Building Reinforcement Learning Algorithms that Generalize: From Latent Dynamics Models to Meta-Learning

JD Co-Reyes

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-178
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-178.html

August 10, 2021

Building Reinforcement Learning Algorithms that Generalize: From Latent Dynamics Models to Meta-Learning

by

John D Co-Reyes

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Assistant Professor Sergey Levine, Chair
Professor Pieter Abbeel
Professor Alison Gopnik

Summer 2021

Building Reinforcement Learning Algorithms that Generalize: From Latent Dynamics Models to Meta-Learning

Abstract

Building Reinforcement Learning Algorithms that Generalize: From Latent Dynamics Models to Meta-Learning

by

John D Co-Reyes

Doctor of Philosophy in Computer Science

University of California, Berkeley

Assistant Professor Sergey Levine, Chair

Building general purpose RL algorithms that can efficiently solve a wide variety of problems will require encoding the right structure and representations into our models. A key component of our generalization capability is our ability to develop an internal model of the world that can be used for robust prediction and efficient planning. In this thesis, we discuss work that leverages representation learning to learn better predictive models of physical scenes and enable an agent to generalize to new tasks by planning with the learned model under the framework of model-based RL. We cover two kinds of abstraction that can enable good generalization: state abstraction in the form of object level representations and temporal abstraction in the form of skill representations for hierarchical RL. By incorporating these abstractions into our models, we can achieve efficient learning and combinatorial generalization over long horizon, multi-stage problems. We also discuss the role of meta-learning in automatically learning the right structure for general RL algorithms. By leveraging large scale evolutionary based computation, we can meta-learn general purpose RL algorithms that have better sample efficiency and final performance over a wide variety of tasks. Finally, we cover how these internal models can be used to compute the RL objective itself and train general RL agents with complex behavior without having to design the reward function.

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I want to thank my advisor, Sergey Levine, for his mentorship and for guiding me through my PhD. His commitment and continuous support has made this PhD possible and shaped me into a better researcher. Thank you to my committee members, Pieter Abbeel and Alison Gopnik for their support and guidance. I also want to thank my undergraduate advisor, Yisong Yue, for inspiring me to pursue a career in research.

I am grateful to have had the opportunity to collaborate with an amazing group of students, faculty, and researchers during my PhD. For the work in this thesis, I want to thank Michael Chang, Michael Janner, Nicholas Rhinehart, Ben Eysenbach, Chelsea Finn, Danijar Hafner, Jiajun Wu, and Josh Tenenbaum. I would especially like to thank mentors Glen Berseth and Abhishek Gupta for not only collaborating with me but also providing support and guidance on a range of issues throughout my PhD. I would also like to thank all of the undergraduate students I worked with during my PhD including Suvansh Sanjeev, Rishi Veerapaneni, Andrew Liu, Jie Qiu, Sarah Feng, and Jenny Wang. Beyond the work in this thesis, I also had the pleasure of working with a number of other students and faculty including Justin Fu, Nick Altieri, Jacob Andreas, and John Denero.

Thank you to all the researchers in Berkeley AI Research for creating a collaborative and friendly environment. In particular, I want to thank my nearby labmates Vitchyr Pong, Justin Fu, Avi Singh, Marvin Zhang, and Dierdre Quillen for all the fun discussions and late nights working together in the lab. Discussions with Vitchyr Pong, who sat next to me, were especially helpful in getting me through the challenging times of my PhD. I also appreciate the support of Sandy Huang, Alex Lee, Greg Kahn, and Coline Devin.

I'm fortunate to have had the opportunity to do an internship at Google Brain under Aleskandra Faust and Honglak Lee, collaborating with Yingjie Miao, Daiyi Peng, Esteban Real, Sergey Levine, and Quoc V Le. Aleksandra provided me great freedom to work on any problem and I'm also grateful for support and thoughtful conversations with many including Luke Metz, Hanjun Dai, Xingyou Song, Krzysztof Choromanski, and Kevin Wu.

Finally, I'd like to thank my parents, Gigi Cobonpue and John Reyes, and my partner, Cullen Reilly, for all their years of love and support.

Chapter 1

Introduction

Humans have a remarkable capability to solve almost any problem given enough time. Our brains are flexible enough to adapt, acquire the right information, and learn new skills to tackle any challenge. From navigation to tool use to conducting basic research, we can rapidly generalize to new situations that we've never seen before. How can we build a similar system artificially that can solve any relevant problem efficiently? This thesis will outline progress along this goal of building general purpose agents that have the same problem solving flexibility as humans. The first part will focus on representation learning and designing the right inductive biases for reasoning about the world in terms of more abstract and composable building blocks while the second part will focus on removing human supervision to automatically design general purpose agents.

Reinforcement learning (RL) provides a general framework to design intelligent computational systems that can solve any problem that humans can. Advances in reinforcement learning has led to human-level performance in Atari (Mnih et al., 2015), Go (Silver et al., 2016) and has been successful in other domains including robotics (Levine et al., 2016; Kalashnikov et al., 2018), NLP (Paulus et al., 2018), and chip design (Mirhoseini et al., 2021). However, generalization and sample efficiency are still key challenges with existing model-free reinforcement learning algorithms. While RL training usually requires millions of samples (Schulman et al., 2015) and is specific to a particular environment, humans are able to generalize to completely new situations and master new tasks quickly with little data.

In this thesis, we take the stance that a large degree of our generalization capability comes from our ability to build an internal model of the world. Advances in model-based RL has led to greater sample efficiency (Nagabandi et al., 2017) and faster adaptation (Clavera et al., 2019). This model can be used to predict what happens when the agent takes a particular action, plan an optimal course of actions, and deal with partial observability in large open ended worlds. The particular structure of this internal model can enable good generalization. For example, operating on raw image observations would be quite costly. If we instead operate on a more abstract space such as representing the world as being composed of objects, our agent can plan and reason about the world over the space of discrete entities and their interactions rather than pixels. This more compact representation of the world can enable to agent to generalize to more complex scenes with any number of new objects.

Being able to decompose the world in terms of abstract and reusable building blocks such as objects is a powerful tool for generalization. We can leverage existing knowledge in a combinatorial number of ways and understand novel scenes in terms of known concepts. Humans understand the world in terms of objects at an early age (Spelke and Kinzler, 2007) and learn concepts such as object permanence (Baillargeon et al., 1985) at five months old. Young children start to use gestures to refer symbolically to objects or events (Acredolo and Goodwyn, 1988) and this becomes more general over time to include concepts, abstractions, behaviors, and categories. These composable building blocks can be used for building powerful structured mental models of the world. Abstracting our raw visual perception of the world into a compositional description in terms of more abstract entities can enable combinatorial generalization over new combinations of known entities.

While modern neural networks have been successful in a variety of domains such as visual object recognition (Krizhevsky et al., 2012) and speech recognition (Hinton et al., 2012), they still struggle to generalize such as to different datasets (Yosinski et al., 2014), are fragile to distributional shift in reinforcement learning (Kansky et al., 2017), and require more training data than humans (Tsividis et al., 2017). This suggests that neural networks do not explicitly learn a compositional representation that can easily be reused for new tasks and instead learn mostly surface statistics such as between textures and classifications in images (Lake and Baroni, 2018). A reasonable solution would be to take a more unified approach which incorporates inductive biases in neural networks that enables learning about the right set of abstractions and symbols. By tightly integrating neural networks with the ability to learn and manipulate abstract entities, the learned representations can continuously adapt to the data which avoids the need for task-specific engineering and rigid interfaces.

The first part of this thesis will cover how can we build inductive biases into deep learning models for abstraction which can enable higher level reasoning and generalization for model-based RL. A large challenge is the binding problem which is about dynamically and flexibly combining information distributed throughout the network to represent and relate symbol like entities Greff et al. (2020). We make progress towards this challenge by building a probabilistic entity-centric dynamic latent variable model that can acquire entity representations from raw visual observations without supervision. To ground the entity representations to actual objects in the environment, we frame the binding problem as an inference problem and develop an interactive inference algorithm that uses temporal continuity and interactive feedback to bind information about object properties to the entity variables. This model can be thought of as a differentiable physics simulator with built in priors for reasoning about objects and once learned can then be used for prediction and planning. By factorizing the model in terms of entities, we can generalize to never before seen configurations on challenging block-stacking tasks.

We argue abstractions are a necessary component for generalization and efficient learning. These abstractions can cover both space (representing objects from raw pixels) but also time. For a task such as navigation, it would be difficult for a human to reason about the actions at the level of contracting different muscles each second. We might temporally abstract this problem into temporally extended skills such as taking one step or moving to a waypoint. Reasoning at this higher level abstract space can enable us to compose and chain together skills to solve temporally extended tasks. Prior work in hierarchical reinforcement learning attempts to solve this problem by decomposing the problem into smaller subproblems and learning a set of discrete options to represent skills which can solve each subproblem (Bacon et al., 2017). However, this fixes the number of skills and we also usually have to manually engineer the learning of each skill. Similar to the entity-problem over objects, we want representations that can dynamically and flexibly adapt to the data while still enabling higher levels of abstraction. In the second half of the first part of this thesis, we take a representation learning perspective on the hierarchical RL problem and transform the problem of learning lower layers in a hierarchy into the the problem of learning trajectory-level generative models. This continuous latent representation of trajectories can be used to solve temporally extended and multi-stage problems.

State abstraction and temporal abstraction are examples of how structure in internal models can enable an agent to generalize to many different situations. Millions of years of evolution have created the right structure for this level of generalization in humans. Designing these structures manually requires a great deal of engineering. Is it possible to bypass some of the manual design and automatically learn structure which can generalize? The second part of this thesis will cover how we can remove supervision and manual effort in designing these general purpose models.

For humans we might think that learning happens at two different levels. During a single lifetime, a human learns new skills and adapts to the environment. However at a much longer timescale, the structure of the brain which enables rapid adaptation is being evolved and optimized over time. The inner loop learning process could be described by RL, but the outer loop optimization might require a new kind of meta-learning. We will first discuss how meta-learning can be used to automatically learn structure in our algorithms which can enable good generalization. The main challenges are defining the right language to represent RL algorithms and developing a meta-learning method which can perform the outer loop optimization. We develop a language to represent an RL objective as a computational graph and then introduce an evolutionary based meta-learning method that leverages large scale computational to find reinforcement learning update rules that obtain better performance than manually designed ones. The benefit of our algorithm language is that the learned algorithms are interpretable. Interestingly, we find that the learned algorithms bear resemblance to recently proposed algorithms, and perform a kind of constrained optimization even though such structure was not defined explicitly. Such a method could also be used to meta-learn better abstractions for the examples discussed previously. Abstractions for learning about objects (space) and behaviors (time) might emerge as the dominant structure for the most efficient and general agents.

While model structural priors and meta-optimization are two important ingredients for general RL algorithms, a large missing component is the objective function to optimize. Designing good reward functions for RL is generally challenging and requires manual effort and fine tuning per task. Unsupervised RL objectives could lead to complex and useful behavior without manually designing the reward function. Humans and animals explore their environment and acquire useful skills even in the absence of clear goals, exhibiting intrinsic motivation. An unsupervised objective would provide an agent with an intrinsically-grounded drive to acquire understanding and control of its environment in the absence of an extrinsic reward signal. In the final part of this thesis, we design an embodied agent and general-purpose intrinsic reward signal that leads to the agent controlling partially-observed environments when equipped only with a high-dimensional sensor and no prior knowledge. The compact and general learning objective will be to minimize the entropy of the agent's state visitation distribution using a latent dynamics model. By minimizing entropy or surprise which is analogous to maintaining homeostasis, an agent can learn useful behavior such as cleaning up a room or building a house to survive in an open world environment. This objective induces the agent to both gather information about its environment corresponding to reducing uncertainty and to gain control over its environment, corresponding to reducing the unpredictability of future world states. By optimizing this objective, the agent learns to discover, represent and exercise control of dynamic objects in a variety of partially-observed environments without extrinsic reward.

In this thesis we outline progress along the goal of creating general RL agents with the same problem solving ability as humans. We first introduce methods for building abstraction into latent dynamics models for model based RL, then discuss meta-learning this structure automatically, and finally show how these models can be used for unsupervised RL objectives. This can be summarized as designing the structure, meta-optimization, and the objective function for building general purpose agents:

- In Chapter 3, we make progress toward state abstraction in the form of object level dynamics models which can enable combinatorial generalization from training on a few objects to deploying on completely new scenes with many objects. This work appeared previously as Veerapaneni et al. (2019).

- In Chapter 4 we discuss progress towards temporal abstraction in the form of trajectory level generative models which can enable compositional generalization by chaining together abstract skills to solve temporally extended problems. This work was published previously as Co-Reyes et al. (2018).

- In Chapter 5 we look at how meta-learning can be used to automatically learn RL algorithms which generalize over a wide variety of environments. This work appeared previously as Co-Reyes et al. (2021).

- In Chapter 6, we use latent dynamics models for defining a simple entropy minimizing objective which can lead to complex and useful behavior without manually designed reward functions. This work appeared as Rhinehart et al. (2021).

- Finally, we discuss open challenges and future work in Chapter 7.

Chapter 2

Preliminaries

2.1 Problem Statement

In reinforcement learning, the environment is modeled as a partially observed Markov decision process (POMDP), represented by $M = \{\mathcal{S}, \mathcal{A}, R, P, \Omega, O, \gamma, \rho\}$. This is defined by a state space $\mathcal{S}$ with states $\mathbf{s} \in \mathcal{S}$, action space $\mathcal{A}$ with actions $\mathbf{a} \in \mathcal{A}$, transition dynamics $P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, observation space Ω with observations $\mathbf{o} \in \Omega$, observation function $O(\mathbf{o}_t|\mathbf{s}_t)$, reward function $R(\mathbf{s}_t, \mathbf{a}_t)$, and initial state distribution $\rho(s)$. For a fully observed MDP, the observation function becomes the identity function. The agent is a policy $\pi(\mathbf{a}_t|\mathbf{o}_{\leq t})$ which maps observations to an action distribution. The goal is to find the optimal policy π^* that maximizes the discounted expected return $v^\pi(\rho) = E_{\pi, s_0 \sim \rho} \sum_{t=0}^{T} \gamma^t R(s_t, a_t)$. We measure performance of an agent by both its final return at the end of training and also by its sample efficiency or number of training environment steps needed to achieve a particular return.

We call the training environment distribution that the agent collects experience to learn a policy for model-free RL or dynamics model for model-based RL to be M_{train} while M_{test} will be the distribution of MDPs that we evaluate the trained agent on. To test generalization, we make M_{train} and M_{test} differ in various parameters. For example, in Chapter 3 where we test generalization of our agent to different number of objects, M_{train} will contain scenes with 1-4 objects and M_{test} will contain scenes with 5-9 objects. In Chapter 5, where we test algorithm generalization, M_{train} and M_{test} are radically different and have different action and state spaces, and dynamics functions.

2.2 Generative Latent Dynamics Models

In model-based reinforcement learning, the agent learns the dynamics function $P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ and uses this to plan for an optimal sequence of actions. The reward function is either assumed to be known or also learned. Generally, the state is not observed as in a POMDP and we instead learn a latent state representation z_t which approximates $\mathbf{s}_t$ along with a latent dynamics function $P(z_{t+1}|z_t, a_t)$. There are many ways to learn this representation and most of the work in this thesis will build a generative model $p(o_t|z_t)$ (or $p(o_{\geq t}|z_t, a_{\geq t})$ for sequences) and encoder $q(z_t|o_{\leq t})$. An important component of training this model is based on the framework of variational inference. Variational inference methods use a tractable proxy distribution $q(z \mid o)$ to estimate an intractable posterior $p(z \mid o)$. Given a model with observations o and latent variables z we can decompose the likelihood $p(o)$ in terms of $q(z \mid o)$:

$$\log p(o) = D_{KL}(q(z \mid o) \parallel p(z \mid o)) + \mathcal{L}(o) \tag{2.1}$$

where $\mathcal{L}(o) = \mathbb{E}_q[\log p(o \mid z)] - D_{KL}(q(z \mid o) \parallel p(z))$ is called the evidence lower bound (ELBO). Since KL divergence is non-negative, we obtain the lower bound:

$$\log p(o) \geq \mathbb{E}_q[\log p(o \mid z)] - D_{KL}(q(z \mid o) \parallel p(z)) \tag{2.2}$$

The variational autoencoder is a particular realization of this variational inference procedure. This model can be trained by maximizing the ELBO using standard optimization methods. We refer the reader to Hoffman and Blei (2015); Kingma and Welling (2013) for details.

2.3 Representation Learning and Abstractions for Generalization

We as humans are able to use own internal models of the world to accurately predict what might happen in completely new contexts. We would like to design the learned latent dynamics model as discussed in the previous section to be able to generalize from the training distribution M_{train} to very different test environments M_{test}. We can test generalization for both the supervised learning case where a learned dynamics model trained in M_{train} also has low loss on M_{test}, and also for the reinforcement learning objective we care about which is the return of a policy on M_{test}.

Internal representations or abstractions of the world can enable generalization because they might contain invariances that transfer between M_{train} and M_{test}. By encoding structural priors into our models such as in the observation model $p(o|z)$ or dynamics model $p(z_{t+1}|z_t, a_t)$, we enforce particular abstractions that can generalize better. The next two Chapters will explore how we can build structure into generative latent models that can generalize. In Chapter 3, we encode state abstraction into our model by factorizing z into a discrete set of latent variables $z_{1:k}$ such that each z_i will represent a different entity in the scene. In Chapter 4, we instead encode temporal abstraction into our model by having z represent a temporally extended trajectory segment $p(o_{t:t+h}, |z)$ which we can then compose together into longer horizon plans. Building this structure into our models is complex and Chapter 5, explores if we can instead meta-learn components of the RL algorithm that will automatically generalize. In Chapter 6, we see how we can use these models and the distribution of z to compute an unsupervised objective which leads to useful behavior if optimized in dynamic environments.

Chapter 3

State Abstraction for Combinatorial Generalization

A powerful tool for modeling the complexity of the physical world is to frame this complexity as the composition of simpler entities and processes. For example, the study of classical mechanics in terms of macroscopic objects and a small set of laws governing their motion has enabled not only an explanation of natural phenomena like apples falling from trees but the invention of structures that never before existed in human history, such as skyscrapers. Paradoxically, the creative *variation* of such physical constructions in human society is due in part to the *uniformity* with which human models of physical laws apply to the literal building blocks that comprise such structures – the reuse of the same simpler models that apply to primitive entities and their relations in different ways obviates the need, and cost, of designing custom solutions from scratch for each construction instance.

The challenge of scaling the generalization abilities of learning robots follows a similar characteristic to the challenges of modeling physical phenomena: the complexity of the task space may scale combinatorially with the configurations and number of objects, but if all scene instances share the same set of objects that follow the same physical laws, then transforming the problem of modeling scenes into a problem of modeling objects and the local physical processes that govern their interactions may provide a significant benefit in generalizing to solving novel physical tasks the learner has not encountered before. This is the central hypothesis of this chapter.

We test this hypothesis by defining models for perceiving and predicting raw observations that are themselves compositions of simpler functions that operate locally on *entities* rather than globally on scenes. Importantly, the symmetry that all objects follow the same physical laws enables us to define these learnable entity-centric functions to take as input argument a variable that represents a generic entity, the specific instantiations of which are all processed by the same function. We use the term *entity abstraction* to refer to the abstraction barrier that isolates the abstract *variable*, which the entity-centric function is defined with respect to, from its concrete *instantiation*, which contains information about the appearance and dynamics of an object that modulates the function's behavior.

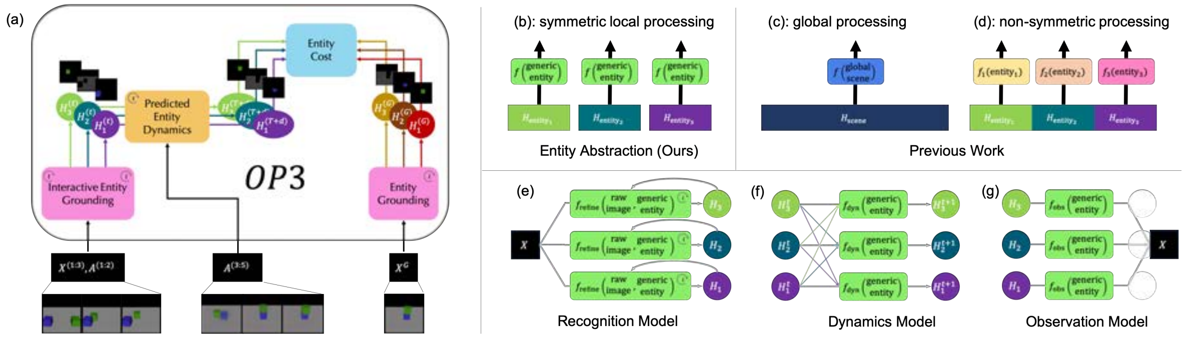

(a)
Entity Cost
Predicted Entity Dynamics
$H_3^{(t)}$ $H_2^{(t)}$ $H_1^{(t)}$
$H_3^{(T+d)}$ $H_2^{(T+d)}$ $H_1^{(T+d)}$
$H_3^{(G)}$ $H_2^{(G)}$ $H_1^{(G)}$
OP3
Interactive Entity Grounding
Entity Grounding
$X^{(1:3)}, A^{(1:2)}$
$A^{(3:5)}$
X^G
(b): symmetric local processing
f(generic entity)
H_{entity_1} H_{entity_2} H_{entity_3}
Entity Abstraction (Ours)
(c): global processing
f(global scene)
H_{scene}
(d): non-symmetric processing
$f_1(\text{entity}_1)$ $f_2(\text{entity}_2)$ $f_3(\text{entity}_3)$
Previous Work
(e)
X
f_{refine}(raw image, generic entity)
H_3 H_2 H_1
Recognition Model
(f)
H_3^t H_2^t H_1^t
f_{dyn}(generic entity)
H_3^{t+1} H_2^{t+1} H_1^{t+1}
Dynamics Model
(g)
f_{obs}(generic entity)
Observation Model

Figure 3.1: **OP3**. **(a)** OP3 can infer a set of entity variables $H_{1:K}^{(T)}$ from a series of interactions (interactive entity grounding) or a single image (entity grounding). OP3 rollouts predict the future entity states $H_{1:K}^{(T+d)}$ given a sequence of actions $a^{(T:T+d)}$. We evaluate these rollouts during planning by scoring these predictions against inferred goal entity-states $H_k^{(G)}$. **(b)** OP3 enforces the **entity abstraction**, factorizing the latent state into *local* entity states, each of which are symmetrically processed with the same function that takes in a *generic entity* as an argument. In contrast, prior work either **(c)** process a global latent state (Hafner et al., 2018) or **(d)** assume a fixed set of entities processed in a permutation-sensitive manner (Finn et al., 2016; Kulkarni et al., 2019; Xu et al., 2018; Watters et al., 2019). **(e-g)** Enforcing the entity-abstraction on modeling the **(f)** dynamics and **(g)** observation distributions of a POMDP, and on the **(e) interactive inference** procedure for grounding the entity variables in raw visual observations. Actions are not shown to reduce clutter.

Defining the observation and dynamic models of a model-based reinforcement learner as neural network functions of abstract entity variables allows for symbolic computation in the space of entities, but the key challenge for realizing this is to ground the values of these variables in the world from raw visual observations. Fortunately, the language of partially observable Markov decision processes (POMDP) enables us to represent these entity variables as latent random state variables in a state-factorized POMDP, thereby transforming the variable binding problem into an inference problem with which we can build upon state-of-the-art techniques in amortized iterative variational inference (Marino et al., 2018a,b; Greff et al., 2019) to use temporal continuity and interactive feedback to infer the posterior distribution of the entity variables given a sequence of observations and actions.

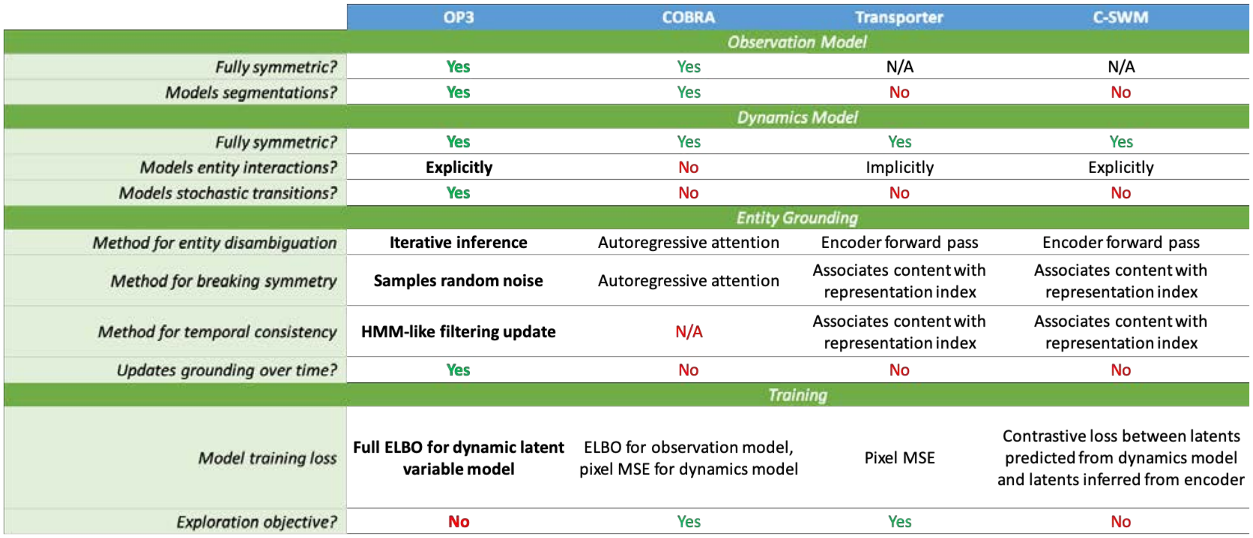
We present a framework for *object-centric perception, prediction, and planning* (OP3), a model-based reinforcement learner that predicts and plans over entity variables inferred via an *interactive inference* algorithm from raw visual observations. Empirically OP3 learns to discover and bind information about actual objects in the environment to these entity variables *without any supervision on what these variables should correspond to.* As all computation within the entity-centric function is *local in scope* with respect to its input entity, the process of modeling the dynamics or appearance of each object is *protected* from the computations involved in modeling other objects, which allows OP3 to generalize to modeling a variable number of objects in a variety of contexts with no re-training.

Contributions: Our conceptual contribution is the use of entity abstraction to integrate graphical models, symbolic computation, and neural networks in a model-based reinforcement learning (RL) agent. This is enabled by our technical contribution: defining models as the composition of locally-scoped entity-centric functions and the interactive inference algorithm for grounding the abstract entity variables in raw visual observations without any supervision on object identity. Empirically, we find that OP3 achieves two to three times greater accuracy than state of the art video prediction models in solving novel single and multi-step block stacking tasks.

3.1 Related Work

Representation learning for visual model-based reinforcement learning: Prior works have proposed learning video prediction models (Wichers et al., 2018; Denton et al., 2017; Lee et al., 2018; Finn et al., 2016) to improve exploration (Oh et al., 2015) and planning (Finn and Levine, 2017) in RL. However, such works and others that represent the scene with a single representation vector (Hafner et al., 2018; Zhang et al., 2018; Mnih et al., 2015; Oh et al., 2016) may be susceptible to the binding problem (Greff et al., 2015; Rosenblatt, 1961) and must rely on data to learn that the same object in two different contexts can be modeled similarly. But processing a disentangled latent state with a single function (Whitney et al., 2016; Chen et al., 2016; Kulkarni et al., 2015, 2019; Goel et al., 2018) or processing each disentangled factor in a permutation-sensitive manner (Lee et al., 2018; Xu et al., 2019; Kulkarni et al., 2019) (1) assumes a fixed number of entities that cannot be dynamically adjusted for generalizing to more objects than in training and (2) has no constraints to enforce that multiple instances of the same entity in the scene be modeled in the same way. For generalization, often the particular arrangement of objects in a scene does not matter so much as what is constant across scenes – properties of individual objects and inter-object relationships – which the inductive biases of these prior works do not capture. The entity abstraction in OP3 enforces symmetric processing of entity representations, thereby overcoming the limitations of these prior works.

Unsupervised grounding of abstract entity variables in concrete objects: Prior works that model entities and their interactions often pre-specify the identity of the entities (Chang et al., 2016; Battaglia et al., 2016; Hamrick et al., 2017; Janner et al., 2018; Narasimhan et al., 2018; Bapst et al., 2010; Ajay et al., 2019), provide additional supervision (Girshick et al., 2014; He et al., 2017; Wang et al., 2018; Yang et al., 2018), or provide additional specification such as segmentations (Janner et al., 2018), crops (Fragkiadaki et al., 2015), or a simulator (Wu et al., 2017; Kansky et al., 2017). Those that do not assume such additional information often factorize the entire scene into pixel-level entities (Santoro et al., 2017; Zambaldi et al., 2018; Du and Narasimhan, 2019), which do not model objects as coherent wholes. None of these works solve the problem of grounding the entities in raw observation, which is crucial for autonomous learning and interaction. OP3 builds upon recently proposed ideas in grounding entity representations via inference on a symmetrically factorized generative model of static (Greff et al., 2015, 2017, 2019) and dynamic (van



	OP3	COBRA	Transporter	C-SWM
	Observation Model			
Fully symmetric?	Yes	Yes	N/A	N/A
Models segmentations?	Yes	Yes	No	No
	Dynamics Model			
Fully symmetric?	Yes	Yes	Yes	Yes
Models entity interactions?	**Explicitly**	No	Implicitly	Explicitly
Models stochastic transitions?	Yes	No	No	No
	Entity Grounding			
Method for entity disambiguation	**Iterative inference**	Autoregressive attention	Encoder forward pass	Encoder forward pass
Method for breaking symmetry	**Samples random noise**	Autoregressive attention	Associates content with representation index	Associates content with representation index
Method for temporal consistency	**HMM-like filtering update**	N/A	Associates content with representation index	Associates content with representation index
Updates grounding over time?	Yes	No	No	No
	Training			
Model training loss	**Full ELBO for dynamic latent variable model**	ELBO for observation model, pixel MSE for dynamics model	Pixel MSE	Contrastive loss between latents predicted from dynamics model and latents inferred from encoder
Exploration objective?	No	Yes	Yes	No

Figure 3.2: **Comparison with other methods.** Unlike other methods, OP3 is a fully probabilistic factorized dynamic latent variable model, giving it several desirable properties. First, OP3 is naturally suited for combinatorial generalization (Battaglia et al., 2018) because it enforces that local properties are invariant to changes in global structure. Because every learnable component of the OP3 operates symmetrically on each entity, including the mechanism that disambiguates entities itself (c.f. COBRA, which uses a learned autoregressive network to disambiguates entities, and Transporter and C-SWMs, which use a forward pass of a convolutional encoder for the global scene, rather than each entity), the weights of OP3 are invariant to changes in the number of instances of an entity, as well as the number of entities in the scene. Second, OP3's recurrent structure makes it straightforward to enforce spatiotemporal consistency, object permanence, and refine the grounding of its entity representations over time with new information. In contrast, COBRA, Transporter, and C-SWMs all model single-step dynamics and do not contain mechanisms for establishing a correspondence between the entity representations predicted from the previous timestep with the entity representations inferred at the current timestep.

Steenkiste et al., 2018) scenes, whose advantage over other methods for grounding (Zhu et al., 2019; Eslami et al., 2016; Burgess et al., 2019; Kosiorek et al., 2018; Watters et al., 2019) is the ability to refine the grounding with new information. In contrast to other methods for binding in neural networks (Levy and Gayler, 2008; Kanerva, 2009; Smolensky, 1990; Vaswani et al., 2017), formulating inference as a mechanism for variable binding allows us to model uncertainty in the values of the variables.

Comparison with similar work: The closest three works to OP3 are the Transporter (Kulkarni et al., 2019), COBRA (Watters et al., 2019), and C-SWMs (Kipf et al., 2019). The Transporter enforces a sparsity bias to learn object keypoints, each represented as a feature vector at a pixel location, and the method's focus on keypoints has the advantage of enabling long-term object tracking, modeling articulated composite bodies such as joints, and scaling to dozens of objects. C-SWMs learn entity representations using a contrastive loss, which has the advantage of overcoming the difficulty in attending to small but relevant features as well as the large model capacity requirements usually characteristic of the pixel reconstructive loss. COBRA uses the autoregressive attention-based MONet (Burgess et al., 2019) architecture to obtain entity representations, which has the advantage of being more computationally efficient and stable to train. Unlike works such as (Greff et al., 2017; Eslami et al., 2016; Burgess et al., 2019; Greff et al., 2019) that infer entity representations from static scenes, these works represent complementary approaches to OP3 (Figure 3.2) for representing dynamic scenes.

Symmetric processing of entities – processing each entity representation with the same function, as OP3 does with its observation, dynamics, and refinement networks – enforces the invariance that local properties are invariant to changes in global structure because it prevents the processing of one entity from being affected by other entities. How symmetric the process is for obtaining these entity representations from visual observation affects how straightforward it is to directly transfer models of a single entity across different global contexts, such as in modeling multiple instances of the same entity in the scene in a similar way or in generalizing to modeling different numbers of objects than in training. OP3 can exhibit this type of zero-shot transfer because the learnable components of its refinement process are fully symmetric across entities, which prevents OP3 from overfitting to the global structure of the scene. In contrast, the KeyNet encoder of the Transporter and the CNN-encoder of C-SWMs associate the *content* of the entity representation with the *index* of that entity in a global representation vector (Figure 3.1d), and this permutation-sensitive mapping entangles the encoding of an entity with the global structure of the scene. COBRA lies in between: it uses a learnable autoregressive attention network to infer segmentation masks, which entangles local object segmentations with global structure but may provide a useful bias for attending to salient objects, and symmetrically encodes entity representations given these masks[1].

As a recurrent probabilistic dynamic latent variable model, OP3 can refine the grounding of its entity representations with new information from raw observations by simply applying a belief update similar to that used in filtering for hidden Markov models. The Transporter, COBRA, and C-SWMs all do not have mechanisms for updating the belief of the entity representations with information from subsequent image frames. Without recurrent structure, such methods rely on the assumption that a single forward pass of the encoder on a static image is sufficient to disambiguate objects, but this assumption is not generally true: objects can pop in and out of occlusion and what constitutes an object depends temporal cues, especially in real world settings. Recurrent structure is built into the OP3 inference update (Appendix 7), enabling OP3 to model object permanence under occlusion and refine its object representations with new information in modeling real world videos (Figure 3.7).

[1] A discussion of the advantages and disadvantages of using an attention-based entity disambiguation method, which MONet and COBRA use, versus an iterative refinement method, which IODINE (Greff et al., 2019) and OP3 use, is discussed in Greff et al. (2019).

3.2 Entity Modeling Problem

Let x^* denote a physical scene and $h^*_{1:K}$ denote the objects in the scene. Let X and A be random variables for the image observation of the scene x^* and the agent's actions respectively. In contrast to prior works (Hafner et al., 2018) that use a single latent variable to represent the state of the scene, we use a set of latent random variables $H_{1:K}$ to represent the state of the objects $h^*_{1:K}$. We use the term *object* to refer to h^*_k, which is part of the physical world, and the term *entity* to refer to H_k, which is part of our model of the physical world. The generative distribution of observations $X^{(0:T)}$ and latent entities $H^{(0:T)}_{1:K}$ from taking T actions $a^{(0:T-1)}$ is modeled as:

$$p\left(X^{(0:T)}, H^{(0:T)}_{1:K} \;\middle|\; a^{(0:T-1)}\right) = p\left(H^{(0)}_{1:K}\right) \prod_{t=1}^{T} p\left(H^{(t)}_{1:K} \;\middle|\; H^{(t-1)}_{1:K}, a^{(t-1)}\right) \prod_{t=0}^{T} p\left(X^{(t)} \;\middle|\; H^{(t)}_{1:K}\right) \tag{3.1}$$

where $p(X^{(t)} \mid H^{(t)}_{1:K})$ and $p(H^{(t)}_{1:K} \mid H^{(t-1)}_{1:K}, A^{(t-1)})$ are the observation and dynamics distribution respectively shared across all timesteps t. Our goal is to build a model that, from simply observing raw observations of random interactions, can generalize to solve novel compositional object manipulation problems that the learner was never trained to do, such as building various block towers during test time from only training to predict how blocks fall during training time.

When all tasks follow the same dynamics we can achieve such generalization with a planning algorithm if given a sequence of actions we could compute $p(X^{(T+1:T+d)} \mid X^{(0:T)}, A^{(0:T+d-1)})$, the posterior predictive distribution of observations d steps into the future. Approximating this predictive distribution can be cast as a variational inference problem (Appdx. A.2) for learning the parameters of an approximate observation distribution $\mathcal{G}(X^{(t)} \mid H^{(t)}_{1:K})$, dynamics distribution $\mathcal{D}(H^{(t)}_{1:K} \mid H^{(t-1)}_{1:K}, A^{(t-1)})$, and a time-factorized recognition distribution $\mathcal{Q}(H^{(t)}_{1:K} \mid H^{(t-1)}_{1:K}, X^{(t)}, A^{(t-1)})$ that maximize the evidence lower bound (ELBO), given by $\mathcal{L} = \sum_{t=0}^{T} \mathcal{L}^{(t)}_{\mathrm{r}} - \mathcal{L}^{(t)}_{\mathrm{c}}$, where

$$\begin{aligned}
\mathcal{L}^t_{\mathrm{r}} &= \mathbb{E}_{h^t_{1:K} \sim q\left(H^t_{1:K} \,\mid\, h^{0:t-1}_{1:K}, x^{1:t}, a^{0:t-1}\right)} \left[\log \mathcal{G}\left(x^t \mid h^t_{1:K}\right)\right] \\
\mathcal{L}^t_{\mathrm{c}} &= \mathbb{E}_{h^{t-1}_{1:K} \sim q\left(H^{t-1}_{1:K} \,\mid\, h^{1:t-2}_{1:K}, x^{1:t-1}, a^{0:t-2}\right)} \left[D_{KL}\left(\mathcal{Q}\left(H^t_{1:K} \mid h^{t-1}_{1:K}, x^t, a^{t-1}\right) \| \mathcal{D}\left(H^t_{1:K} \mid h^{t-1}_{1:K}, a^{t-1}\right)\right)\right].
\end{aligned}$$

The ELBO pushes $\mathcal{Q}$ to produce states of the entities $H_{1:K}$ that contain information useful for not only reconstructing the observations via $\mathcal{G}$ in $\mathcal{L}^{(t)}_{\mathrm{r}}$ but also for predicting the entities' future states via $\mathcal{D}$ in $\mathcal{L}^{(t)}_{\mathrm{c}}$. Sec. 3.3 will next offer our method for incorporating entity abstraction into modeling the generative distribution and optimizing the ELBO.

3.3 Object-Centric Perception, Prediction and Planning

The *entity abstraction* is derived from an assumption about symmetry: that the problem of modeling a dynamic scene of multiple entities can be reduced to the problem of (1) modeling a single entity and its interactions with an *entity-centric* function and (2) applying this function to every entity in the scene. Our choice to represent a scene as a set of entities exposes an avenue for directly encoding such a prior about symmetry that would otherwise not be straightforward with a global state representation.

As shown in Fig. 3.1, a function F that respects the entity abstraction requires two ingredients. The first ingredient (Sec. 3.3) is that $F(H_{1:K})$ is expressed in part as the higher-order operation $\texttt{map}(f, H_{1:K})$ that broadcasts the same entity-centric function $f(H_k)$ to every entity variable H_k. This yields the benefit of automatically transferring learned knowledge for modeling an individual entity to all entities in the scene rather than learn such symmetry from data. As f is a function that takes in a single generic entity variable H_k as argument, the second ingredient (Sec. 3.3) should be a mechanism that binds information from the raw observation X about a particular object h_k^* to the variable H_k.

Entity Abstraction in the Observation and Dynamics Models

The functions of interest in model-based RL are the observation and dynamics models $\mathcal{G}$ and $\mathcal{D}$ with which we seek to approximate the data-generating distribution in equation 3.1.

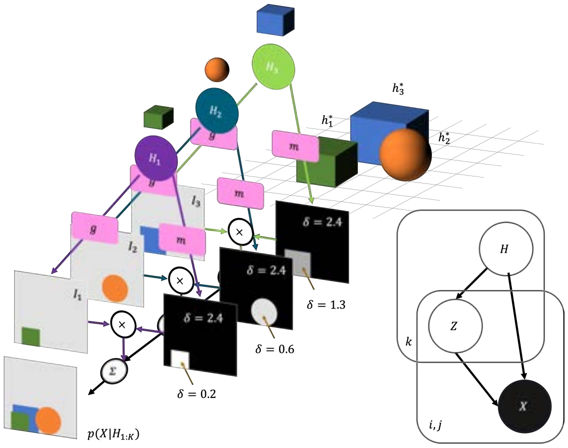


Figure 3.3: **(a)** The observation model $\mathcal{G}$ models an observation image as a composition of sub-images weighted by segmentation masks. The shades of gray in the masks indicate the depth δ from the camera of the object that the sub-image depicts. **(b)** The graphical model of the generative model of observations, where k indexes the entity, and i, j indexes the pixel. Z is the indicator variable that signifies whether an object's depth at a pixel is the closest to the camera.

Observation Model: The observation model $\mathcal{G}(X \mid H_{1:K})$ approximates the distribution $p(X \mid H_{1:K})$, which models how the observation X is caused by the combination of entities $H_{1:K}$. We enforce the entity abstraction in $\mathcal{G}$ (in Fig. 3.1g) by applying the same entity-centric function $g(X \mid H_k)$ to each entity H_k, which we can implement using a mixture model at each pixel (i, j):

$$\mathcal{G}\left(X_{(ij)} \mid H_{1:K}\right) = \sum_{k=1}^{K} m_{(ij)}\left(H_k\right) \cdot g\left(X_{(ij)} \mid H_k\right), \tag{3.2}$$

where g computes the mixture components that model how each individual entity H_k is independently generated, combined via mixture weights m that model the entities' relative depth from the camera, the derivation of which is in Appdx. A.1.

Dynamics Model: The dynamics model $\mathcal{D}(H'_{1:K} \mid H_{1:K}, A)$ approximates the distribution $p(H'_{1:K} \mid H_{1:K}, A)$, which models how an action A intervenes on the entities $H_{1:K}$ to produce their future values $H'_{1:K}$. We enforce the entity abstraction in $\mathcal{D}$ (in Fig. 3.1f) by applying the same entity-centric function $\mathscr{d}(H'_k \mid H_k, H_{[\neq k]}, A)$ to each entity H_k, which reduces the problem of modeling how an action affects a scene with a combinatorially large space of object configurations to the problem of simply modeling how an action affects a single *generic* entity H_k and its interactions with the list of other entities $H_{[\neq k]}$. Modeling the action as an finer-grained intervention on a *single* entity rather than the entire scene is a benefit of using local representations of entities rather than global representations of scenes.

However, at this point we still have to model the combinatorially large space of *interactions* that a single entity could participate in. Therefore, we can further enforce a *pairwise* entity abstraction on $\mathscr{d}$ by applying the same *pairwise* function $\mathscr{d}_{oo}(H_k, H_i)$ to each entity pair (H_k, H_i), for $i \in [\neq k]$. Omitting the action to reduce clutter (the full form is written in Appdx. A.6), the structure of the $\mathcal{D}$ therefore follows this form:

$$\mathcal{D}\left(H'_{1:K} \mid H_{1:K}\right) = \prod_{k=1}^{K} \mathscr{d}\left(H'_k \mid H_k, H_k^{\text{interact}}\right), \text{ where } H_k^{\text{interact}} = \sum_{i \neq k}^{K} d_{oo}\left(H_i, H_k\right). \tag{3.3}$$

The entity abstraction therefore provides the flexibility to scale to modeling a variable number of objects by solely learning a function $\mathscr{d}$ that operates on a single generic entity and a function $\mathscr{d}_{oo}$ that operates on a single generic entity pair, both of which can be re-used for across all entity instances.

Interactive Inference for Binding Object Properties to Latent Variables

For the observation and dynamics models to operate from raw pixels hinges on the ability to bind the properties of specific physical objects $h^*_{1:K}$ to the entity variables $H_{1:K}$. For latent variable models, we frame this variable binding problem as an inference problem: binding information about $h^*_{1:K}$ to $H_{1:K}$ can be cast as a problem of inferring the parameters of $p(H^{(0:T)} \mid x^{(0:T)}, a^{(0:T-1)})$, the posterior distribution of $H_{1:K}$ given a sequence of interactions. Maximizing the ELBO in Sec. 3.2 offers a method for learning the parameters of the observation and dynamics models while simultaneously learning an approximation to the posterior $q(H^{(0:T)} \mid x^{(0:T)}, a^{(0:T-1)}) = \prod_{t=0}^{T} \mathcal{Q}(H^{(t)}_{1:K} \mid H^{(t-1)}_{1:K}, x^{(t)}, a^{(t)})$, which we have chosen to factorize into a per-timestep recognition distribution $\mathcal{Q}$ shared across timesteps. We also choose to enforce the entity abstraction on the process that computes the recognition

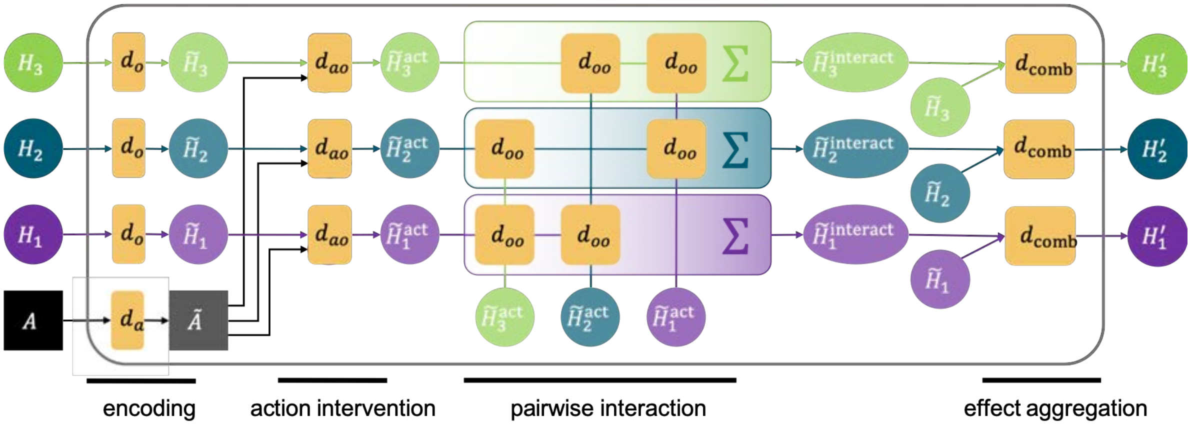


Figure 3.4: The **dynamics model** $\mathcal{D}$ models the time evolution of every object by symmetrically applying the function $\mathscr{d}$ to each object. For a given object, d models the individual dynamics of that object (d_o), embeds the action vector (d_a), computes the action's effect on that object (d_{ao}), computes each of the other objects' effect on that object (d_{oo}), and aggregates these effects together (d_{comb}).

distribution $\mathcal{Q}$ (in Fig. 3.1e) by decomposing it into a recognition distribution q applied to each entity:

$$\mathcal{Q}\left(H_{1:K}^{(t)} \,|\, h_{1:K}^{(t-1)}, x^{(t)}, a^{(t)}\right) = \prod_{k=1}^{K} q\left(H_k^{(t)} \,|\, h_k^{(t-1)}, x^{(t)}, a^{(t)}\right). \tag{3.4}$$

Whereas a neural network encoder is often used to approximate the posterior (Hafner et al., 2018; Xu et al., 2018; Kulkarni et al., 2019), a single forward pass that computes q in parallel for each entity is insufficient to break the symmetry for dividing responsibility of modeling different objects among the entity variables (Zhang et al., 2019) because the entities do not have the opportunity to communicate about which part of the scene they are representing.

We therefore adopt an *iterative inference* approach (Marino et al., 2018a) to compute the recognition distribution $\mathcal{Q}$, which has been shown to break symmetry among modeling objects in static scenes (Greff et al., 2019). Iterative inference computes the recognition distribution via a *procedure*, rather than a single forward pass of an encoder, that iteratively refines an initial guess for the posterior parameters $\lambda_{1:K}$ by using gradients from how well the generative model is able to predict the observation based on the current posterior estimate. The initial guess provides the noise to break the symmetry.

For scenes where position and color are enough for disambiguating objects, a static image may be sufficient for inferring q. However, in interactive environments disambiguating objects is more underconstrained because what constitutes an object depends on the goals of the agent. We therefore incorporate actions into the amortized varitional filtering framework (Marino et al., 2018b) to develop an *interactive inference* algorithm (Appdx. A.4 and Fig. 3.5) that uses temporal continuity and interactive feedback to disambiguate objects. Another benefit

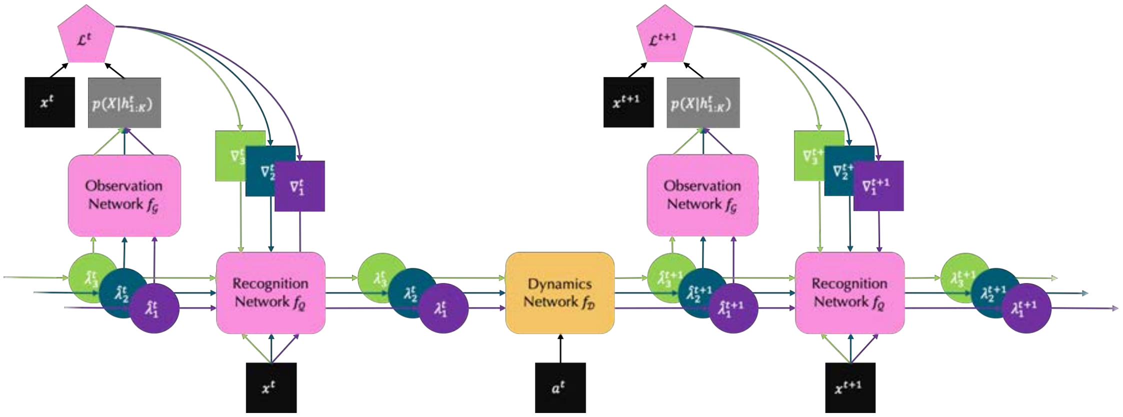


Figure 3.5: **Amortized interactive inference** alternates between refinement (pink) and dynamics (orange) steps, iteratively updating the belief of $\lambda_{1:K}$ over time. $\hat{\lambda}$ corresponds to the output of the dynamics network, which serves as the initial estimate of λ that is subsequently refined by $f_\mathcal{G}$ and $f_\mathcal{Q}$. ∇ denotes the feedback used in the refinement process, which includes gradient information and auxiliary inputs (Appdx. A.4).

of enforcing entity abstraction is that preserving temporal consistency on entities comes for free: information about each object remains bound to its respective H_k through time, mixing with information about other entities only through explicitly defined avenues, such as in the dynamics model.

Training at Different Timescales

The variational parameters $\lambda_{1:K}$ are the interface through which the neural networks f_g, $f_\mathscr{d}$, f_q that respectively output the distribution parameters of $\mathcal{G}$, $\mathcal{D}$, and $\mathcal{Q}$ communicate. For a *particular* dynamic scene, the execution of interactive inference optimizes the variational parameters $\lambda_{1:K}$. *Across* scene instances, we train the weights of f_g, $f_\mathscr{d}$, f_q by backpropagating the ELBO through the entire inference procedure, spanning multiple timesteps. OP3 thus learns at three different timescales: the variational parameters learn (1) across M steps of inference within a single timestep and (2) across T timesteps within a scene instance, and the network weights learn (3) across different scene instances.

Beyond next-step prediction, we can directly train to compute the posterior predictive distribution $p(X^{(T+1:T+d)} \mid x^{(0:T)}, a^{(0:T+d)})$ by sampling from the approximate posterior of $H^{(T)}_{1:K}$ with $\mathcal{Q}$, rolling out the dynamics model $\mathcal{D}$ in latent space from these samples with a sequence of d actions, and predicting the observation $X^{(T+d)}$ with the observation model $\mathcal{G}$. This approach to action-conditioned video prediction predicts future observations directly from observations and actions, but with a bottleneck of K time-persistent entity-variables with which the dynamics model $\mathcal{D}$ performs symbolic relational computation.

Object-Centric Planning

OP3 rollouts, computed as the posterior predictive distribution, can be integrated into the standard visual model-predictive control (Finn and Levine, 2017) framework. Since interactive inference grounds the entities $H_{1:K}$ in the actual objects $h^*_{1:K}$ depicted in the raw observation, this grounding essentially gives OP3 access to a *pointer* to each object, enabling the rollouts to be in the space of entities and their relations. These pointers enable OP3 to not merely predict in the space of entities, but give OP3 access to an *object-centric action space*: for example, instead of being restricted to the standard `(pick_xy, place_xy)` action space common to many manipulation tasks, which often requires biased picking with a scripted policy (Levine et al., 2018; Kalashnikov et al., 2018), these pointers enable us to compute a mapping (Appdx. A.7) between `entity_id` and `pick_xy`, allowing OP3 to automatically use a `(entity_id, place_xy)` action space without needing a scripted policy.

Generalization to Various Tasks

We consider tasks defined in the same environment with the same physical laws that govern appearance and dynamics. Tasks are differentiated by goals, in particular goal configurations of objects. Building good cost functions for real world tasks is generally difficult (Fu et al., 2018) because the underlying state of the environment is always unobserved and can only be modeled through modeling observations. However, by representing the environment state as the state of its entities, we may obtain finer-grained goal-specification without the need for manual annotations (Ebert et al., 2018). Having rolled out OP3 to a particular timestep, we construct a cost function to compare the predicted entity states $H^{(P)}_{1:K}$ with the entity states $H^{(G)}_{1:K}$ inferred from a goal image by considering pairwise distances between the entities, another example of enforcing the pairwise entity abstraction. Letting S' and S denote the set of goal and predicted entities respectively, we define the form of the cost function via a composition of the task specific distance function c operating on entity-pairs:

$$C\left(H^{(G)}_{1:K}, H^{(P)}_{1:K}\right) = \sum_{a \in S'} \min_{b \in S} c\left(H^{(G)}_a, H^{(P)}_b\right), \tag{3.5}$$

in which we pair each goal entity with the closest predicted entity and sum over the costs of these pairs. Assuming a single action suffices to move an object to its desired goal position, we can greedily plan each timestep by defining the cost to be $\min_{a \in S', b \in S} \; c(H^{(G)}_a, H^{(P)}_b)$, the pair with minimum distance, and removing the corresponding goal entity from further consideration for future planning.

3.4 Experiments

Our experiments aim to study to what degree entity abstraction improves generalization, planning, and modeling. Sec. 3.4 shows that from only training to predict how objects fall, OP3 generalizes to solve various novel block stacking tasks with two to three times better accuracy than a state-of-the-art video prediction model. Sec. 3.4 shows that OP3 can plan for multiple steps in a difficult multi-object environment. Sec. 3.4 shows that OP3 learns to ground its abstract entities in objects from real world videos.

Combinatorial Generalization without Object Supervision

We first investigate how well OP3 can learn object-based representations without additional object supervision, as well as how well OP3's factorized representation can enable combinatorial generalization for scenes with many objects.

Domain: In the MuJoCo (Todorov et al., 2012) block stacking task introduced by Janner et al. (2018) for the O2P2 model, a block is raised in the air and the model must predict the steady-state effects of dropping the block on a surface with multiple objects, which implicitly requires modeling the effects of gravity and collisions. The agent is never trained to stack blocks, but is tested on a suite of tasks where it must construct block tower specified by a goal image. Janner et al. (2018) showed that an object-centric model with access to *ground truth* object segmentations can solve these tasks with about 76% accuracy. We now consider whether OP3 can do better, but *without any supervision on object identity.*

SAVP	O2P2	OP3 (ours)
24%	76%	**82%**

Table 3.1: Accuracy (%) of block tower builds by the SAVP baseline, the O2P2 oracle, and our approach. O2P2 uses image segmentations whereas OP3 uses only raw images as input.

# Blocks	SAVP	OP3 (xy)	OP3 (entity)
1	54%	73%	**91%**
2	28%	55%	**80%**
3	28%	41%	**55%**

Table 3.2: Accuracy (%) of multi-step planning for building block towers. (xy) means `(pick_xy, place_xy)` action space while (entity) means `(entity_id, place_xy)` action space.

Setup: We train OP3 on the same dataset and evaluate on the same goal images as Janner et al. (2018). While the training set contains up to five objects, the test set contains up to nine objects, which are placed in specific structures (bridge, pyramid, etc.) not seen during training. The actions are optimized using the cross-entropy method (CEM) (Rubinstein and Kroese, 2004), with each sampled action evaluated by the greedy cost function described in Sec. 3.3. Accuracy is evaluated using the metric defined by Janner et al. (2018), which checks that all blocks are within some threshold error of the goal.

Results: The two baselines, SAVP (Lee et al., 2018) and O2P2, represent the state-of-the-art in video prediction and symmetric object-centric planning methods, respectively. SAVP models objects with a fixed number of convolutional filters and does not process entities symmetrically. O2P2 does process entities symmetrically, but requires access to ground truth object segmentations. As shown in Table 3.1, OP3 achieves better accuracy than O2P2, even without any ground truth supervision on object identity, possibly because grounding the entities in the raw image may provide a richer contextual representation than encoding each entity separately without such global context as O2P2 does. OP3 achieves three times the accuracy of SAVP, which suggests that symmetric modeling of entities is enables the flexibility to transfer knowledge of dynamics of a single object to novel scenes with different configurations heights, color combinations, and numbers of objects than those from the training distribution. Fig. A.1 and Fig. A.2 in the Appendix show that, by grounding its entities in objects of the scene through inference, OP3's predictions isolates only one object at a time without affecting the predictions of other objects.

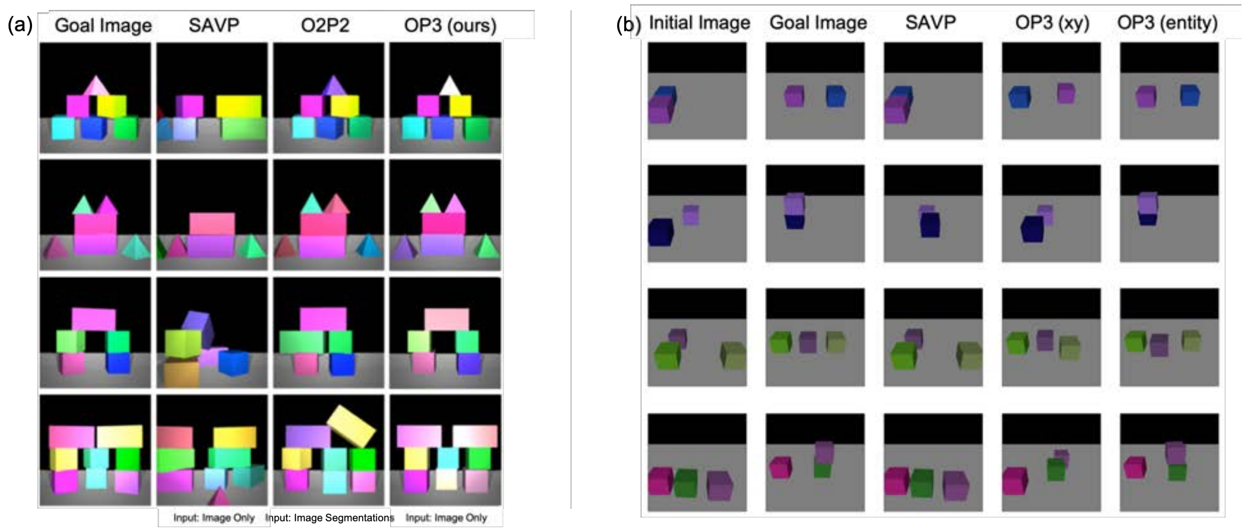


Figure 3.6: **(a)** In the block stacking task from (Janner et al., 2018) with single-step greedy planning, OP3's generalizes better than both O2P2, an oracle model with access to image segmentations, and SAVP, which does not enforce entity abstraction. **(b)** OP3 exhibits better multi-step planning with objects already present in the scene. By planning with MPC using random pick locations (SAVP and OP3 (xy)), the sparsity of objects in the scene make it rare for random pick locations to actually pick the objects. However, because OP3 has access to pointers to the latent entities, we can use these to automatically bias the pick locations to be at the object location, without any supervision (OP3 (entity)).

Multi-Step Planning

The goal of our second experiment is to understand how well OP3 can perform multi-step planning by manipulating objects already present in the scene. We modify the block stacking task by changing the action space to represent a picking and dropping location. This requires reasoning over extended action sequences since moving objects out of place may be necessary.

Goals are specified with a goal image, and the initial scene contains all of the blocks needed to build the desired structure. This task is more difficult because the agent may have to move blocks out of the way before placing other ones which would require multi-step planning. Furthermore, an action only successfully picks up a block if it intersects with the block's outline, which makes searching through the combinatorial space of plans a challenge.

As stated in Sec. 3.3, having a pointer to each object enables OP3 to plan in the space of entities. We compare two different action spaces `(pick_xy, place_xy)` and `(entity_id, place_xy)` to understand how automatically filtering for pick locations at actual locations of objects enables better efficiency and performance in planning. Details for determining the `pick_xy` from `entity_id` are in appendix A.7.

Results: We compare with SAVP, which uses the `(pick_xy, place_xy)` action space. With this standard action space (Table 3.2) OP3 achieves between 1.5-2 times the accuracy of SAVP. This performance gap increases to 2-3 times the accuracy when OP3 uses the `(entity_id, place_xy)` action space. The low performance of SAVP with only two blocks highlights the difficulty of such combinatorial tasks for model-based RL methods, and highlights the both the generalization and localization benefits of a model with entity abstraction. Fig. 3.6b shows that OP3 is able to plan more efficiently, suggesting that OP3 may be a more effective model than SAVP in modeling combinatorial scenes. Fig. 3.7a shows the execution of interactive inference during training, where OP3 alternates between four refinement steps and one prediction step. Notice that OP3 infers entity representations that decompose the scene into coherent objects and that entities that do not model objects model the background. We also observe in the last column ($t = 2$) that OP3 predicts the appearance of the green block even though the green block was partially occluded in the previous timestep, which shows its ability to retain information across time.

Real World Evaluation

The previous tasks used simulated environments with monochromatic objects. Now we study how well OP3 scales to real world data with cluttered scenes, object ambiguity, and occlusions. We evaluate OP3 on the dataset from Ebert et al. (2018) which contains videos of a robotic arm moving cloths and other deformable and multipart objects with varying textures.

We evaluate qualitative performance by visualizing the object segmentations and compare against vanilla IODINE, which does not incorporate an interaction-based dynamics model into the inference process. Fig. 3.7b highlights the strength of OP3 in preserving temporal continuity and disambiguating objects in real world scenes. While IODINE can disambiguate monochromatic objects in static images, we observe that it struggles to do more than just color segmentation on more complicated images where movement is required to disambiguate objects. In contrast, OP3 is able to use temporal information to obtain more accurate segmentations, as seen in Fig. 3.7b where it initially performs color segmentation by grouping the towel, arm, and dark container edges together, and then by observing the effects of moving the arm, separates these entities into different groups.

3.5 Discussion

We have shown that enforcing the entity abstraction in a model-based reinforcement learner improves generalization, planning, and modeling across various compositional multi-object

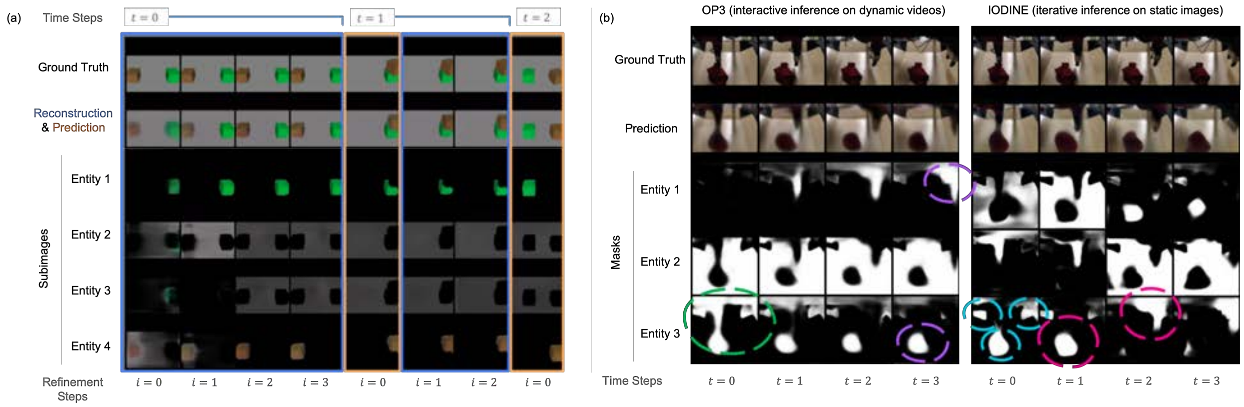


Figure 3.7: Visualization of interactive inference for block-manipulation and real-world videos (Ebert et al., 2018). Here, OP3 interacts with the objects by executing pre-specified actions in order to disambiguate objects already present in the scene by taking advantage of temporal continuity and receiving feedback from how well its prediction of how an action affects an object compares with the ground truth result. **(a)** OP3 does four refinement steps on the first image, and then 2 refinement steps after each prediction. **(b)** We compare OP3, applied on dynamic videos, with IODINE, applied independently to each frame of the video, to illustrate that using a dynamics model to propagate information across time enables better object disambiguation. We observe that initially, both OP3 (green circle) and IODINE (cyan circles) both disambiguate objects via color segmentation because color is the only signal in a static image to group pixels. However, we observe that as time progresses, OP3 separates the arm, object, and background into separate latents (purple) by using its currently estimates latents predict the next observation and comparing this prediction with the actually observed next observation. In contrast, applying IODINE on a per-frame basis does not yield benefits of temporal consistency and interactive feedback (red).

tasks. In particular, enforcing the entity abstraction provides the learner with a pointer to each entity variable, enabling us to define functions that are local in scope with respect to a particular entity, allowing knowledge about an entity in one context to directly transfer to modeling the same entity in different contexts. In the physical world, entities are often manifested as objects, and generalization in physical tasks such as robotic manipulation often may require symbolic reasoning about objects and their interactions. However, the general difficulty with using purely symbolic, abstract representations is that it is unclear how to continuously update these representations with more raw data. OP3 frames such symbolic entities as random variables in a dynamic latent variable model and infers and refines the posterior of these entities over time with neural networks. This suggests a potential bridge to connect abstract symbolic variables with the noisy, continuous, high-dimensional physical world, opening a path to scaling robotic learning to more combinatorially complex tasks.

Chapter 4

Temporal Abstraction for Compositional Generalizaton

Deep reinforcement learning (RL) algorithms can learn complex skills from raw observations Mnih et al. (2015); Levine et al. (2016); Silver et al. (2016). However, domains that involve temporally extended tasks and extremely delayed or sparse rewards can pose a tremendous challenge for standard methods. A longtime goal in RL has been to develop effective hierarchy induction methods that can acquire temporally extended lower-level primitives, which can then be built upon by a higher level policy that operates at a coarser level of temporal abstraction Sutton et al. (1999); Dayan and Hinton (1992); Dietterich (1998); Parr and Russell (1997). A higher-level policy that is provided with temporally extended and intelligent behaviors can reason at a higher level of abstraction and solve more temporally-extended tasks. Furthermore, the same lower-level skills could be reused to accomplish multiple tasks efficiently.

Prior work has proposed to acquire discrete sets of lower-level skills through hand-specification of objectives or bottlenecks Florensa et al. (2017); Frans et al. (2017); Sutton et al. (1999) and top-down training of hierarchically-organized policies Dayan and Hinton (1992); Vezhnevets et al. (2017). Requiring prior knowledge and hand-specification restricts the generality of the method, while purely top-down training suffers from challenging optimization and exploration and limits the reusability of lower-level skills, providing a solution to just one task. Furthermore, the top-level meta-policy must still be trained with reinforcement learning for each task, and while this tends to be more efficient than learning from scratch if the skills are useful, it still requires considerable time and experience collection. Several works have also proposed "bottom up" training of lower-level skills using unsupervised objectives Bacon et al. (2017); Gregor et al. (2016), but such methods either also require hand-specifying some prior knowledge, or learn discrete skills that may not necessarily be sufficient to solve the higher level task.

In this chaper, we propose a novel hierarchical reinforcement learning algorithm (SeCTAR) that uses a bottom up approach to learn continuous representations for trajectories, without the explicit need for hand-specification or subgoal information. Our work builds on two main ideas: first, we propose to build a continuous latent space of skills, rather than a discrete set of behaviors or options, and second, we propose to use a probabilistic latent variable model that simultaneously learns to produce skills in the world and predict their outcomes. By providing a higher-level controller with a continuous space of behaviors, it can exercise considerable control, without being restricted to a small discrete set of primitives. At the same time, since the behaviors are temporally extended, the higher-level policy still benefits from temporal abstraction. Furthermore, by training a model that both acquires a set of skills and predicts their outcomes, we can avoid needing to train a higher-level policy with reinforcement learning, and directly use these outcome predictions to perform model-based control at the higher level. This results in a hybrid model-free and model-based method, where the behaviors that actually interact with the environment are trained in model-free fashion, while the higher-level behavior is model-based. This also neatly addresses one of the major shortcomings of model-based reinforcement learning, which is the difficulty of accurately predicting low-level physical events at a fine temporal resolution. Since the predictions only need to accurately reflect the outcomes of closed-loop and temporally extended behaviors, they are substantially easier than low-level modeling of environment dynamics, while still being conducive to effective higher-level planning.

Our model is based on a trajectory-level variational autoencoder (VAE) Kingma and Welling (2013). The continuous latent space of behaviors is constructed by learning to embed and generate trajectories obtained via a fully unsupervised exploration objective. In addition to learning to generate the state sequences along these trajectories, the model simultaneously learns to reproduce those trajectories in the environment via a policy conditioned on the VAE latent variable. In this way, the latent-conditioned policy aims to "imitate" the VAE decoder. The fact that the latent-conditioned policy and the VAE decoder are representing the same behavior allows us to treat the decoder as a model of the closed loop behavior of the policy. This allows us to use the decoder to plan in the latent space by sampling latents and simulating their corresponding trajectories. We can then choose the best latents that solve the task and execute the plan with the latent-conditioned policy.

The main contribution of this work is a hierarchical reinforcement learning algorithm that acquires a continuous low-level latent space of skills, together with a predictive model that can predict the outcomes of those skills, which can be used to carry out more complex higher-level tasks. We propose a novel training procedure for this model, and show that higher-level extended tasks can be performed directly with model-based planning, without any additional reinforcement learning to learn a high level policy. Our experimental evaluation demonstrates that this approach can be used to accomplish a variety of delayed and sparse reward tasks, including interaction with objects and waypoint navigation, while outperforming reinforcement learning methods such as TRPO Schulman et al. (2015), exploration driven methods such as VIME Houthooft et al. (2016) as well as prior work on hierarchical reinforcement learning such as FeUdal Networks Vezhnevets et al. (2017) and option critic Bacon et al. (2017). All our results, videos, and experimental details can be found at `https://sites.google.com/view/sectar`

4.1 Self-Consistent Trajectory Autoencoder

In this work, our aim is to perform long-horizon planning by learning latent representations over trajectories. Given a task with a long horizon H, we define trajectories in the context of SeCTAR as sequences of states $[s_0, s_1, ..., s_T]$ of length T, where $T < H$. Each complete episode in the MDP M (of length H) may be composed of several of these shorter trajectories. We hypothesize that building representations for these trajectories will allow us to reason more effectively over the entire horizon.

To that end, we introduce the self-consistent trajectory autoencoder (SeCTAR) to acquire latent representations of trajectories. The SeCTAR model is based on the variational autoencoder, but with two decoders: the state decoder, which decodes latent variables directly into sequences of states, and the policy decoder, which is a latent-conditioned policy capable of generating the encoded trajectory when executed in the environment. This two-headed model allows the state decoder to be a predictive model of the behavior that a policy decoder can execute in the environment.

The latent representations learned by SeCTAR can be used for planning over long episodes, by reasoning at the level of latent variables (representing extended state sequences) rather than at the level of individual states and actions. We will introduce a model-based planning algorithm based on SeCTAR in Section 4.1 to perform planning in the latent space to solve long horizon tasks.

Solving tasks with sparse rewards and long horizons requires effective exploration. We show that we can improve the exploration behavior needed for hierarchical reasoning, using the SeCTAR model and an entropy based exploration objective. This results in an iterative training procedure described in Section 4.1, which we find important for performing hierarchical tasks. We first introduce the SeCTAR model, describe how it can be trained, and show its usefulness for hierarchical planning. We then describe how we can perform exploration in the loop to improve performance.

Graphical Model

We consider the problem of learning latent representations of trajectories $[s_0, s_1, \cdots, s_T]$. We begin by extending the framework of VAEs (Kingma and Welling, 2013), with trajectories τ as the observation, a trajectory-level encoder $q_\phi(z \mid \tau)$, and a *state decoder* $p_{\theta_{SD}}(\tau \mid z)$. The graphical model representing this model is shown in Fig 4.1. We will discuss the training procedure of this model in Section 4.1. A trained model can generate sequences of states by sampling a latent variable z and decoding using $p_{\theta_{SD}}(\tau \mid z)$.

While sequences of states are predictive of behavior, they do not allow us to act directly in the real world: the states may not be fully dynamically consistent, and we do not know the actions that would realize them. To enable our model to actually act in the world and visit states that are predicted by the state decoder $p_{\theta_{SD}}(\tau \mid z)$, we introduce a second decoder – the policy decoder $p_{\theta_{PD}}(a \mid s, z)$. The policy decoder cannot generate the entire trajectory

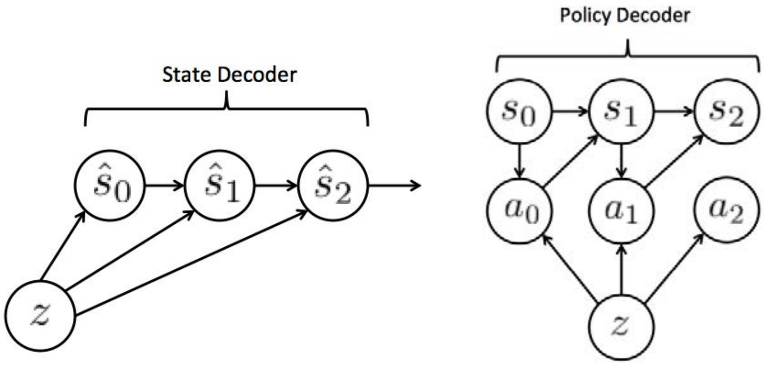


Figure 4.1: Graphical models representing the state and policy decoders. The state decoder (shown on the left) directly generates a trajectory conditioned on the latent variable, while the policy decoder generates a trajectory by conditioning a policy which is rolled out in the environment. As is standard in model-free RL, the environment dynamics are unknown, so the policy decoder must be trained by sampling rollouts.

directly like the state decoder, but has to actually act sequentially in the environment to produce trajectories. We train this policy decoder to produce behavior in the environment consistent with the predictions made by the state decoder by minimizing the KL divergence between the distribution over state sequences under the state decoder and the policy decoder. Both the state and policy decoder are trained jointly with the recognition network $q_\phi(z \mid \tau)$.

We describe the model assuming that the trajectory data τ is observed and fixed, which allows us to use maximum likelihood estimation to train the model. In Section 4.1, we will describe how we can improve trajectory distributions by alternating between model fitting and entropy based exploration, in order to generate better τ data automatically.

Training SeCTAR with Variational Inference

We can train the latent variable model described in Section 4.1 with a procedure that is similar to VAE training. Unlike a standard VAE, we must also account for the relationship between the policy decoder and state decoder. We want to maximize the likelihood of the trajectory data $p(\tau)$ under the state decoder for different z, while also ensuring that the state and policy decoder are consistent, minimizing the KL divergence between them.

$$\begin{aligned} &\max && \log p(\tau) \\ &\text{subject to} && \mathbb{E}_{q_\phi}[D_{KL}(p_{\theta_{PD}}(\tau \mid z) \parallel p_{\theta_{SD}}(\tau \mid z))] = 0 \end{aligned}$$

By applying the KL divergence as a penalty on the likelihood, we can write an unconstrained objective as

$$\max_{\theta_{SD},\theta_{PD},\phi} \log p(\tau) - \lambda \mathbb{E}_{q_\phi}[D_{KL}(p_{\theta_{PD}}(\tau \mid z) \parallel p_{\theta_{SD}}(\tau \mid z))] \tag{4.1}$$

Introducing the evidence lower bound (ELBO) in place of the marginal likelihood $\log(p(\tau))$, we obtain

$$\begin{aligned}\log p(\tau) - \lambda\mathbb{E}_{q_\phi}[D_{KL}(p_{\theta_{PD}}(\tau \mid z) \parallel p_{\theta_{SD}}(\tau \mid z)] \\ \geq \mathbb{E}_{q_\phi}[\log p_{\theta_{SD}}(\tau \mid z))] - D_{KL}(q_\phi(z \mid \tau) \parallel p(z)) + \\ \lambda\big[\mathbb{E}_{q_\phi, p_{\theta_{PD}}(\tau|z)}[\log p_{\theta_{SD}}(\tau \mid z)] + \mathcal{H}(p_{\theta_{PD}}(\tau \mid z))\big]\end{aligned} \tag{4.2}$$

Intuitively, this corresponds to optimizing the ELBO while constraining the state and policy decoders to be mutually consistent. This induces the state decoder to fit the observed data and the policy decoder to match the state decoder while also maximizing the entropy of the policy's action distribution (as in maximum entropy RL Schulman et al. (2017a)).

We parameterize our encoder $q_\phi(z \mid \tau)$ and state decoder $p_{\theta_{SD}}(\tau \mid z)$ with recurrent neural networks, since they operate on sequences of states, while the policy decoder is a feedforward neural network, as shown in Figure 4.2. Since SeCTAR will be used for generating multiple trajectories sequentially, each starting in a different state we condition the state decoder on the initial state s_0, allowing SeCTAR to generalize behavior across different initial states. The state decoder is completely differentiable and can be trained with backpropagation, but the policy decoder interacts with the environment's non-differentiable dynamics, so we cannot train it with backpropagation through time, instead requiring reinforcement learning.

Optimization of the objective in Equation 4.2, with respect to each of the parameters $\theta_{SD}, \theta_{PD}, \phi$ yields the different components of our model training.

State Decoder: Optimizing the objective with respect to θ_{SD} maximizes the terms $\mathbb{E}_q[\log p_{\theta_{SD}}(\tau \mid z)] + \lambda\mathbb{E}_{q,p_{\theta_{PD}}(\tau'|z)}[\log p_{\theta_{SD}}(\tau' \mid z)]$. The first term encourages the state decoder to maximize the likelihood of the observed data, while the second term encourages the state decoder to match the policy decoder. In practice, we didn't find a significant advantage in optimizing the second term with respect to θ_{SD} so it is omitted from our implementation. Since θ_{SD} is differentiable this objective can be directly optimized using backpropagation.

Policy Decoder: Optimizing with respect to θ_{PD} maximizes the terms $\lambda\big[\mathbb{E}_{q,p_{\theta_{PD}}(\tau'|z)}[\log p_{\theta_{SD}}(\tau' \mid z)] + \mathcal{H}(p_{\theta_{PD}}(\tau \mid z))\big]$. The first term encourages samples drawn from the policy decoder to maximize the likelihood under the state decoder, while the second term is an entropy regularization. Since $p_{\theta_{PD}}(\tau \mid z)$ is non differentiable, we use reinforcement learning to optimize this objective with reward computed by trajectory likelihood under the state decoder, regularized with an entropy objective. In practice, trajectory data from the environment actually consists of sequences of both states *and* actions. We find that pretraining the policy decoder with behavior cloning to match the actions in the trajectory provides a good initialization for subsequent finetuning with RL.

To optimize this model, we sample a batch of trajectories from the current set of training trajectories and alternate between training the state decoder with backpropagation with the standard VAE loss and training the policy decoder by initializing with behavior cloning and doing RL finetuning with the reward function described above using PPO Schulman et al. (2017b), backpropagating gradients into the encoder in both cases.

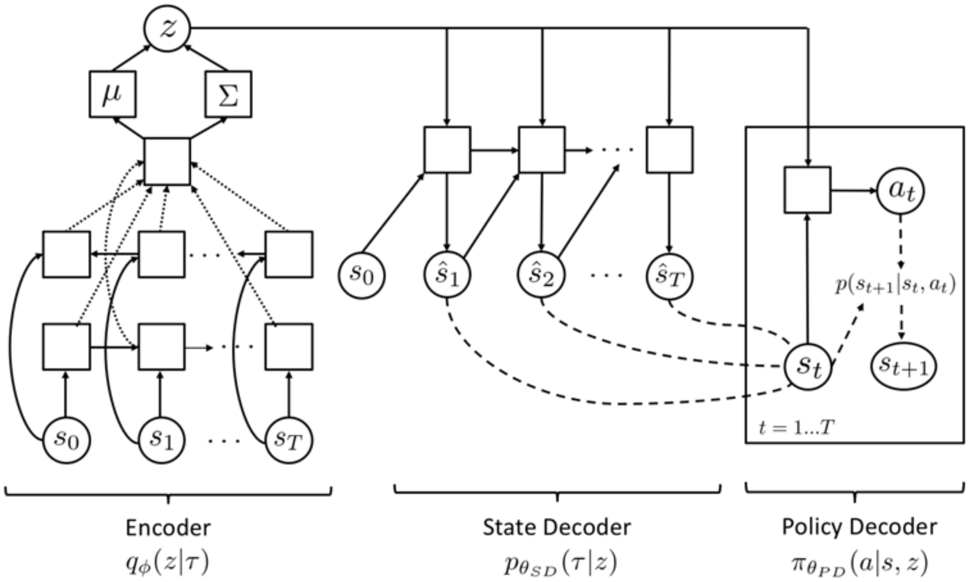


Figure 4.2: The SeCTAR model computation graph. A trajectory is encoded into a latent distribution, from which we sample a latent z. We then (1) directly decode z into a sequence of states using a recurrent state decoder and (2) condition a policy decoder on z to produce the same trajectory through sequential execution in the environment.

Hierarchical Control with SeCTAR

After training the SeCTAR model as described above, we can apply it to perform hierarchical control. Since SeCTAR provides us with a latent representation of trajectories, we can design a meta-controller that reasons sequentially in the space of these latent variables at a coarser time scale than the individual time steps in the environment. Decision making in the latent space serves two purposes. First, it allows for more coherent exploration than randomized action selection. Second, it shortens the effective horizon of the problem to be solved in latent space.

To perform temporally extended planning, we can use a meta-controller that sequentially chooses latent space values z. Each latent z is used to condition the policy decoder $\pi_{\theta_{PD}}(a \mid s, z)$, which is executed in the environment for T steps, after which the meta-controller picks another latent. Although there are several choices for designing or learning such a meta-controller, we consider an approach using model-based planning with model predictive control (MPC), which takes advantage of the state decoder. Model predictive control is an effective control method which performs control by finite horizon model based planning, with iterative replanning at every time step. We refer readers to García et al. (1989) for a comprehensive overview.

An important property of the SeCTAR model is that the differentiable state decoder and the non-differentiable policy decoder are trained to be consistent with each other

Algorithm 1 Model predictive control in latent space

```
Given: trained SeCTAR model, Reward function R
for timestep $t \in \{1, \dots, H/T\}$ do
    Sample $K$ sequences of latents from the prior where each sequence has $H_{MPC}$ number of latents
    Use the state decoder to predict environment states of length $T \times H_{MPC}$ for each latent sequence.
    Evaluate the reward per sequence, and choose the best sequence of latents.
    Execute the policy decoder $\pi_{\theta_{PD}}(a \mid s, z)$ conditioned on the first latent $z$ from the chosen sequence, for $T$ steps starting at $s_0^t$.
end for
```

(Equation 4.2). The state decoder represents a model of how the policy decoder will actually behave in the environment for a particular latent. This is similar to a dynamics model, but built at the trajectory level rather than the the transition level (i.e., operating on (s_t, a_t, s_{t+1}). In this work, we use this interpretation of the state decoder as a model to build a model predictive controller in latent space. Note that the state decoder only needs to make predictions about the outcomes of the corresponding closed-loop policy, which is significantly easier than forward dynamics prediction for arbitrary actions. We use the latent space as the action space for MPC, and perform simple shooting-based planning via random sampling and replanning to generate a sequence of latent variables that maximize a given reward function.

Specifically, given an episode of length H and SeCTAR trained with trajectories of length T, we solve the following planning problem in the latent space over a horizon of H/T (the effective horizon in latent space)

$$\begin{aligned} \max_{z_0, z_1 \dots, z_{H/T}} \quad & \sum_{t=0}^{H/T} \gamma^{tT} R(\tau_t) \\ \text{subject to} \quad & \tau_t \sim p_{\theta_{SD}}(\tau \mid z_t, s_0^t) \\ & s_0^0 = s_0, s_0^{t+1} = s_T^t \end{aligned}$$

Here, τ_t is a trajectory sampled from the state decoder $p_{\theta_{SD}}(\tau \mid z_t, s_0^t)$ conditioned on the current state, and s_0^t represents the start of the trajectory segment, which is the last state in the previous segment. $R(\tau_t) = \sum_{i=0}^{T-1} \gamma^i R(s_i^t, a_i^t)$ is the discounted sum of rewards of trajectory τ_t. To perform this optimization, we use a simple shooting based method Nagabandi et al. (2017) for model-based planning in latent space, described in Algorithm 1.

Exploration for SeCTAR

The model proposed in Section 4.1, provides an effective way to learn representations for trajectories and generate behavior via a state and policy decoder. However, if we assume that the trajectory data is observed and fixed as we have thus far, the trajectories that our

Algorithm 2 Overall algorithm overview

```
Initialize replay buffer and SeCTAR with data from randomly initialized $\pi_e$
for iteration $j \in \{1, \ldots, J\}$ do
    Execute model predictive control in latent space as in Algorithm 1
    Run the explorer $\pi_e$ for $T \times H_e$ starting from a random sample of states visited by MPC
    Update $\pi_e$ using PPO with reward as negative ELBO (4.4) estimated on each of the $H_e$ trajectories
    Train SeCTAR as described in Section 4.1 using data collected by $\pi_e$ in this iteration, mixed with some data from prior iterations in the replay buffer
end for
```

model can generate are restricted by the distribution of observed data. This is particularly problematic in the setting of RL problems over long horizons, where there is a need to explore the environment significantly. The distribution of trajectories that SeCTAR is trained on cannot simply be fixed but needs to be updated periodically to explore more of the state space.

In order to collect data to train the SeCTAR, we introduce a policy π_e that we refer to as the explorer policy. The goal of the explorer policy is to collect data which is as useful as possible for training the SeCTAR model and performing hierarchical planning with it. The explorer policy should gather data by (1) exploring in regions which are relevant to the hierarchical task being solved, and (2) exploring diverse behavior within these regions.

We explore in the neighborhood of task relevant states by initializing the explorer policy near the distribution of states visited by the MPC controller described in Section 4.1. We can achieve this by running the hierarchical controller with a randomly truncated horizon, and letting the explorer policy take over execution. For environments that allow resets to a given state, we can also start the explorer policy directly from a random sample of states visited by the MPC controller.

For π_e to explore diverse behavior, we propose maximizing the entropy of the marginal trajectory distribution $p(\tau)$ induced under π_e. Previous work on maximum entropy RL Haarnoja et al. (2017); Mnih et al. (2016); Schulman et al. (2017a) typically maximize the conditional entropy $\mathcal{H}(\pi(a \mid s))$ of the policy distribution $\pi(a \mid s))$. In this work we suggest maximizing the marginal entropy over distributions of entire trajectories, which is different from maximizing entropy over the policy distribution. The objective can be written as:

$$\max_{\theta} \mathcal{H}(p_\theta(\tau)) = -\mathbb{E}_{p_\theta(\tau)}[\log p_\theta(\tau)] \tag{4.3}$$

Optimizing this objective reduces (on applying product rule, and removing a constant baseline) to policy gradient, with $-\log p_\theta(\tau)$ as the reward function per trajectory. The log likelihood $\log p_\theta(\tau)$ is typically intractable to estimate. However, SeCTAR provides us an effective way to estimate $\log p_\theta(\tau)$ by using a lower bound. SeCTAR optimizes the evidence lower bound (ELBO) to maximize likelihood of trajectories, which suggests a simple approximation for

$\log p_\theta(\tau)$ via the negated ELBO

$$-\mathbb{E}_q[\log p_{\theta_{SD}}(\tau \mid z)] + D_{KL}(q(z \mid \tau) \parallel p(z)), \tag{4.4}$$

as an approximation of $-\log(p_\theta(\tau))$. We can then perform policy gradient for exploration with this reward function.

We combine the previously discussed model-predictive control and entropy maximization methods into an iterative procedure which interleaves exploration with model fitting and hierarchical planning, as summarized in Algorithm 2.

4.2 Related Work

Hierarchical reinforcement learning is a well studied area in reinforcement learning Sutton et al. (1999); Dayan and Hinton (1992); Schmidhuber (2008); Parr and Russell (1997); Dietterich (1998). One method is the options framework which involves learning temporally extended subpolicies. However, the number of options is usually both finite and fixed beforehand which may not be optimal for more complex domains such as continuous control tasks. Another challenge is acquiring skills autonomously which previous work bypasses by hand engineering subgoals Sutton et al. (1999) or using pseudo-rewards Dietterich (1998). Some end-to-end gradient-based methods to learn options have recently been proposed as well Bacon et al. (2017); Fox et al. (2017). Our work on the other hand, learns a continuous set of skills without supervision by learning representations over trajectories, and optimizing the entropy over trajectory distributions to encourage a diverse and useful set of primitives.

In most environments, good exploration is a prerequisite for hierarchy. A number of prior works have been proposed to guide exploration based on criteria such as intrinsic motivation Schmidhuber (2008); Stadie et al. (2015), state-visitation counts Strehl and Littman (2008); Bellemare et al. (2016), and optimism in the face of uncertainty Brafman and Tennenholtz (2003). In this work, we suggest a simple unsupervised exploration method which aims to maximize entropy of the marginal of trajectory distributions. This can be thought of as a means of density based exploration, related to Bellemare et al. (2016); Fu et al. (2017) but operating at a trajectory level.

Several recent and concurrent works have proposed methods which are related to ours but have clear distinctions. Florensa et al. (2017); Heess et al. (2016); Hausman et al. (2018) learn stochastic neural networks to modulate low level behavior which is trained on a "proxy" reward function. However, our method does not assume that such a proxy reward function is provided, as it is often restrictive and difficult to obtain in practice. Mishra et al. (2017) uses trajectory segment models for planning but has no mechanism for exploration and does not consider hierarchical tasks. