q_\theta} \left[ \mathbf{m}_t \right]. \tag{3.12}$$

Again, this expectation is approximated with a one-sample Monte Carlo estimate. This term represents the number of skills that trajectory $\tau$ is split into, and adding this quantity to our ELBO loss results in skills of longer length.

Our abstraction bonus differs from the compression loss used by LOVE [14] in that their loss represents the average number of bits encoded in the skill latents. We found that when applied to the Maze2d and Kitchen environments, the skills learned by LOVE were trivial in that they encoded no information. We discuss this further in the evaluation section.

Altogether, the loss that we optimize to learn our segmentation model is given by

$$\mathcal{L}(\tau, \theta) = \mathcal{L}_{\text{ELBO}}(\tau, \theta) + \lambda \mathcal{L}_{\text{A}}(\tau, \theta), \tag{3.13}$$

where $\lambda$ is a positive scalar that we refer to as the abstraction bonus coefficient. In practice, since scaling the different terms comprising $\mathcal{L}_{\text{ELBO}}$ can effect the representations learned [3, 12], scale factors for each of those terms are used as well.

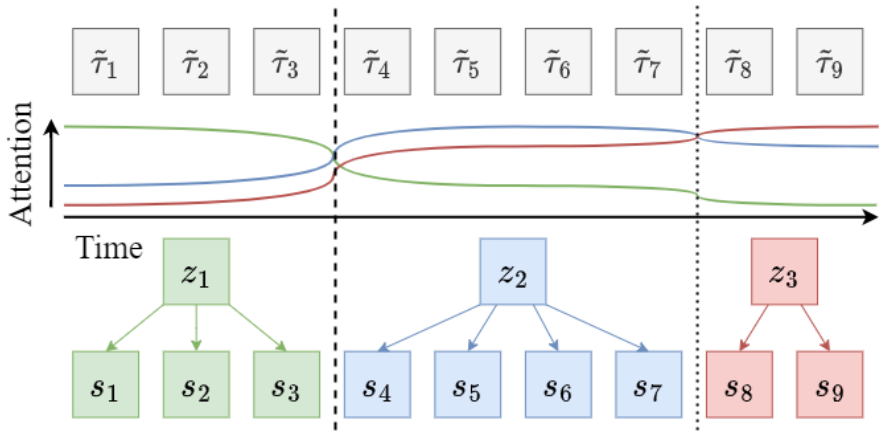


Figure 3.2: A depiction of our skill encoding scheme. The dotted vertical lines are segmentation variables $\mathbf{m}_t$ that take values greater than 0. The length of the dots represents the continuous value taken by these $\mathbf{m}_t$, longer dots implying a higher value. The $\tilde{\tau}_t$ represent our original sequential data, $o_t, a_t$ that have been preprocessed before the skill encoding step. Each $\mathbf{z}_i$ is encoded from the $\tilde{\tau}_t$ with its corresponding attention mask, depicted in the same color. These $\mathbf{z}_i$ are then used to condition the corresponding low-level latent variables $\mathbf{s}_t$.

### 3.3.2 Continuous Relaxation and Architecture

Here we present our transformer-based skill segmentation model architecture and the continuous relaxation that we use to train the model to generate discrete segmentations.

Our proposed architecture allows for the use of the continuous Concrete random variable [20], in contrast to the straight-through variant [13] that is employed by many theoretically similar methods [14, 15, 23, 35]. We find that our approach learns semantically meaningful trajectory segmentations without enforcing a prior on the length of the learned skills, as many previously proposed methods call for [15, 17, 23, 31].

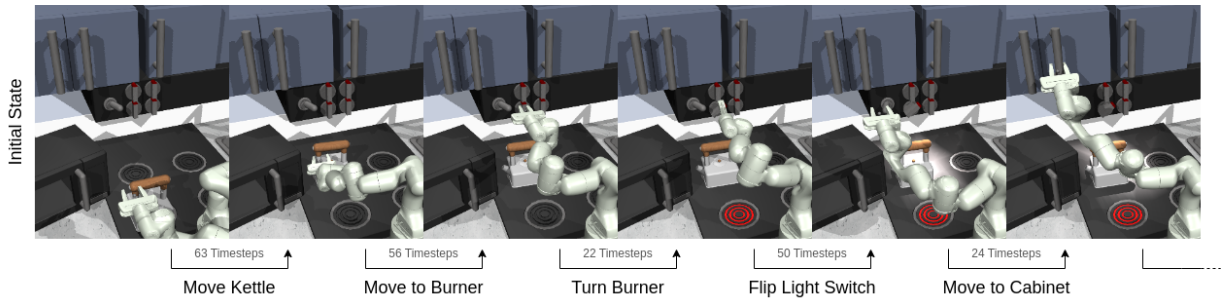


Figure 3.3: Frames taken from segmentation points discovered by our segmentation model. These frames occur at semantically meaningful points in the trajectory and thus can be easily labeled with human-interpretable skills. Many of these frames also occur at points where trajectories deviate from one another as they accomplish different subsets of kitchen tasks.

We begin by describing how the posterior distribution over a skill latent variable $\mathbf{z}_t$ is inferred. First, our model computes an unnormalized weight vector $\vec{w}^t \in \mathbb{R}^T$ where $T$ is the length of the trajectory $\tau$. The skill distribution input vector $\tilde{z}_t$ is then computed using an attention mechanism over (embedded) trajectories $\tilde{\tau} \in \mathbb{R}^{T \times d}$, where $d$ is a hyperparameter. More specifically,

$$\tilde{z}_t = \sum_{i=1}^{T} \alpha_i \tilde{\tau}_t. \tag{3.14}$$

where $\alpha_i$ are normalized weights computed as

$$\alpha_i = \frac{w_i^t}{\sum_{j=1}^{T} w_j^t}. \tag{3.15}$$

The manner in which we compute our normalized weights differs from the softmax normalization used by a standard attention mechanism.

Note that for any two timesteps $i$ and $j$, if identical weight vectors are computed for these two timesteps, then the skill distribution input vectors computed for these two timesteps will also be identical, *i.e.,* $\vec{w}^i = \vec{w}^j \implies \tilde{z}_i = \tilde{z}_j$.This is the case when our model infers that $i$ and $j$ belong to the same abstract timestep. The parameters for the posterior distribution over the skill latent variable $\mathbf{z}_t$ are then deterministically computed from $\tilde{z}_t$ so $\tilde{z}_i = \tilde{z}_j$ implies the distributions for $\mathbf{z}_i$ and $\mathbf{z}_j$ are the same. We then sample $\mathbf{z}_t$ from the posterior distribution via the reparameterization trick [16].

