Extended Abstract

Motivation Modern model-based reinforcement learning (RL) methods, such as DreamerV3, predict pointwise transitions in a latent space. While effective in many domains, this approach demands large training data and struggles to generalize across structurally similar states that differ only in surface-level details or context.

We hypothesize that decomposing the state into soft memberships over multiple learnable subspaces—each representing meaningful structure—can improve generalization, robustness, and sample efficiency. This idea builds on state abstraction but goes further by learning overlapping, interpretable latent dynamics.

Method We propose a set-based state abstraction framework in which latent states are represented as soft memberships in a set of learned subspaces (factors). Each subspace models a subset of state transitions, and the dynamics are computed by weighting each subspace's transition model according to its membership probability. This allows transitions to be modeled not just point-to-point, but between latent factors, enabling structural reuse and generalization.

We incorporate this approach into the DreamerV3 pipeline by replacing the recurrent state model with a "Set Dreamer" that uses a learned factor-based dynamics model. Each transition is generated by selecting dynamics from a learned matrix according to the factor membership vector of the current state.

Implementation We implemented a grid world transition model in PyTorch, using a custom environment and a custom neural network. We also designed and implemented several novel loss functions and wrote a custom hyperparameter search script to search over weighted combinations of these loss functions. We tested the ability of our model to reconstruct the generative factors used to define the transition function.

We implemented Set Dreamer in PyTorch, leveraging the codebase in `https://github.com/NM512/dreamerv3-torch/tree/main`. We compared Set Dreamer with the default DreamerV3 hyperparameters on the Atari100k Pong task Machado et al. (2018). For fair comparison, the hyperparameters were set to make the model sizes comparable, and we used the average evaluation return as a performance metric.

Results On its own, our implementation was unable to reconstruct the generative factors. We showed that as posed this problem is not identifiable. However, once we pretrained a transition function and fixed 2/10 factors, we showed we were able to learn the remaining 8 using our custom loss function.

Set Dreamer reached a final average evaluation return of 2.45 while DreamerV3 reached 21.

Discussion Contrary to our hypothesis that the set-based state abstraction framework would improve sample efficiency, Set Dreamer underperformed compared to the default DreamerV3.

Conclusion Our work extends the state abstraction literature and shows the limits of simultaneously learning subsets of state space that are causally related to each other, and the transition function between these subsets. Although finding such subsets promises to greatly increase data efficiency of model-based RL methods, we show that in deterministic systems this problem is ill-posed. Our work contributes multiple novel loss functions that, in simple stochastic settings, or settings with few predefined factors, facilitate the learning of others.

Dynamics Modeling between Learnable State Space Subsets for Data Efficient Reinforcement Learning

Hyun Dong Lee
Stanford University
`hdlee@stanford.edu`

Kyle Ellefsen
Stanford University
`kyleellefsen@gmail.com`

Abstract

Model-based reinforcement learning methods learn a recurrent model of the environment to support “imagination"-based planning in latent space. Despite their promising performance on complex tasks, these methods still often require substantial amounts of training data and fail to leverage structural similarities among different states, owing to their pointwise transition functions. In this work, we propose a novel approach that decomposes the world model dynamics into transitions between structured and overlapping subspaces of the latent space. With a toy grid world example, we designed and explored various loss functions to investigate whether we can recover the true subspaces. We further implemented Set Dreamer, a modification of DreamerV3 with our set-based state abstraction framework, and tested it on the Atari100k Pong task. Despite being a promising idea, the experimental results and our extensive analyses show how learning the correct subspaces is highly non-trivial, especially in high-dimensional and entangled environments.

1 Introduction

Model-based reinforcement learning has shown promising performance in complex environments Ha and Schmidhuber (2018); Hafner et al. (2019, 2023); Ahmadi et al. (2023). In particular, DreamerV3 has successfully leveraged compact latent representations and “imagination"-based planning to learn effective policies from limited data. Despite its success, DreamerV3 models the dynamics in the world model as pointwise transition functions in a latent state space, which may often require substantial amounts of training data and fail to generalize across states that share structural similarities.

To address these limitations, we explore a novel approach based on the hypothesis that the world model dynamics can be decomposed into transitions between structured subspaces of the latent state space. Drawing inspiration from state abstraction Li et al. (2006), we propose to represent each latent state as a soft membership distribution over a set of learnable subspaces, each grouping a coherent aspect of the environment's dynamics. By modeling transitions between these subspaces rather than individual latent states, we aim to enhance sample efficiency.

Our research questions were motivated by the goals of interpretability and data efficiency, inspired by reasoning in humans. In this project, we propose a novel *set-based state abstraction framework*. Our main research question was: can we learn overlapping state abstractions such that we reduce the data efficiency requirements by enabling ‘natural‘ generalization across similar states, where similarity is context-dependent?

2 Related Work

2.1 State Abstraction and Bisimulation

State abstraction is the process of mapping high-dimensional states in a Markov decision process (MDP) to a lower-dimensional or simplified representation that preserves relevant decision-making properties Li et al. (2006). It is a long-standing goal in reinforcement learning (RL), aimed at reducing the complexity of decision-making by mapping high-dimensional states to lower-dimensional representations that preserve task-relevant information. The intuition behind state abstraction is that by grouping states that are similar together, information learned about one state can be transferred to another, increasing data efficiency. Another advantage of the state abstraction approach is that, transitions between subsets of the state base have the potential to be somewhat more interpretable than transitions done in an abstract latent space (see Dreamer below). Bisimulation metrics formalize this by grouping together states with identical future dynamics and reward distributions. Zhang et al. Zhang et al. (2021) proposed a method to learn bisimulation-invariant latent spaces without relying on reconstruction losses, leading to improved generalization in visual environments.

However, recent work has identified limitations in offline RL settings where bisimulation can struggle due to incomplete data coverage and reward scaling issues Zang et al. (2023). In nearly all of the state abstraction literature, including bisimulation, the state space is partitioned on an equivalence relation in which points in state space are equivalent if they share properties, e.g. they have the same optimal action or the same reward and next state probabilities. However, some partitions might be efficient for some subtasks but ineffective for others. Decomposing the state space into *overlapping* subsets would enable different representations for different subtasks, but this remains underexplored in the literature.

2.2 Latent World Models in Model-Based RL

Model-based RL methods such as PlaNet Hafner et al. (2019) and Dreamer Hafner et al. (2023) learn recurrent state-space models (RSSMs) that support imagination-based planning in latent space. The motivation behind this ‘imagining‘ is that, by learning an accurate world model, agents can plan far into the future, in contrast to methods like TD updating that rely on propagating a scalar value backwards through time. The hope is that planning in an imagined latent space can improve data efficiency. Dreamer v3 Hafner et al. (2023) is a model-based reinforcement learning algorithm that learns a world model to simulate future trajectories and optimizes behavior by planning in this learned latent space. It was a notable advance over its predecessors -v1 and -v2 because of its stable training and generalization enhancements. It implements the world model as a Recurrent State-Space Model (RSSM) alongside convolutional encoders and decoders for observations. The policy and value functions are trained using “imagined" rollouts, with robustness techniques based on normalization, balancing, and transformations.

In the original Dreamer v3 paper, ablation showed that performance was heavily dependent on reconstruction loss. The reconstruction loss was required to prevent latent space collapse. However, reconstruction loss encourages the model to learn spurious correlations in the high dimensional observation, and as a result it becomes very sensitive to background features, decreasing robustness. There are a number of model-based reinforcement learning papers that learn a dynamics model in latent space without including a reconstruction loss. MuZero Schrittwieser et al. (2020) doesn’t include an explicit loss on ensuring temporal consistency in the latent space, but instead uses the dynamics model on the latent space to predict the value function, the policy, and the rewards, which all have losses on them. EfficientZero Ye et al. (2021) modifies MuZero to add a temporal consistency loss that in the latent space. DreamerPro Deng et al. (2022) modifies Dreamer by dropping the reconstruction loss and replacing it with a fixed number of ‘prototypes’, which are points in latent space. To avoid trivial solutions they explicitly encourage data points to be assigned to each cluster within a batch uniformly. While all these approaches avoid a reconstruction loss, they are still not data efficient enough to work on many robotics tasks.

While Dreamer-v3 is a model-based RL method with SOTA data efficiency, we believe model-based RL can still be more data efficient. By taking ideas from state-abstraction (namely the idea that features can be constructed out of sets of the state space), and dropping the requirement that these sets partition the state space, we hoped to test our ideas in the Dreamer model by substituting out the original encoder with one based on our novel architecture. If a model can learn sets of state space that are ‘the same’ for predicting future trajectories, it should be able to imagine rollouts of the future with more accuracy and without memorizing spurious correlations.

3 Method

3.1 Grid world

3.1.1 Motivation Behind Set Abstraction

The motivation for state abstraction arises from the desire to learn rules that apply to sets of states, rather than individual states, thus reducing the amount of data needed. Consider a grid world where states are the set of coordinates (x, y), with $x, y \in \{1, ..., 5\}$. Suppose a rule states: "if $x = 1$ then in the next timestep $x = 2$", then without state abstraction one would need to observe this transition for all 5 possible values of y. However, if the coordinates X and Y are known to be independent factors, this rule could be learned with fewer samples. We can define factors $f_1 = \{s \in S|x = 1\}$ and $g_1 = \{s' \in S'|x = 2\}$. The dynamics hypothesis $\phi_1 = (f_1, g_1)$ represents the prediction that if the current state x is in f_1 (i.e., its first coordinate is 1), then the next state s' will be in g_1 (i.e., its first coordinate will be 2).

3.1.2 Definitions

Each factor is a soft state indicator function.

$$f : S \to [0, 1] \tag{Factor}$$

Intuitively, if a factor mapped to the two element set $\{0, 1\}$, it would indicate a subset of the state space, and we could talk about a particular state being ‘in’ or ‘out of’ a factor. A factor is like the receptive field of a neuron, which measures how intensely a particular element ‘activates’ the factor.

A dynamics hypothesis (DH) is defined to be a pair consisting of an antecedent factor and a consequent factor.

$$\phi_i := (\text{ant}_{\phi_i}, \text{con}_{\phi_i}) \tag{Dynamics Hypothesis}$$

The dynamics-hypothesis-wise dynamics model is the weighted average of the normalized consequent factor and a uniform distribution over the next state space, weighted by the probability that the current state is in the antecedent.

$$p_{\phi_i}(s'|s) := \frac{1}{Z_{\phi_i}(s)} \text{con}_{\phi_i}(s')^{\text{ant}_{\phi_i}(s)} \tag{DH model}$$

The overall model is the product of these weighted by $P(\phi_i)$, normalized:

$$p_\Phi(s'|s) := \frac{1}{Z(s)} \prod_{\phi_i \in \Phi} p_{\phi_i}(s'|s)^{P(\phi_i)} \tag{Aggregate Model}$$

For each dynamics hypothesis and for the overall model, we can define the following cross-entropies and losses.

$$H(\mathcal{T}, p_\Phi|\mathbb{U}_S) := \mathop{\mathbb{E}}_{s\sim\mathbb{U}_S(\cdot)} \left[\mathop{\mathbb{E}}_{s'\sim\mathcal{T}(\cdot|s))} - \log p_\Phi(s'|s) \right] \tag{Aggregate Cross Entropy (CE_Φ)}$$

$$H(\mathcal{T}, p_\phi|\text{ant}_\phi) := \mathop{\mathbb{E}}_{s\sim\text{ant}_\phi(\cdot)} \left[\mathop{\mathbb{E}}_{s'\sim\mathcal{T}(\cdot|s))} - \log p_\phi(s'|s) \right] \tag{DH Gated Cross Entropy ($CE_{\phi|ant}$)}$$

$$H(\mathcal{T}, p_\phi|\mathbb{U}_S) := \mathop{\mathbb{E}}_{s\sim\mathbb{U}_S(\cdot)} \left[\mathop{\mathbb{E}}_{s'\sim\mathcal{T}(\cdot|s))} - \log p_\phi(s'|s) \right] \tag{DH Cross Entropy (CE_ϕ)}$$

The corresponding losses are

$$\mathcal{L}_{CE} = H(\mathcal{T}, p_\Phi | \mathbb{U}_S) \tag{1}$$

$$\mathcal{L}_{CE_{\phi|ant}} = \frac{1}{|\Phi|} \sum_{\phi_i \in \Phi} P(\phi_i) \cdot H(\mathcal{T}, p_{\phi_i} | \text{ant}_\phi) \tag{2}$$

$$\mathcal{L}_{CE_\phi} = \frac{1}{|\Phi|} \sum_{\phi_i \in \Phi} P(\phi_i) \cdot H(\mathcal{T}, p_{\phi_i} | \mathbb{U}_S) \tag{3}$$

$$\mathcal{L} := \mathcal{L}_{CE} + \beta_1 \mathcal{L}_{CE_{\phi|ant}} + \beta_2 \mathcal{L}_{CE_\phi} + \beta_3 \mathcal{L}_R \tag{4}$$

Why have 3 different cross-entropies? The first—the aggregate cross-entropy—indicates a good predictive model overall. The DH-gated cross-entropy measures, in the region of the antecedent space that this particular DH claims to be an expert in, how good its predictions are. However, minimizing this causes the antecedent size to shrink, resulting in a prediction that is accurate but overly specific and does not generalize. Finally, the DH cross-entropy measures how good this specific DH is at predicting across the entire antecedent space, encouraging the antecedent to be large. One idea is that, in optimization, for the DH Gated CE, one could freeze the antecedent of each dynamics hypothesis and modify the consequent to minimize this cross-entropy loss. For the DH Cross Entropy, one could conversely freeze the consequent and optimize the antecedent to expand as much as possible to increase abstraction and generalization. Since DH-specific CEs do not take into account the interaction between DH, the aggregate cross entropy would minimize redundancy between the DHs so they do not make the same predictions as each other; they provide coverage over the antecedent space, and simultaneously active DHs don't predict the same consequent.

Entropy Type	**Frozen**	**Optimized**	**Objective**
CE_Φ	–	Φ	ant coverage $\uparrow$, con orthogonality $\uparrow$
CE_ϕ	con$_\phi$	ant$_\phi$	$\lvert$ant$_\phi\rvert$ $\uparrow$
$CE_{\phi\|ant}$	ant$_\phi$	con$_\phi$	Consequent quality

Why aggregate DH-wise CEs by weighting by their probabilities? $P(\phi)$ is the degree of belief the agent has that ϕ is 'true', or more formally, that $CE_{\phi|ant} \approx 0$. Since dynamics hypotheses share factors, when doing gradient descent, if all CE_ϕ were treated equally, factors constituent of a ϕ with low $P(\phi)$ would contribute to the gradient as much as those with a high $P(\phi)$, which would cause forgetting of good dynamics hypotheses.

3.1.3 Redundancy Loss

Also core to the intuition is that for a state s, each dynamics hypothesis will reduce the possible consequent space independently. That is, if we view them as hard indicator functions in some $\mathbb{B}^N$ space, each one 'eliminates' a different binary dimension, such that if we start out with a space of size 2^N and we apply k dynamics hypotheses, then we would end up with a space of possibilities of size 2^{N-k}. The entropy of a factor measures how much of the log consequent space is remaining, so we can measure how much a DH shrinks the consequent space by measuring the change in entropy it is responsible for. Formally, for a factor ϕ the entropy at a point s can be defined as

$$H_s(p_\phi) := \mathop{\mathbb{E}}_{s' \sim p_\phi(\cdot|s)} [-\log p_\phi(s'|s)] \tag{5}$$

and the change of entropy ΔH can be defined as the difference between the entropy of a uniform distribution over S' and the final entropy

$$\Delta H_s(p_\phi) := \log |S'| - H_s(p_\phi) \tag{6}$$

For a collection of dynamics hypotheses Φ, the distribution formed by multiplying, then normalizing (as in Equation Aggregate Model) must have a lower ΔH than the sum of their individual entropies

$$\Delta H_s(p_\Phi) \leq \sum_{\phi_i \in \Phi} \Delta H_s(p_{\phi_i}) \tag{7}$$

Therefore, we can form a measure of information redundancy in a collection of dynamics hypotheses as

$$R(s) = \left(\sum_{\phi_i \in \Phi} \Delta H_s(p_{\phi_i}) \right) - \Delta H_s(p_\Phi) \tag{8}$$

$R(s) \in [0, \infty]$ measures how much redundancy there is. As a simple example of this, suppose we have a 16x16 grid world, and currently there are two active dynamics hypotheses ϕ_1 and ϕ_2, where $\text{con}_{\phi_1} = (x = 5)$ and $\text{con}_{\phi_2} = (y = 2)$. Then since $|S| = 2^8$ a uniform distribution has $H = 8$ bits, a factor that specifies the x coordinate has 4 bits, so the remaining entropy in our grid world after observing con_{ϕ_1} would be 4 bits, and another factor that specified the y coordinate would completely eliminate the entropy. That is, $\Delta H_s(p_{\phi_1}) = \Delta H_s(p_{\phi_2}) = 4$ and $\Delta H_s(p_\Phi) = 8$, so there is no redundancy.

We can define a new loss term

$$\begin{aligned}
\mathcal{L}_R :=& \mathop{\mathbb{E}}_{s \sim \mathbb{U}_S(\cdot)} [R(s)] && (9)\\
=& \mathop{\mathbb{E}}_{s \sim \mathbb{U}_S(\cdot)} \left[\left(\sum_{\phi_i \in \Phi} \Delta H_s(p_{\phi_i}) \right) - \Delta H_s(p_\Phi) \right] && (10)\\
=& \mathop{\mathbb{E}}_{s \sim \mathbb{U}_S(\cdot)} \left[\left(\sum_{\phi_i \in \Phi} \log |S'| - H_s(p_{\phi_i}) \right) - (\log |S'| - H_s(p_\Phi)) \right] && (11)\\
=& (|\Phi| - 1) \log |S'| + \mathop{\mathbb{E}}_{s \sim \mathbb{U}_S(\cdot)} \left[H_s(p_\Phi) - \left(\sum_{\phi_i \in \Phi} H_s(p_{\phi_i}) \right) \right] && (12)\\
=& K + \mathop{\mathbb{E}}_{s \sim \mathbb{U}_S(\cdot)} \left[H_s(p_\Phi) - \left(\sum_{\phi_i \in \Phi} H_s(p_{\phi_i}) \right) \right] && (13)
\end{aligned}$$

This loss should encourage sparsity of active DH, as inactive DHs (those where $\text{ant}_\phi(s) \approx 0$) will have uniform distributions as models, which have $\Delta H_s(p_\phi) \approx 0$. Importantly, this objective is independent of the true dynamics model, it is entirely a property of the relation between the factors themselves.

3.1.4 Grid world Architecture

The complete model for the grid world was a neural network with two components: an encoder and a transition matrix. The encoder mapped a one-hot representation of the state space through a 3-layer MLP followed by a sigmoid function onto a 10 dimensional latent vector where each element is constrained between 0 and 1. Each element corresponds to a factor, so there are 10 factors in total to represent the $|S| = 25$ states. The $(i, j)^{th}$ entry of the transition matrix represented the probability $P(\phi_{i,j})$ of the dynamics hypothesis defined as the pair of the ith antecedent factor and the jth consequent factor. The model was implemented in PyTorch and trained on a infinite data loader (i.e. new samples were constantly generated).

3.2 Set Dreamer

Here we introduce Set Dreamer. In Set Dreamer, we replace the GRU sequence model in DreamerV3 with the subspace abstraction framework.

We first review the world model of DreamerV3. DreamerV3's world model learns compact representations of input images through autoencoding and enables planning by predicting future image representations and rewards. The world model first maps input images x_t and current hidden state h_t to stochastic representations z_t. DreamerV3 then uses a GRU that takes in the previous hidden state, action, and image embedding to compute the next hidden state. The model is trained to use the hidden state to predict the image embedding and a concatenation of the hidden state and the image embedding to predict the reward and continuation flag and reconstruct the input image. The following is a list of equations that constitute the world model, as defined in the DreamerV3 paper Hafner et al. (2023):

$$
\begin{array}{l}
\text{RSSM} \left\{ \begin{array}{rl} \text{Sequence model:} & h_t = f_\phi\big(h_{t-1},\, z_{t-1},\, a_{t-1}\big) \\ \text{Encoder:} & z_t \sim q_\phi\big(z_t \mid h_t, x_t\big) \\ \text{Dynamics predictor:} & \hat{z}_t \sim p_\phi\big(\hat{z}_t \mid h_t\big) \end{array} \right. \\
\begin{array}{ll} \text{Reward predictor:} & \hat{r}_t \sim p_\phi\big(\hat{r}_t \mid h_t, z_t\big) \\ \text{Continue predictor:} & \hat{c}_t \sim p_\phi\big(\hat{c}_t \mid h_t, z_t\big) \\ \text{Decoder:} & \hat{x}_t \sim p_\phi\big(\hat{x}_t \mid h_t, z_t\big) \end{array}
\end{array}
\tag{1}
$$

Now, we define Set Dreamer. We replace the standard sequence model of DreamerV3 with a subspace abstraction framework:

$$\tau_{t-1} = f_\phi(h_{t-1}, z_{t-1}, a_{t-1}), \quad \tau_{t-1} \in \mathbb{R}^{D_h \times D_h}_{[0,1]} \tag{14}$$

$$h_t = 1 - \prod_{i=1}^{D_h} (1 - h_{t-1,i}\tau_{t-1,:,i}) \tag{15}$$

where $\tau_{t-1,:,i}$ is the ith column of τ_{t-1} and $h_{t-1,i}$ is the ith entry of h_{t-1}. Here, the function f_ϕ is parameterized by a neural network (e.g., an MLP) and is followed by a sigmoid activation function to ensure outputs between 0 and 1.

The matrix τ_t represents a *dynamics hypothesis matrix*, where the element in the jth row and ith column corresponds to the probability that the dynamics hypothesis $\phi_{i\to j}$ is valid. We interpret each dimension i of the recurrent state $h_t \in [0,1]^{D_h}$ as representing the probability of membership in subspace U_i at time t. Specifically, the recurrence in equation (15) can be understood as follows:

$$
\begin{aligned}
p(h_{t,j} = 1) =& 1 - p(h_{t,j} = 0) \\
=& 1 - \prod_{i=1}^{D_h} (1 - p(h_{t-1,i} = 1)p(\phi_{i\to j} = 1))
\end{aligned}
$$

This simplification arises under the assumptions that:

- If the system is a member of subspace U_i and if the dynamics hypothesis $\phi_{i\to j}$ is valid for at least one $i \in [1, \dots, D_h]$, the system will transition into subspace U_j.
- If the system is not a member of subspace U_i or if the dynamics hypothesis $\phi_{i\to j}$ is invalid for all $i \in [1, \dots, D_h]$, the system will not transition into subspace U_j.

Although this recurrence formulation enjoys a clear probabilistic interpretation, it suffers from a couple of practical drawbacks:

- Numerical underflow / vanishing gradients: Multiplying many factors (i.e. when D_h is large) in $[0,1]$ underflows to zero in finite precision, and $\prod_{i=1}^{D_h}(1 - p(h_{t-1,i} = 1)p(\phi_{i\to j} = 1))$ vanishes once any term is zero.
- Parameter blow-up: Naively computing a dense $\tau \in \mathbb{R}^{D_h \times D_h}$ from the hidden state, action, and image embedding with a feedforward NN is costly.

Thus, rather than multiplying D_h factors, we exploit the structure of the Poisson process and its hazard rate to obtain a numerically stable framework.

1. Interpret each coordinate i at step $t-1$ as an exposure time $h_{t-1,i} \in [0,1]$ to a source i.
2. Learn a nonnegative "hazard" rate matrix $\Omega \in \mathbb{R}^{D_h \times D_h}_{\geq 0}$. Each entry of Ω_{ji} (the ith column and jth row entry of Ω) represents how much an exposure to source i leads to subspace j. In our case, ending up in subspace j would be considered a "failure" or "death".
3. Assuming independence across sources i, the survival function at coordinate j is $\exp(-\sum_{i=1}^{D_h} \Omega_{ji} h_{t-1,i})$.

4. The final recurrence thus becomes

$$h_t = 1 - \exp(-\Omega h_{t-1})$$

where the exponential is applied element-wise.

This formulation is exactly equivalent to our initial formulation in equation (15) when the hidden states are restricted to be binary. With this reparameterization, we get a more numerically stable framework with better gradients.

In addition, in order to avoid parameter blow-up, we parameterize τ as a low-rank plus diagonal matrix. In other words, we compute $U_\tau = f_{\phi_U}(h_{t-1}, z_{t-1}, a_{t-1}) \in \mathbb{R}^{D_h \times r}$, $V_\tau = f_{\phi_V}(h_{t-1}, z_{t-1}, a_{t-1}) \in \mathbb{R}^{D_h \times r}$, and $e_\tau = f_{\phi_e}(h_{t-1}, z_{t-1}, a_{t-1}) \in \mathbb{R}^{D_h}$, for $r << D_h$. Then, we set $\tau = U_\tau V_\tau + \text{diag}(e_\tau)$, which greatly reduce the number of parameters.

4 Experimental Setup

4.1 Grid world problem

We call the factors that were used in the generation of a dynamics process the 'generative factors'. To test out the ability of our loss functions to reconstruct the generative factors, we set up a toy grid world. The grid world consisted of a the discrete Cartesian plane of size 5×5 and a dynamics process $\mathcal{T}(s'|s)$ that mapped $\mathcal{T} : (x, y) \mapsto (f(x), g(y))$, where f and g are deterministic stationary functions of one coordinate. The functions were constructed so that there each coordinate (e.g. x) was permuted with a cycle size equal to the size of the entire dimension, so there were no smaller loops. During training we sampled from a uniform distribution over the state space denoted $\mathbb{U}_S(S)$ and trained with single step trajectories. The goal was then, based on these single step trajectories, to simultaneously learn factors that were sufficient in modeling the dynamics, and the dynamics themselves. Although we tracked our four losses, our main success metric was the reconstruction of the generative factors, described below.

4.1.1 Objective: Factor Reconstruction

Factors are defined below, but in the grid world setup, the generative factors (factors that generated the dynamics) are indicator functions over the discrete Cartesian coordinates that pick out a fixed x-coordinate (column) or y-coordinate (row). For example, the factor $f_{x=4} : \mathbb{Z}^2 \to \{0, 1\}$ that maps each point (x, y) to 1 if $x = 4$ and 0 otherwise. this problem, each x coordinate and each y coordinate correspond to a distinct factor. That is, since a particular x coordinate in a discrete cartesian plane is a column, and each y coordinate a row, the factor $x = 4$ corresponds to the column at $x = 4$. More specifically, a factor is the indicator function mapping points in the cartesian plane to the set $0, 1$ indicating whether or not $x = 4$.

Our main objective with the grid world was to show that we could learn the generative factors from only the single step transitions. To measure this, we define a factor reconstruction score. Given a generative factor $f : S \to \{0, 1\}$ and a reconstructed factor $\hat{f} : S \to [0, 1]$,

$$Rec(f, \hat{f}) := \frac{1}{|S|} \sum_{s \in S} -f(s) \log \hat{f}(s) - (1 - f(s)) \log(1 - \hat{f}(s))$$

Since we are attempting reconstruction up to permutation, we greedily matched learned factors with the generative factor with the best (lowest) reconstruction score, and summed $Rec(f, \hat{f})$ over all matched pairs of factors. We always used the same number of generative and reconstructive factors. This loss was not used in training at any point, only in evaluation.

4.2 Atari100k Pong

To test Set Dreamer, we compared the performance of Set Dreamer and the default DreamerV3 on the Atari100k Pong task Machado et al. (2018). We used the "ALE/Pong-v5" environment, whose action space consists of 6 discrete actions (NOOP, FIRE, RIGHT, LEFT, RIGHTFIRE, and LEFTFIRE) and the observation space consists of RGB images of size $(210, 160, 3)$. Atari100k is a low-data

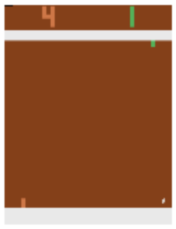


Figure 1: Atari Pong Environment.

variant of the default Atari benchmark, where the agent is limited to 100k agent steps to interact with the environment. Both Set Dreamer and default DreamerV3 were trained for 400k steps. The hyperparameters of DreamerV3 were set to the default hyperparameters outlined in Hafner et al. (2023), and the hyperparameters of Set Dreamer were set to match the parameter count of DreamerV3 for a fair comparison.

5 Results

Here we show the quantitative and qualitative experimental results from the Gridworld and Atari Pong experiments.

5.1 Quantitative Evaluation

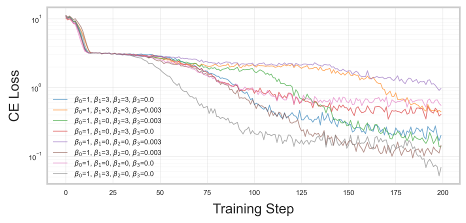
5.1.1 Gridworld

Because we had 4 losses, we performed a hyperparameter search on their relative weights (i.e. β coefficients). Unfortunately, when we trained the entire model end to end, we were not able to recover the original factors, or indeed minimize the cross entropy loss. For unknown reasons the transition matrix, although it contained only 100 parameters, was extremely slow to converge. And, when we pretrained the transition matrix to convergence (10,000 epochs), we still were unable to learn factors that minimized the cross entropy loss. However, when pretrained the transition matrix and fixed two factors to the generative factors (corresponding to indicators for one column and one row of our grid world respectively), we were able to reconstruct the other 8 factors. The results in Table 1 and Figure 2 show a hyperparameter search which started out with the pretrained transition matrix and 2 fixed factors.

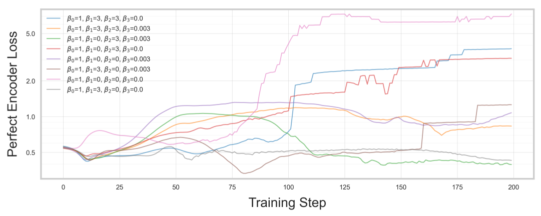
Table 1: Experimental results for training runs with different loss weights showing all four components of the training losses, as well as the generative factor reconstruction score (i.e. perfect encoder loss). All numbers were taken after the end of the 200 epoch run. The reconstruction score was not used in any training objective, while the aggregate cross entropy $\mathcal{L}_{CE}$ was used in all runs.

Run	$\mathcal{L}_{CE}$	$\mathcal{L}_{CE_{\phi\|ant}}$	$\mathcal{L}_{CE_{\phi}}$	$\mathcal{L}_{R}$	Perfect Encoder Loss
β_0=1, β_1=0, β_2=0, β_3=0.0	0.5507	0.3023	0.3254	36.8493	7.2977
β_0=1, β_1=0, β_2=0, β_3=0.003	0.9728	0.2527	0.3057	17.7221	1.0777
β_0=1, β_1=0, β_2=3, β_3=0.0	0.4078	0.2444	0.2909	25.1974	3.0952
β_0=1, β_1=0, β_2=3, β_3=0.003	0.1376	0.2185	0.2893	28.1841	0.3952
β_0=1, β_1=3, β_2=0, β_3=0.0	0.0690	0.1973	0.2809	29.1047	0.4284
β_0=1, β_1=3, β_2=0, β_3=0.003	0.1448	0.1929	0.2829	29.5455	1.2591
β_0=1, β_1=3, β_2=3, β_3=0.0	0.2024	0.2063	0.2898	33.1035	3.7269
β_0=1, β_1=3, β_2=3, β_3=0.003	0.4443	0.1977	0.2809	26.2355	0.8324

Although there was considerable trial-to-trial noise, a consistent finding was that addition of one or more of the losses in addition to the vanilla cross entropy loss were required for generative factor reconstruction.



(a) Cross Entropy Loss

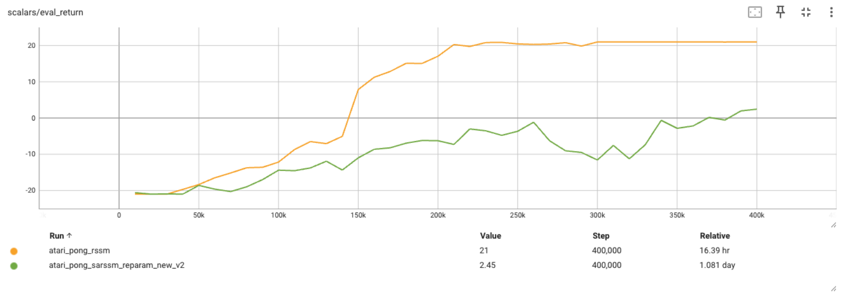


(b) Perfect Encoder Loss

Figure 2: Learning curves showing (a) $\mathcal{L}_{CE}$, the aggregate cross entropy and (b) $\mathcal{L}_R$, the distance the reconstructed factors are from the generative factors.

Table 2: Set Dreamer vs. DreamerV3 Performance Comparison on Atari100k Pong

Method	Final Average Eval Return (↑)
Default DreamerV3	21
Set Dreamer	2.45



	Run ↑	Value	Step	Relative
●	atari_pong_rssm	21	400,000	16.39 hr
●	atari_pong_sarssm_reparam_new_v2	2.45	400,000	1.081 day

Figure 3: Average eval return as a function of training steps.

5.1.2 Atari100k Pong

Here we show the average evaluation return as a function of training steps. We see that the default DreamerV3 outperforms Set Dreamer, converging to a higher average evaluation return in a shorter number of steps. We hypothesize that this gap in the performance could be caused by the following:

1. In many cases, we observed that the paddle that the agent controls gets stuck at either end of the frame. This phenomenon is potentially due to the collapsing states (i.e., all the states tend toward zeros or ones as we apply the world model recurrence), caused by the difficulty of learning the useful subspaces in a high-dimensional and entangled environment.
2. Set Dreamer is not as expressive as the default DreamerV3 that uses a GRU as its backbone sequence model. In particular, we think that the performance gap could also be due to the fact that we had to set the rank r of Set Dreamer to a small value to make the number of parameters between the two models comparable. While the number of parameters was comparable, making the comparison fair, this may have been too restrictive for Set Dreamer to learn an expressive dynamics hypothesis matrix τ.

5.2 Qualitative Analysis

5.2.1 Grid world

One way to capture the performance of the grid world model is to plot out the receptive fields. As shown in Figure 4, the receptive fields of each factor changed dramatically during training. Videos of the factor learning process reveal 'waves' of factor reconstruction, propagating backwards from the fixed factors. For example, Factor 0 (corresponding to $x = 0$) is always followed by Factor 1 (corresponding to $x = 1$). Since Factor 1 is fixed, the first factor to be learned is Factor 0, followed by

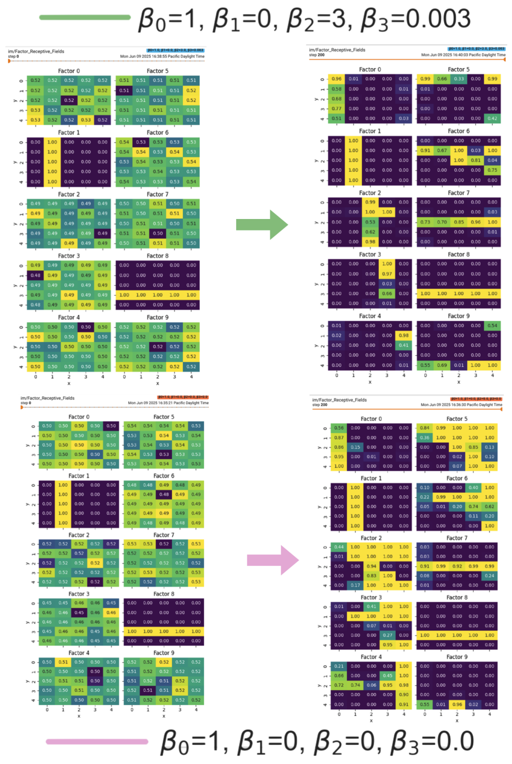


Figure 4: Changes in Receptive Fields from the best (top) and worst (bottom) hyperparameters in the grid training. Left are the starting receptive fields, with factors 1 and 8 fixed to be the column $x = 1$ and row $y = 3$ respectively. Addition of a nonzero beta for $\mathcal{L}_{CE_\phi}$ and $\mathcal{L}_R$ can be seen to improve reconstruction of the generative factors.

factor 5 (corresponding to $x = 4$), etc. It seems as if once these target factors are learned, it unblocks the learning of the causally upstream factor. This qualitative observation was very consistent across many hyperparameters, particularly when the coefficient for the CE_ϕ was nonzero.

6 Discussion

6.1 Why can't we recover the generative factors in a deterministic Markov process?

The generative factors are not identifiable We suspected that, given these single step trajectories and our four losses, it was impossible to reconstruct the original generative factors without fixing several of the factors. In order to show this, we constructed two sets of dynamics hypotheses, shown in Figure 6. One set consisted of the generative dynamics hypotheses, while the other was generated randomly such that it still was consistent with the transition function. As seen in Table 3, when we computed the loss terms for both complete models, they had exactly the same values. This shows in a proof by example that this problem has multiple global minima, no matter what coefficients are placed on the losses. In other words, recovering the unique generative factors in a deterministic grid world is not possible.

Atomic and Compound Dynamics Hypotheses Let's define a *point* dynamics hypothesis $\dot{\phi}$ to be one that has exactly a single state in the antecedent and exactly a single state in the consequent. In a deterministic Markov process, every state goes deterministically to exactly one state. Given some state space of size $|S|$, this can be perfectly modeled with a single point dynamics hypothesis for every antecedent state, indicating where it transitions to. Given a collection of such distinct point dynamics hypotheses, the product of all these conditional densities perfectly models the Markov process. That is, $H(\mathcal{T}, \left(\prod_{\dot{\phi}_i \in \Phi} p_{\dot{\phi}_i}\right) | \mathbb{U}_S) = 0$, as there is no uncertainty remaining about the next

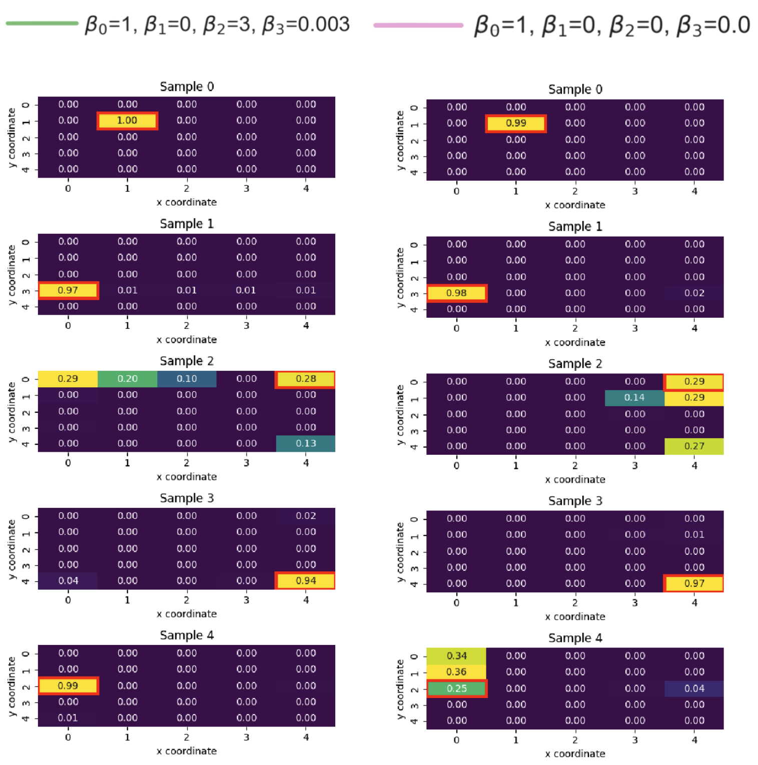


Figure 5: Example next-state predictions of the complete dynamics models.

Table 3: Comparison of Loss Values for two collections of DHs

	Generative DHs	**Random DHs**
$\mathcal{L}_{CE}$	0.0	0.0
$\mathcal{L}_{CE_{\phi\|\text{ant}}}$	2.0	2.0
$\mathcal{L}_{CE_\phi}$	3.5	3.5
$\mathcal{L}_R$	0.0	0.0

state. To see this, observe that for all dynamics hypotheses where s_t is not in the antecedent, $p_{\phi_i}(s'|s)$ is equal to the uniform distribution over the next state, whereas it is equal to the Dirac distribution of the correct consequent factor. The product of an arbitrary number of uniforms with a Dirac is that Dirac.

Define two binary operators on dynamics hypothesis: $\wedge$ and $\vee$. We can define these as the product t-norm and probabilistic sum s-norm (from fuzzy logic), respectively

$$\phi_1 \wedge \phi_2 := (\text{ant}_{\phi_1}(s) \cdot \text{ant}_{\phi_2}(s), \text{con}_{\phi_1}(s) \cdot \text{con}_{\phi_2}(s)) \tag{16}$$

$$\phi_1 \vee \phi_2 := (\text{ant}_{\phi_1}(s) + \text{ant}_{\phi_2}(s) - \text{ant}_{\phi_1}(s) \cdot \text{ant}_{\phi_2}(s), \text{con}_{\phi_1}(s) + \text{con}_{\phi_2}(s) - \text{con}_{\phi_1}(s) \cdot \text{con}_{\phi_2}(s)) \tag{17}$$

Recall that both factors of a dynamics hypothesis map $S \to [0, 1]$. Dynamics hypotheses are thus closed under $\wedge$ and $\vee$. Consider the disjunction of two dynamics hypotheses. The resulting dynamics hypothesis would state "if s_t is an element of either ant_{ϕ_1} or ant_{ϕ_2} then s_{t+1} will be in either con_{ϕ_1} or con_{ϕ_2}" The interpretation of the conjunction of two dynamics hypotheses would follow similarly: "if s_t is an element of both ant_{ϕ_1} and ant_{ϕ_1} then s_{t+1} will be in con_{ϕ_1} and con_{ϕ_2}". Note that in these operators, information about the original factors is not preserved in the new factors.

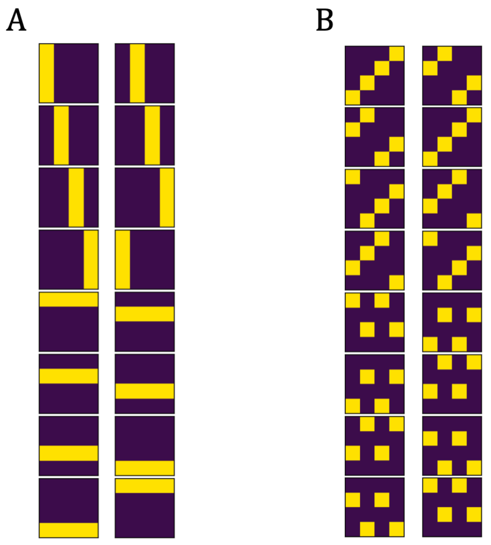


Figure 6: The generative collection of dynamics hypotheses (A) vs a random collection (B). Each row consists of a DH (a pair of factors) where the antecedent (left) deterministically precedes the consequent (right). Both have exactly the same values for all 4 losses on the task, when each state (x, y) deterministically transitions to $(x + 1 mod N, y + 1 mod N)$.

Defining conjunction and disjunction on dynamics hypotheses is useful because it allows us to compute the conditional cross entropy of the resulting dynamics hypotheses for example $H(\mathcal{T}, p_{\phi_1 \wedge \phi_2} | \mathbb{U}_S)$ or $H(\mathcal{T}, p_{\phi_1 \vee \phi_2} | \mathbb{U}_S)$. Suppose we have a collection of distinct correct point dynamics hypotheses $\dot{\phi}$. What happens as we take their disjunction? The antecedent grows, the coverage of the antecedent goes up, which decreases the C.E. However, as the consequent grows, the surprise of the consequent, even when inside the antecedent, increases, decreasing the C.E.

Core to the intuition of our formulation is the idea that good dynamics hypotheses are those that have antecedents and consequents that are bound in real causal relationships. In our gridworld, we sought to essentially combine (with this $\vee$ operator) point dynamics hypotheses where, for example, all points in the antecedent and consequent corresponded to the same values of x or y. that is, if $\phi_1 = ((x = 1, y = 4), (x = 2, y = 7))$ and $\phi_2 = ((x = 1, y = 8), (x = 2, y = 1))$ then $\phi_1 \vee \phi_2 = ((x = 1, y = (4 \vee 8)), (x = 2, y = (7 \vee 1))$ gets us closer to a pure factor that drops all reference to y (i.e. $(x = 1, x = 2)$). However, the cross-entropy of this disjunction $(\phi_1 \vee \phi_2)$ is no less than that of any other arbitrary disjunction. Because of this, our training algorithm that only uses CE_ϕ as the loss learns arbitrary disjunctions of true point dynamics hypotheses, not corresponding to the factors. However, when we also consider CE_Φ, the more independent these antecedents are relative to each other, the lower the cross entropy.

Suppose we have two dynamics hypotheses ϕ_1 and ϕ_2 that have two distinct antecedent factors A_1 and A_2, but have the same consequent C. For example, let $A_1 = (x = 0 \vee 1, y = 0))$, let $A_2 = (x = 2 \vee 3, y = 0)$, and let $C = (y = 1)$. In words, both ϕ_1 and ϕ_2 say that for particular values of x, y transitions from 0 to 1, but they have disjoint values of x. Suppose we have a new factor ϕ_3 that has the same consequent factor, but merges the two antecedents to be $A_3 = (x = 0 \vee 1 \vee 2 \vee 3, y = 0)$. Then this new ϕ_3 has the same cross entropy as $\{\phi_1, \phi_2\}$, but is more compact, and is closer to the generative factor (a pure factor of y).

Our argument is that ϕ_3 is likely to be learned when the y transitions are randomized, but not when the Markov process is deterministic. This is because the formation of ϕ_3 requires ϕ_1 and ϕ_2 to have exactly the same consequent, a pure factor of y. However, as A_1, the antecedent of ϕ_1, has information about the x coordinate, it is more likely that instead of learning the consequent C, it will instead learn something about where the x's transition to (i.e. $\tilde{C} = (x = 6 \vee 7 \vee 8, y = 1)$, and that will get entangled with the y transition in C. Once the two consequents for ϕ_1 and ϕ_2 diverge, the merger of the dynamics hypotheses into ϕ_3 would actually increase the cross entropy, and so would not be learned.

7 Conclusion

In this work, we explored how we can improve interpretability and data efficiency of model-based reinforcement learning methods by incorporating the set-based state abstraction framework, a novel framework where we represent each latent state as a soft membership distribution over a set of subspaces and model the dynamics between these subspaces. We tested our framework on a toy grid world problem and extensively analyzed the effects of different loss functions and why we cannot recover the ground truth factors. We further implemented Set Dreamer by incorporating the set-based state abstraction framework into DreamerV3 and tested on the Atari100k Pong task. Although modeling dynamics between learnable subsets of state space is a promising idea, offering interpretability and potential for generalization, we found that learning the correct subsets (factors) is highly non-trivial, especially in high-dimensional, entangled environments. When naively integrated into DreamerV3, Set Dreamer fails to learn the task, highlighting the practical limitations of this abstraction. For future work, we could explore nonlinear interactions between factors (e.g., attention or gating) or incorporate temporal hierarchies, where some factors evolve on longer timescales.

8 Team Contributions

- **Hyun Dong Lee:** Architecture design, implementation. Set Dreamer implementation. Paper write-up. Poster creation.
- **Kyle Ellefsen:** Architecture design, implementation. Loss function design. Grid world implementation and related ablation experiments. Paper write-up. Poster creation.

Changes from Proposal We focused more on understanding why our framework fails in the toy grid world example, recognizing the importance of diagnosing issues in a simpler setting.

References

Aida Ahmadi, Danijar Hafner, et al. 2023. Learning Contextual World Models for Generalization. In *NeurIPS*.

Fei Deng, Ingook Jang, and Sungjin Ahn. 2022. Dreamerpro: Reconstruction-free model-based reinforcement learning with prototypical representations. In *International Conference on Machine Learning*. PMLR, 4956–4975.

David Ha and Jürgen Schmidhuber. 2018. World models. *arXiv preprint arXiv:1803.10122* (2018).

Danijar Hafner, Nicolas Heess, et al. 2023. Mastering Diverse Domains through World Models. In *International Conference on Learning Representations (ICLR)*.

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

Lihong Li, Thomas J Walsh, and Michael L Littman. 2006. Towards a unified theory of state abstraction for MDPs. *AI&M* 1, 2 (2006), 3.

Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew J. Hausknecht, and Michael Bowling. 2018. Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents. *Journal of Artificial Intelligence Research* 61 (2018), 523–562.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature* 588, 7839 (2020), 604–609.

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. 2021. Mastering atari games with limited data. *Advances in neural information processing systems* 34 (2021), 25476–25488.

Hengyuan Zang, Lirong Li, et al. 2023. Understanding and Addressing the Pitfalls of Bisimulation Metrics in Offline Reinforcement Learning. *arXiv preprint arXiv:2306.00949* (2023).

Amy Zhang, Rowan McAllister, William Whitney, et al. 2021. Learning Invariant Representations for Reinforcement Learning without Reconstruction. In *International Conference on Learning Representations (ICLR)*.

A Additional Theory

A.1 Decompositions of Conditional Cross Entropies

Section Summary: the conditional entropy $H(\mathcal{T}|\mathbb{U}_S)$ is a lower bound on the conditional cross entropy of any model.

All three cross-entropy terms above are conditional cross-entropies, conditional on the parent variable distribution $p(s)$. They can be decomposed into the sum of the conditional KL divergence, which can be reduced when varying the parameters of the DH, and the conditional entropy, which cannot.

$$\underbrace{H(\mathcal{T}, p_\Phi|\mathbb{U}_S)}_{\text{Conditional Cross Entropy}} = \underbrace{H(\mathcal{T}|\mathbb{U}_S)}_{\text{Conditional Entropy}} + \underbrace{D_{\text{KL}}(\mathcal{T}||p_\Phi|\mathbb{U}_S)}_{\text{Conditional KL Divergence}}$$

The KL Divergence is bounded in $[0, \infty]$ and is a measure of how dissimilar our model of the dynamics process to $\mathcal{T}$. If the dynamics is deterministic, the conditional entropy term is zero, and the KL term and the cross entropy term are equal. We can compare how well our model p_Φ is doing to a baseline model. For example, we could use the marginal distribution of the next state as a baseline:

$$p(s') := \mathop{\mathbb{E}}_{s\sim\mathbb{U}_S(\cdot)} [\mathcal{T}(s'|s)]$$

We could introduce a measure of how much better off we are using a model p_Φ compared to this baseline as the difference in conditional KL divergences.

$$\Delta D_{KL}(p_\Phi) := D_{\text{KL}}(\mathcal{T}||p(s')|\mathbb{U}_S) - D_{\text{KL}}(\mathcal{T}||p_\Phi|\mathbb{U}_S) \tag{18}$$

$$= H(\mathcal{T}, p(s')|\mathbb{U}_S) - H(\mathcal{T}, p_\Phi|\mathbb{U}_S) \tag{19}$$

$$= \mathop{\mathbb{E}}_{s\sim\mathbb{U}_S(\cdot)} \left[\mathop{\mathbb{E}}_{s'\sim\mathcal{T}(\cdot|s))} - \log p(s') \right] - \mathop{\mathbb{E}}_{s\sim\mathbb{U}_S(\cdot)} \left[\mathop{\mathbb{E}}_{s'\sim\mathcal{T}(\cdot|s))} - \log p_\Phi(s'|s) \right] \tag{20}$$

$$= H(S'|S) - H(\mathcal{T}, p_\phi|\mathbb{U}_S) \tag{21}$$

That is, it is the average reduction in surprise of the consequent state when sampling the antecedent state from a uniform distribution. How much does using this dynamics hypothesis alone decrease the average surprise compared to a base model that simply modeled $P(s')$ and was blind to s.