We compute attention weights with a special function $w_k^t = f(\vec{m}, t, k)$ such that if there are no segmentation points between timesteps $i$ and $j$, then $\tilde{z}_i = \tilde{z}_j$. We use the following function,

$$f(M, t, k) = \begin{cases} \prod_{i=1}^{k-t}(1 - m_{t+i-1}) & k \geq t \\ \prod_{i=1}^{t-k}(1 - m_{t-i}) & k < t \end{cases}. \tag{3.16}$$

To make the above process differentiable, we adopt a continuous relaxation wherein the discrete segmentation variables $\mathbf{m}_t$ are replaced with samples from a Concrete distribution such that $\mathbf{m}_t \in [0, 1]$. This gives us a continuous way to merge two distinct adjacent skills into one skill and compute low-variance reparameterization gradients for our model parameters. A visualization of the construction of these unnormalized attention weights can be seen in Fig. 3.2. Maddison et al. 2016 empirically showed that, with a sufficiently low-temperature parameter, latent variable models using Concrete random variables tend to learn solutions where the probability mass of the Concrete distribution concentrates at the vertices of the simplex rather than in the interior (that is, the model tends to sample $m_t \in \{0, 1\}$). We also observe this phenomenon

in our experiments. This means that our continuous relaxation is suitable for learning a discrete trajectory segmentation model.

While prior work primarily uses RNNs [14, 15, 17], and specifically variants of the RSSM introduced by [11], we use the transformer equivalent of the RSSM introduced in TECO [33] for both our high-level skill and low-level state dynamics. Our full implementation can be found at `https://github.com/Thomasw219/thesis_repo`

## 3.4 Evaluation

We demonstrate our approach on the Maze2d and Kitchen tasks from the D4RL offline RL benchmark [8]. We show the ability of our segmentation learning model to robustly discover useful data segmentation schemes and compare the segmentations and reconstructions learned by our approach to those learned by prior work, as well as the empirical distributions over skill length learned by our model in both environments.

### 3.4.1 Learned Segmentations

In both the Maze2d and the Kitchen datasets, our segmentation learning model learns semantically meaningful segmentation points. An example of segmentation points for one of the Kitchen trajectories can be seen in Fig. 3.3. In this environment, the segmentation points we learn are easily interpretable and match with our intuitive definition of a skill in that they correspond to subtasks such as moving the kettle or flipping the light switch. In the Maze2d environments, one of the segmentation schemes that our model identifies is to segment trajectories at timesteps where the underlying PD controller used to generate the data changes setpoints.

The particular segmentation that we learn for a given dataset depends on the abstraction bonus coefficient. However, we found that a wide range of abstraction bonus coefficients led to the same skill segmentations, indicating that our approach isn't particularly sensitive to this parameter. In Fig. 3.4, we show how the average number of timesteps per segment in the dataset changes over the learning process for our segmentation learning model and a version of VTA that has been modified to use an abstraction bonus term, for various abstraction bonus coefficients. Since the same segmentation of a trajectory will imply the same average number of timesteps per segment, we use this metric as a proxy for assessing the repeatability with which a particular segmentation scheme is learned. In our experiments, most of the models that fell into the higher ratio mode of Fig. 3.4 (top) did in fact learn the same, aforementioned scheme of segmenting at the points where the underlying PD controller changed setpoints.

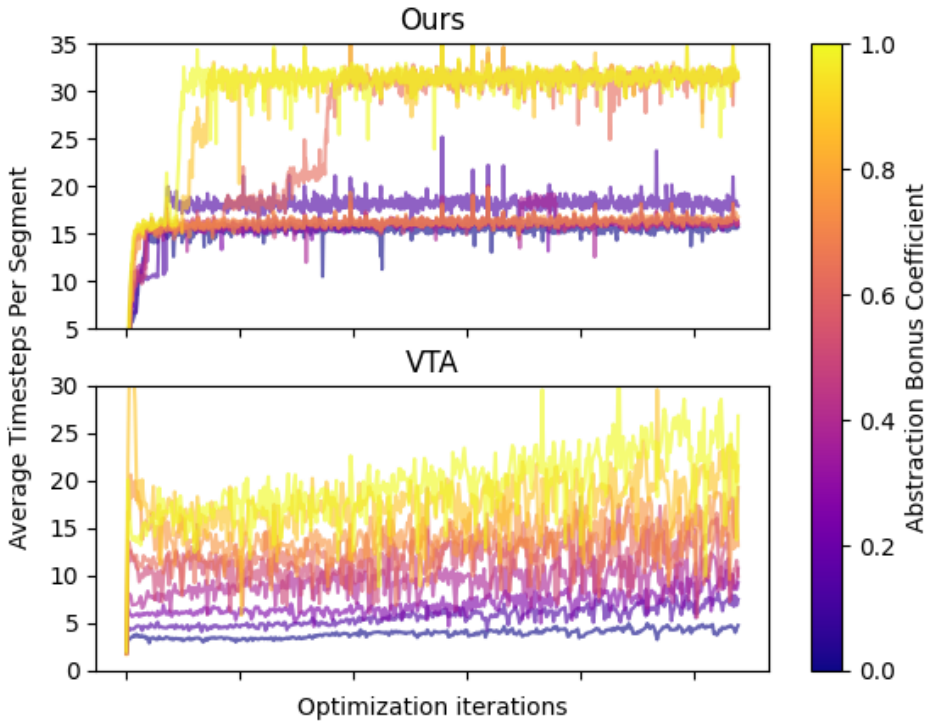


Figure 3.4: Average number of timesteps per trajectory segment vs. epochs for different values of the abstraction bonus coefficient. Our method (top) finds similar segmentation schemes for many different coefficient values, resulting in a bimodal distribution. VTA (bottom), which has been modified to use an abstraction bonus rather than a prior over skill lengths, learns segmentations that are much more sensitive to the abstraction bonus coefficient.

There is some degree of volatility in our method, with different random model initializations converging to different segmentation schemes, given the same abstraction bonus coefficient. Therefore, when selecting a segmentation model to use for trajectory segmentation, we choose an abstraction bonus coefficient that results in reproducible segmentation schemes.

In Fig. 3.5 we compare the segmentations learned by our model to those learned by VTA when given an informative prior [15], LOVE [14], and Directed-Info GAIL [26]. We compare against these prior works in particular since LOVE and Directed-Info GAIL do not use a prior over skill length, similar to our method, and VTA, like our method, uses continuous latent variables to represent the learned skills, whereas LOVE and Directed-Info GAIL encode skills into discrete latent variables. Our method perfectly recovers the locations where the underlying PD controller changes setpoints, while the segmentation learned by VTA is often off by 1 or 2 timesteps. VTA also exhibits other, less frequent, inconsistencies such as false positives and false negatives if we take the region around the changepoints of the underlying PD controller to be ground truth. We also often see consecutive segmentations with VTA, likely due to its independently factorized segmentation posterior. LOVE learns a trivial segmentation of the signal into segments of length 3, its minimum allowable skill length, whereas our model is able to recover meaningful, temporally abstract segmentations. Because LOVE's compression loss is a weighted sum of its segmentation points scaled by the amount of information encoded in the skill latents, it can learn an assignment of skill latents where no information is communicated via the skills and achieve a low compression loss while not performing any meaningful temporal abstraction. Directed-Info GAIL learns a degenerate solution where a single skill is used to describe the entire subsequence. Its reconstruction of the actions taken is generally accurate except for the points where the controller changes goals, likely due to its segmentation's degenerate nature.

### 3.4.2 Skill Length Distributions

Many prior works in the field of segmentation learning such as VTA [15], CompILE [17], TVI [23], and SKID RAW [31] regularize the distribution over the length of the skills they learn to a fixed prior distribution. This prior distribution is typically unimodal, as with the Poisson prior used by CompILE or the Gaussian distribution used by SKID RAW. The assumption that the length of skills follow a unimodal distribution does not hold for many tasks of interest. Take the Maze2d and Kitchen tasks on which we evaluate our method as an example: Figure 3.6 shows the distributions learned by our skill learning approach. Note that even in these relatively simple benchmarks, the distribution over skill lengths can be very wide and multimodal. We can expect that in more realistic tasks the distribution over skill lengths could be even wider, e.g.

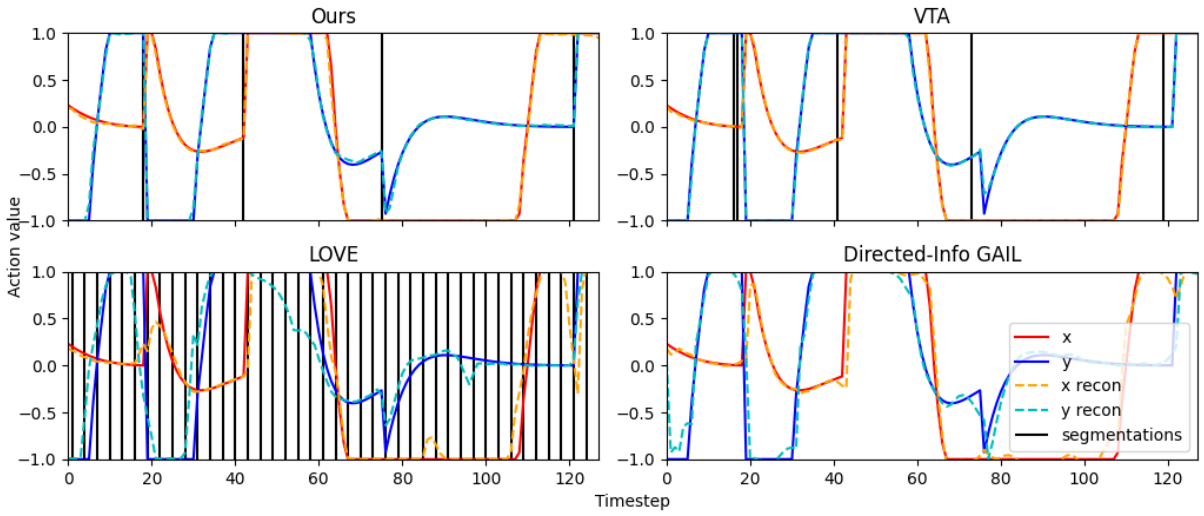


Figure 3.5: A comparison of action reconstructions and segmentations learned by our model and a number of baselines.

baking bread where proofing and baking can take an order of magnitude longer than ingredient mixing or kneading. Other manners of regularizing skill length, such as those used by TVI and VTA are not as simple as a unimodal distribution, but nonetheless softly restrict the possible decompositions of trajectories into skills that their models can learn. Our method imposes no such restriction on the classes of possible distributions learned by the model, allowing us to learn whatever decomposition of trajectories best suits the data.

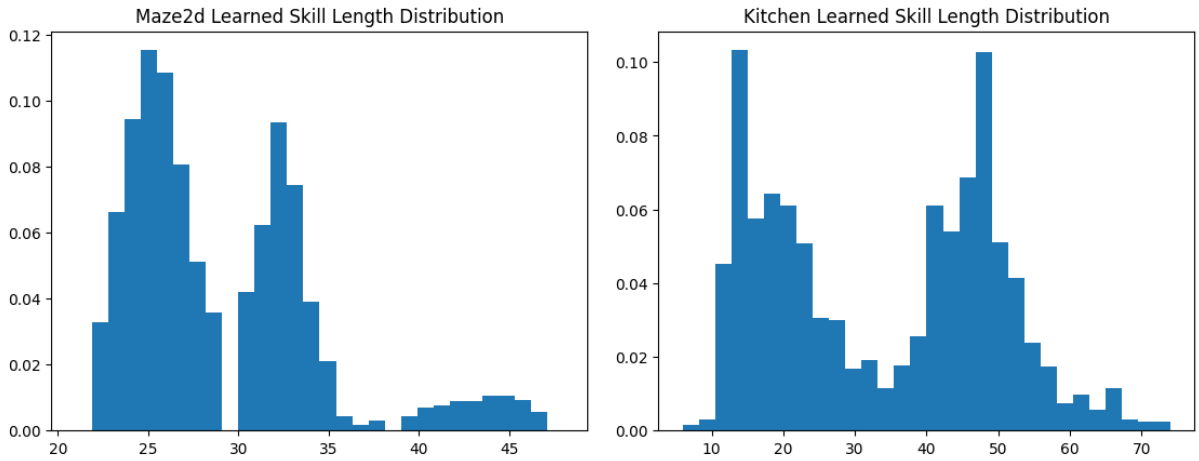


Figure 3.6: The empirical distributions over skill length learned by our approach in the Maze2d and Kitchen environments.

# Chapter 4

# Learning and Planning Over Variable Length Skills

## 4.1 Previous Literature

In sequential decision-making problems, the ability of an agent to execute temporally extended skills can facilitate much more efficient credit assignment over long periods of time or, when combined with a skill-conditioned dynamics model, greatly increase an agent's effective planning horizon. Because of this, the discovery of skills has long been an area of interest in the field of RL.

At a high level, we can divide skill-learning approaches into online and offline. Online skill learning algorithms construct a collection of skills through interaction with the environment but typically require prohibitively large amounts of data to discover useful skills [6, 24]. On the other end of the spectrum, offline skill learning algorithms seek to extract skills from demonstrations and do not require potentially dangerous online exploration in the environment to do so [14, 23, 27]. We primarily focus on methods of the offline variety; in particular, we go into detail on two recently developed algorithms, OPAL and OPOSM that represent model-free and model-based approaches to skill-based offline RL, respectively.

OPAL [2] uses a conditional autoencoding approach with a KL-Divergence constraint on the learned latent variables to get its skills and skill-conditioned policies. Their approach bears a resemblance to the objective for a conditional VAE where the prior is conditioned on the skill's initial state, and the reconstructions of the actions are conditioned on the state at that timestep. After the skill-conditioned policies are learned, a dataset of initial states, skills, terminal states, and cumulative rewards is given to any offline RL algorithm to learn a policy over skills.

OPOSM [7] uses a variational objective to learn skills in the style of a conditional VAE, sim-

ilar to OPAL. However, OPOSM also learns an abstract dynamics model to infer the distribution over terminal states induced by a learned skill. This facilitates planning over skills online, given a new reward function. The formulation used by OPOSM is similar to that learned by SkiMo [27], which also uses a variational objective to learn a skill-conditioned policy and termination state distribution. However, OPOSM also uses a modified expectation maximization (EM) procedure to ensure that the learned skills respect the causal structure of the environment.

OPAL, OPOSM, SKiMo, and many other prior works sample fixed-length segments of trajectories uniformly at random from the given dataset. Even under the condition that the fixed-length skill assumption holds, given most reasonable skill representations and a fixed length greater than one, it is unlikely that a randomly sampled segment from a trajectory will align with the beginning and end of an executed skill.

While OPOSM and OPAL both use fixed-length skills, there do exist a few variable-length model-free skill learning approaches that have been demonstrated on smaller problems [14, 15, 17, 26, 31]. We compared the most relevant among these as alternative segmentation methods to our own in Chapter 3. Other variable-length model-free skill learning methods either require a prior over skill length [15, 17, 31], do not scale to the more complex and temporally abstract problems that we evaluate our algorithm on [14, 26], or both. This necessitated the development of our own segmentation learning approach, described and evaluated further in Chapter 3.

## 4.2 Preliminaries

Our skill learning approach extends the algorithm introduced by OPOSM [7]. After learning a space of skills, we plan over our learned skills with the cross entropy method (CEM) [22] in order to maximize a given reward function online. We provide background on both of these algorithms in this section.

In this chapter, we follow prior skill learning work [2, 7] in assuming full observability. Therefore $\mathbf{s}_t$ no longer represents an unobserved latent random variable but rather the full environment state at time $t$.

### 4.2.1 Learning Fixed-Length Skills from Offline Data

OPOSM [7] uses a modified conditional VAE objective to jointly learn a space of skills $\mathcal{Z}$, a skill-conditioned policy $\pi_\theta(a_t|s_t, z)$, a state-conditioned prior over the skill space $p_\omega(z|s_1)$, and the temporally abstract dynamics induced by those skills $p_\psi(s_H|s_1, z)$.

Assuming skills to be latent random variables that condition both the policy and the induced

dynamics results in the following variational objective:

$$\log p(s_H, \vec{a}|\vec{s}) \geq \mathbb{E}_{\mathbf{z}\sim q_\phi(z|\tau_H)} \left[ \left( \sum_{t=1}^{H} \log \pi_\theta(a_t|s_t, \mathbf{z}) \right) + \log p_\psi(s_H|s_1, \mathbf{z}) \right] - D_{KL}(q_\phi(z|\tau_H) \| p_\omega(z|s_1)) \tag{4.1}$$

where $\vec{a} = a_{1:H}, \vec{s} = s_{1:H}, \tau_H = (\vec{a}, \vec{s})$ and $q_\theta(z|\tau_H)$ is a skill encoder that outputs the posterior distribution of a particular skill. Here, $H$ is a fixed hyperparameter denoting the length of the skills learned by OPOSM. Freed et al. show that naively optimizing this lower bound does not respect the actual causal structure of the environment which can result in the overestimation of the agent's influence on the environment. For example, a state transition that was not a result of the agent's action but instead due to uncontrollable stochasticity in the environment could be wrongly attributed as the result of a skill executed by the agent.

To remedy this, Freed et al. propose a modified EM procedure that minimizes the KL divergence between the approximate skill posterior and the true posterior over skills given by

$$p(z|\tau) = \frac{1}{\eta} \left( \prod_{t=1}^{H} \log \pi_\theta(a_t|s_t, z) \right) p_\theta(z|s_1) \tag{4.2}$$

where $\eta$ is a normalization constant. In the expectation step of the procedure, the parameters $\phi$ are updated to minimize the KL divergence between the approximate and true skill posteriors. In the maximization step $\theta, \psi$ and $\omega$ are updated to maximize the lower bound in Equation 4.1.

### 4.2.2 Planning with the Cross-Entropy Method

The cross-entropy method (CEM) [22] is a stochastic, iterative algorithm that can be used for rare event simulation or optimization. In its simplist form, it is parameterized by a population size $N$, a keep fraction $\rho$, and a sample evaluation function $L(\cdot)$. OPOSM [7] uses CEM as a planner by sampling possible skill trajectories and iteratively refining the distribution over trajectories to maximize a given reward function.

Rather than planning in the raw skill space, OPOSM uses CEM to plan in the space of standardized deviations from the mean skill given by the prior. More specifically, let $\vec{\epsilon} = \Big[\epsilon_1, \ldots, \epsilon_L\Big]$, where $\epsilon_i \in \mathbb{R}^{dz}$, be a particular trajectory of skill deviations, and $s_1$ be our initial state. Then, the trajectory of skills executed $\vec{z}$ and terminal states visited $\vec{s}$ corresponding to

$\vec{\epsilon}$ is computed by iteratively using the learned skill prior and abstract dynamics as follows:

$$\mu_i = \mathbb{E}_{\mathbf{z}\sim p_\omega(z|s_i))}[\mathbf{z}], \sigma_i^2 = \mathrm{Var}_{\mathbf{z}\sim p_\omega(z|s_i))}(\mathbf{z}) \tag{4.3}$$

$$z_i = \mu_i + \sigma_i \epsilon_i \tag{4.4}$$

$$s_{i+1} = \mathbb{E}_{\mathbf{s}\sim p_\psi(s_{i+1}|s_i,z_i))}[\mathbf{s}]. \tag{4.5}$$

Given a reward function $R(\cdot)$, we can write the sample evaluation function $S(\vec{\epsilon})$ used by OPOSM as

$$S(\vec{\epsilon}) = \max_{i\in\{1,\ldots,L\}} R(s_i). \tag{4.6}$$

CEM begins with an initial distribution over $\vec{\epsilon}$. Because OPOSM learns a continuous skill space, the family of distributions over which CEM optimizes is chosen to be joint Gaussian distributions with diagonal covariances. First, $N$ samples, $\vec{\boldsymbol{\epsilon}_1}, \ldots, \vec{\boldsymbol{\epsilon}_N}$ are drawn i.i.d from the initial distribution, which is parameterized by mean $\mu_1^\epsilon$ and covariance $\Sigma_1^\epsilon$. Then, a value $\gamma$ delineating the top performing $\rho$ fraction of trajectories among the samples is computed as

$$\gamma = \max\left\{\gamma' : \frac{1}{N}\sum_{i=1}^{N} I_{\{S(\vec{\boldsymbol{\epsilon}_i})>\gamma'\}} \geq \rho\right\}, \tag{4.7}$$

where $I_{\{\cdot\}}$ is the indicator function. With $\gamma$ computed, new parameters for the distribution over $\vec{\epsilon}$ are computed by finding the highest likelihood mean and variance for the samples $\{\vec{\boldsymbol{\epsilon}_i} : S(\boldsymbol{\epsilon}_i) > \gamma\}$. In the case of the Gaussian distribution, this is simply the sample mean and sample variance. This process repeats until some stopping criterion is met; the particular criterion OPOSM uses is simply reaching a fixed number of iterations. The skill $z_1 = \mu_1 + \sigma_1\mathbb{E}[\epsilon_1]$ is then executed until termination, after which CEM is used to replan in the style of model predictive control (MPC).

## 4.3 Method

Our skill-learning process differs from OPOSM in a couple of important ways. First, while OPOSM samples fixed-length segments uniformly from its dataset, our approach uses a learned segmentation model from Chapter 3 to split trajectories from the dataset into variable-length segments. Next, we learn a similar skill model to OPOSM but with an additional state-dependent skill termination condition that uses the segmentation model's learned skill termination points as supervision. Our particular optimization approach also differs from the modified EM approach used by OPOSM in that we utilize the stop gradient operator to jointly compute gradients for the posterior and the rest of the model while retaining the property that our learned skills will not

overestimate the agent's influence on the environment.

### 4.3.1 Training Data Preprocessing

Given a trained segmentation model, we take the original offline learning dataset and prepare it for variable-length skill learning. We first apply the segmentation model on each trajectory in the dataset to get skill termination points. Since the segmentation model may have been trained on shorter subsequences of trajectories, to get these termination points we apply the segmentation model on subsequences of the size that it was trained on, taken in a sliding-window fashion from the full-length trajectory. It is possible that a particular timestep will be labeled a termination event in one subsequence and not in another, so we compute the proportion of the subsequences in which a given timestep was labeled a termination event and threshold the proportions to get our final termination points.

We then split each trajectory into contiguous subsequences at these termination points. To facilitate batch training of variable length sequences, we pad the trajectories to the length of the maximum subsequence length and generate masks to indicate which timesteps the posterior should take as input. For practicality, we choose an upper limit to the length of the skills so an outlier segment of extremely long length will not require us to pad and mask all other segments to that same length. We also choose a lower limit to save computation. These maximums and minimums are manually chosen after looking at the distribution of learned skills, visualized in Figure 3.6. We ended up using a maximum length of 50 in the Maze2d environment, 75 in the Kitchen environment, and the same minimum length of 5 in both environments.

### 4.3.2 Learning Variable-Length Skills from Offline Data

With the dataset of variable-length segments prepared, we move on to learning our skill model, the architecture of which can be seen in Figure 4.1. Our skill model consists of the approximate posterior $q_\phi(z|\tau)$, a low level policy $\pi_\phi(a_t|s_t, z)$, a termination decoder $p_\phi(m_t|s_t, z_t)$, the abstract dynamics induced by the skills $p_\phi(s_{L(\tau)}|s_1, z)$, and a state-conditioned prior $p_\phi(z|s_1)$. We denote the length of trajectory segment $\tau$ with $L(\tau)$. Although a skill encoder and termination conditions were learned as part of the segmentation model, we re-learn these as part of the skill model learning process because the skill representation may change. To learn our skill model, we maximize an ELBO given by

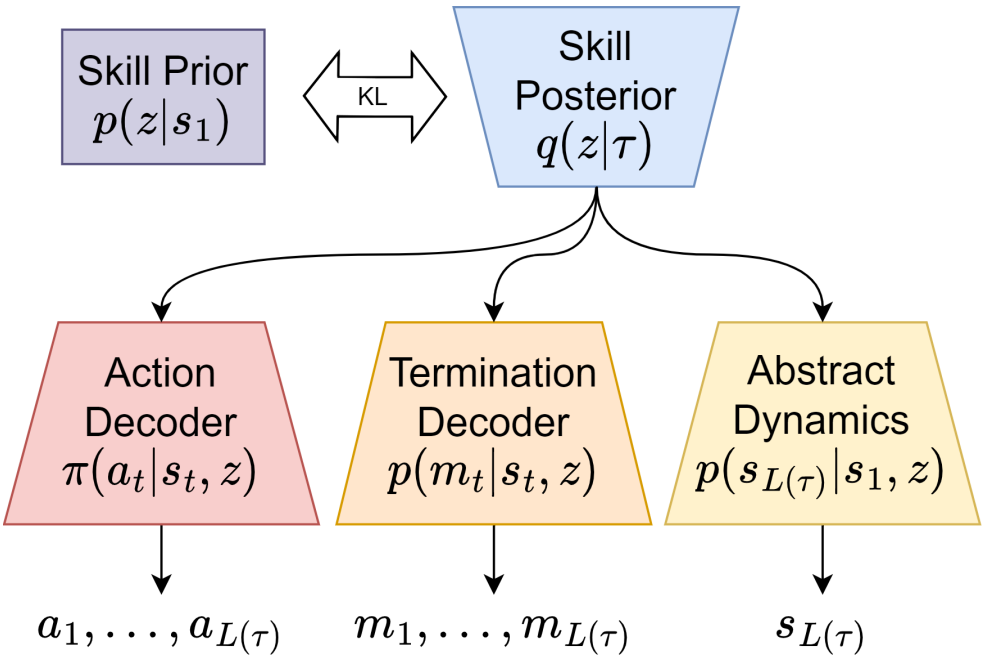


Figure 4.1: The various components of our skill model. Here $L(\tau)$ is the length of the particular trajectory segment $\tau$.

$$\mathcal{L}_{SM}(\phi, \tau) = \mathbb{E}_{\mathbf{z}\sim q(\tau|z)} \left[ \sum_{t=1}^{L(\tau)} (\log \pi_\phi(a_t|s_t, \mathbf{z}) + \log p_\phi(m_t|s_t, \lfloor \mathbf{z} \rfloor)) + \log p_\phi(s_{L(\tau)}|s_1, \lfloor \mathbf{z} \rfloor) \right] - D_{KL}(q_\phi(z|\tau)||p_\phi(z|s)), \tag{4.8}$$

where $\lfloor \rfloor$ denotes a stop grad operator, through which gradients are not backpropagated. We use the stop grad operator to accomplish a similar purpose as the modified EM algorithm proposed in OPOSM [7]. Without the stop grad operator, the model is encouraged to encode the terminal state $s_{L(\tau)}$ directly into $\mathbf{z}$, resulting in an overly confident model. Using stop grad operators removes this incentive and captures the fact that $\mathbf{z}$ influences state transitions only indirectly through its influence on actions.

Once the skill model is learned, we use the temporally abstract dynamics model $p_\phi(s'|s, z)$ to iteratively re-plan sequences of skills, in a similar manner to OPOSM. The primary difference being that at each timestep during the execution of skills, termination variables $\mathbf{m}_t$ are sampled in addition to actions. The current skill is terminated once $\mathbf{m}_t = 1$. Skill sequences are re-planned upon termination of each skill in the style of MPC.

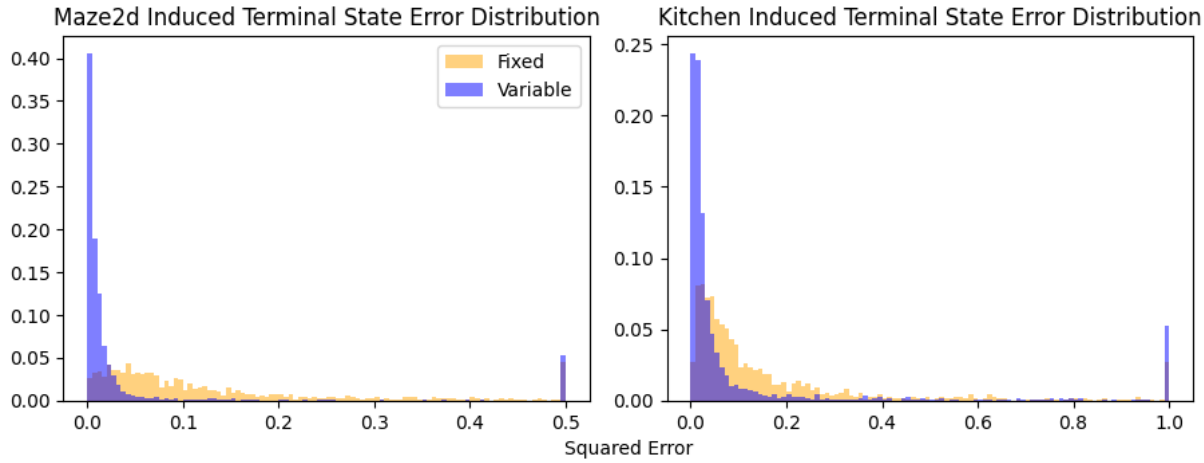


Figure 4.2: The empirical distributions of the error between the induced terminal state from rolling out a learned skill-conditioned policy and the estimated terminal state from skill-conditioned abstract dynamics for both the Maze2d and Kitchen environments. Outliers beyond the range of the graph are clamped to within the range (right-hand side of both graphs).

## 4.4 Evaluation

After learning our set of variable-length skills, we assess the quality of individual variable-length skills compared to fixed-length skills and compare the efficacy of planning over our learned skills with our abstract dynamics to other offline RL approaches. We additionally provide a comparison with variable-length skills learned with a segmentation from VTA rather than our proposed segmentation learning method.

### 4.4.1 Quality of Learned Skills

To test our hypothesis that variable-length skills enable the learning of more accurate abstract dynamics models, we compare our learned skills to an identically trained fixed-length skill model that does not use a state-dependent termination condition. For all experiments, the fixed length we choose is the average length of our learned segmentations.

We evaluate the quality of individual skills by sampling a skill, predicting the skill's terminal state with the learned abstract dynamics, and then running the skill-conditioned policy from the initial state until termination in the real environment. We visualize the empirical distribution of the error between the predicted and ground-truth state transitions in Fig. 4.2. We observe that more mass is concentrated at lower-error values for variable-length skills than for fixed-length skills, indicating that variable-length skills result in more accurate temporally abstract dynamics models.

| Task | Ours | OPOSM | LLP | CQL+OPAL | BC | EMaQ | CQL | PLAS | COMBO |
|---|---|---|---|---|---|---|---|---|---|
| Maze2d Medium | $\mathbf{99.9 \pm 0.1}$ | $\mathbf{98.6 \pm 1.6}$ | $95.3 \pm 2.4$ | - | - | - | - | - | - |
| Maze2d Large | $\mathbf{94.6 \pm 8.9}$ | $80.8 \pm 4.7$ | $77.3 \pm 4.7$ | - | - | - | - | - | - |
| Kitchen Partial | $\mathbf{70.2 \pm 10.9}$ | $57.2 \pm 5.8$ | $39.0 \pm 4.6$ | $52.0 \pm 2.6$ | 33.8 | $\mathbf{74.6 \pm 0.6}$ | $50.1 \pm 1.0$ | 45 | $24.3 \pm 2.0$ |
| Kitchen Mixed | $\mathbf{69.6 \pm 2.5}$ | $66.3 \pm 2.7$ | $46.0 \pm 3.8$ | $32.5 \pm 7.6$ | 47.5 | $\mathbf{70.8 \pm 2.3}$ | $52.4 \pm 2.5$ | 40 | $2.3 \pm 0.8$ |

Table 4.1: Maze2d and Kitchen D4RL task scores with the default goal specified by the original task. We evaluate Maze2d environments by the percent of episodes where the goal is reached and the Kitchen environments with normalized scores.

| Task | Ours | OPOSM | LLP |
|---|---|---|---|
| Maze2d Medium | $\mathbf{98.2 \pm 1.1}$ | $94.6 \pm 2.6$ | $94.7 \pm 2.5$ |
| Maze2d Large | $\mathbf{96.3 \pm 2.2}$ | $78.2 \pm 3.0$ | $91.7 \pm 3.1$ |
| Kitchen Partial | $\mathbf{67.1 \pm 2.1}$ | $64.0 \pm 3.0$ | $43.1 \pm 4.5$ |
| Kitchen Mixed | $\mathbf{64.3 \pm 5.6}$ | $\mathbf{62.2 \pm 2.5}$ | $35.5 \pm 4.5$ |

Table 4.2: Scores for the Maze2d and Kitchen D4RL tasks with randomized goals.

### 4.4.2 Skill Planning on Downstream Tasks

To verify that higher-quality skills lead to higher planning performance on downstream tasks, we compare our method's performance on D4RL [8] tasks with fixed-length skills (OPOSM), in addition to other offline RL algorithms. We evaluate our method on the default goals given by the D4RL datasets and on goals sampled uniformly from the space of possible goals in a given environment.

We compare against both skill-based and standard environment timescale offline RL algorithms. Low-Level Planning (LLP) is a model-based approach that uses provided offline RL data to train a dynamics model conditioned on single actions. This dynamics model is then used to optimize over action sequences with CEM. LLP can be thought of as a special case of OPOSM [7] where the skill length hyperparameter $H$ is set to 1. We also compare to CQL+OPAL, which uses skills learned by OPAL [2] in combination with Conservative Q-Learning (CQL) [18], in addition to Behavioral Cloning (BC) [21], Expected Max Q-Learning (EMaQ) [9], CQL with base level actions, Policy in Latent Action Space (PLAS) [36], and Conservative Offline Model-Based Policy Optimization (COMBO) [34].

Planning over our variable-length skills performs at least as well as OPOSM in every task and often performs significantly better. We also find that our approach is competitive with state-of-the-art offline RL algorithms in the Kitchen tasks while naturally generalizing to random goals, a property that most other offline RL algorithms do not natively support. We were unable to reproduce the performance figures for CQL+OPAL reported by [2]. No code for OPAL is publically available; the numbers we report were generated using partially complete code provided by the authors.

| Task | Ours | OPOSM | VTA |
|---|---|---|---|
| Medium Default | $\mathbf{99.9 \pm 0.1}$ | $\mathbf{98.6 \pm 1.6}$ | $80.6 \pm 15.9$ |
| Medium Random | $\mathbf{98.2 \pm 1.1}$ | $94.6 \pm 2.6$ | $82.3 \pm 8.9$ |

Table 4.3: Scores for the Maze2d Medium environment for variable length skills with segmentation points learned by our model, fixed length skills (OPOSM), and variable length skills learned by VTA with an informative prior. Results for both the default goal specified by the task and randomly sampled goals.

Since VTA is the only alternative segmentation learning algorithm that didn't learn a degenerate solution, we only continue our pipeline and learn a skill model for the segmentation output by VTA. We show the results of planning over skills learned from VTA segmentations in Table 4.3. Note that not only do skills learned with VTA-learned segmentations perform worse than those learned from our segmentation model, but they also perform worse than fixed length skills. This implies that not just any variable length segmentation will improve downstream task performance and that poor or inconsistent segmentations can actually lead to performance worse than the fixed length skill assumption.

All of the results from D4RL task experiments performed by us in this section were averaged over 5 random seeds and displayed with a 95% confidence interval.

# Chapter 5

# Conclusion and Future Work

We have presented a method for offline RL using temporal abstraction. Our approach is novel in that it is a model-based method that learns variable-length skills with state-dependent termination conditions, allowing agents to dynamically adjust the level of temporal abstraction at which they plan. We have shown that our approach enables agents to learn semantically meaningful skills, and more accurate temporally abstract dynamics models, compared to a fixed-length skill-learning approach. Furthermore, we show that our approach outperforms fixed-length skills on downstream tasks, and is competitive with state-of-the-art offline RL algorithms.

The approach we adopted in this work is primarily a result of the desire to stay close to prior work for practicality's sake. We ultimately would like to *jointly* learn a segmentation, policy, termination condition, and temporally abstract dynamics, in place of the segmentation and skill learning phases presented here. In addition, our work relies on the assumption that a task can be completed by maximizing a state-based reward function evaluated only at the terminal states of our skills. We would like to extend our approach to generate intermediate states and actions in a manner that preserves temporally abstract state transitions in order to make the learned skill model more generally applicable.

More high-level extensions to our work focus on expanding our approach to include multi-level hierarchies. As humans, we are capable of planning on many levels of abstraction simultaneously and frequently compose plans to form meta-level plans. It may be possible to further enhance agents' abilities to plan flexibly at various levels of temporal abstraction by including multiple levels of abstraction in a single model and allowing agents to determine which levels are most suitable for a given situation. Additionally, we are interested in investigating methods to help better align the skill-learning objective with the task objective, as currently skills are learned in a purely task-agnostic manner. Intuitively, we expect that an agent may be able to achieve higher task performance by allowing its skills to specialize to particular tasks.

## Acknowledgments

I want to express my gratitude to everyone who made my time at Carnegie Mellon so rewarding. This thesis would not have been possible without my friends and colleagues who helped me throughout the research process, the new friends I made at CMU who brightened my days and give balance to my life, my old friends who continue to provide me their support whenever it is needed, and my family who have worked hard to make all of this possible.

Thanks to Ben and Khush for all their help on the project that turned into this thesis. Ben provided me with the ideas that began my journey in this research direction and gave me an excellent foundation to build on with his prior work in the area. From our conversations and the time we spent working together, I have learned so much about reinforcement learning, communication, and research in general. Khush and I worked side by side for a large portion of this project, including an earlier line of inquiry that ended up being a dead end. However, I would not have arrived at the ideas presented in this thesis without the lessons we learned from that failed attempt and the insightful conversations we had along the way.

To all the friends I made at CMU, thanks for making my time at CMU not just an educational experience but an enjoyable one as well. Thanks to the Men's Club Volleyball Team, the Women's Varsity Volleyball Team, and the Women's Club Volleyball Team for giving me a reprieve from the work whenever I needed it and allowing me to work towards my athletic goals alongside my academic ones. I would also like to thank Abby, Delaney, Eric, Rob, and Sid for so graciously housing me for the final couple months I spent in Pittsburgh

Thanks to all my old friends who have continued to support me. In particular, thanks to Amani, Dishan, Eric, Hyujin, James, Kelvin, and Sherry for all the ways they have helped me over the years.

Finally, thanks to my parents for fostering my interest in academics and supporting me in achieving my dream of becoming the type of person I am today.

# Bibliography


[1] Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299*, 2018. 1

[2] Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. Opal: Offline primitive discovery for accelerating offline reinforcement learning. In *International Conference on Learning Representations*, 2020. 1, 2.1.3, 4.1, 4.2, 4.4.2, 4.4.2

[3] Alexander A. Alemi, Ben Poole, Ian S. Fischer, Joshua V. Dillon, Rif A. Saurous, and Kevin P. Murphy. Fixing a broken elbo. In *International Conference on Machine Learning*, 2017. URL `https://api.semanticscholar.org/CorpusID:3289726`. 3.3.1

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014. URL `https://arxiv.org/abs/1409.0473`. 3.2.1, 3.3

[5] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. URL `https://arxiv.org/abs/1308.3432`. 3.1, 3.2.2

[6] 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*, 2018. 1, 4.1

[7] Benjamin Freed, Siddarth Venkatraman, Guillaume Adrien Sartoretti, Jeff Schneider, and Howie Choset. Learning temporally AbstractWorld models without online experimentation. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10338–10356. PMLR, 23–29 Jul 2023. URL `https://proceedings.mlr.press/v202/freed23a.html`. 1, 1, 2.1.3, 4.1, 4.2, 4.2.1, 4.2.1, 4.2.2, 4.3.2, 4.4.2

[8] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2021. 3.4, 4.4.2

[9] Seyed Kamyar Seyed Ghasemipour, Dale Schuurmans, and Shixiang Shane Gu. Emaq: Expected-max q-learning operator for simple yet effective offline and online rl. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 3682–3691. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/ghasemipour21a.html. 4.4.2

[10] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016. 1

[11] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019. 2.2.2, 2.2.2, 3.1, 3.3.2

[12] Irina Higgins, Loïc Matthey, Arka Pal, Christopher P. Burgess, Xavier Glorot, Matthew M. Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2016. URL https://api.semanticscholar.org/CorpusID:46798026. 3.3.1

[13] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkE3y85ee. 3.2.2, 3.2.2, 3.3, 3.3.2

[14] Yiding Jiang, Evan Liu, Benjamin Eysenbach, J. Zico Kolter, and Chelsea Finn. Learning options via compression. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 21184–21199. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/8567a53e58a9fa4823af356c76ed943c-Paper-Conference.pdf. 3.1, 3.3.1, 3.3.1, 3.3.2, 3.3.2, 3.4.1, 4.1

[15] Taesup Kim, Sungjin Ahn, and Yoshua Bengio. Variational temporal abstraction. *Advances in Neural Information Processing Systems*, 32, 2019. 3.1, 3.3.1, 3.3.2, 3.3.2, 3.4.1, 3.4.2, 4.1

[16] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022. 2.2.1, 2.2.1, 2.2.1, 3.3.1, 3.3.1, 3.3.2

[17] Thomas Kipf, Yujia Li, Hanjun Dai, Vinícius Flores Zambaldi, Alvaro Sanchez-Gonzalez, Edward Grefenstette, Pushmeet Kohli, and Peter W. Battaglia. Compile: Compositional

imitation learning and execution. In *International Conference on Machine Learning*, 2018. URL https://api.semanticscholar.org/CorpusID:153312428. 3.1, 3.3.2, 3.3.2, 3.4.2, 4.1

[18] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1179–1191. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/0d2b2061826a5df3221116a5085a6052-Paper.pdf. 4.4.2

[19] Sang-Hyun Lee and Seung-Woo Seo. Learning compound tasks without task-specific knowledge via imitation and self-supervised learning. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5747–5756. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/lee20f.html. 3.1

[20] 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. 3.1, 3.2.2, 3.2.2, 3.3, 3.3.2, 3.3.2

[21] Dean A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988. URL https://proceedings.neurips.cc/paper_files/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf. 4.4.2

[22] Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1:127–190, 1999. 4.2, 4.2.2

[23] Tanmay Shankar and Abhinav Gupta. Learning robot skills with temporal variational inference. In *International Conference on Machine Learning*, pages 8624–8633. PMLR, 2020. 3.1, 3.3.2, 3.4.2, 4.1

[24] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations*, 2019. 1, 4.1

[25] Archit Sharma, Michael Ahn, Sergey Levine, Vikash Kumar, Karol Hausman, and Shixiang Gu. Emergent real-world robotic skills via unsupervised off-policy reinforcement learning. *arXiv preprint arXiv:2004.12974*, 2020. 1

[26] Mohit Sharma, Arjun Sharma, Nicholas Rhinehart, and Kris M. Kitani. Directed-info GAIL: Learning hierarchical policies from unsegmented demonstrations using directed information. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BJeWUs05KQ. 3.1, 3.4.1, 4.1

[27] Lucy Xiaoyang Shi, Joseph J. Lim, and Youngwoon Lee. Skill-based model-based reinforcement learning. In *Conference on Robot Learning*, 2022. 1, 4.1

[28] Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *Abstraction, Reformulation, and Approximation: 5th International Symposium, SARA 2002 Kananaskis, Alberta, Canada August 2–4, 2002 Proceedings 5*, pages 212–223. Springer, 2002. 1

[29] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html. 2.1

[30] 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. 1, 2.1.3

[31] 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, July 2021. ISSN 2377-3774. doi: 10.1109/lra.2021.3068891. URL http://dx.doi.org/10.1109/LRA.2021.3068891. 3.3.2, 3.4.2, 4.1

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf. 2.2.2, 3.2.1, 3.3

[33] Wilson Yan, Danijar Hafner, Stephen James, and Pieter Abbeel. Temporally consistent video transformer for long-term video prediction. In *International conference on machine learning*. PMLR, 2023. 2.2.2, 3.3.2

[34] Tianhe Yu, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn. Combo: Conservative offline model-based policy optimization. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances*

*in Neural Information Processing Systems*, volume 34, pages 28954–28967. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/f29a179746902e331572c483c45e5086-Paper.pdf. 4.4.2

[35] 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. 3.3.2

[36] Wenxuan Zhou, Sujay Bajracharya, and David Held. Plas: Latent action space for offline reinforcement learning. In *Conference on Robot Learning*, 2020. 4.4.2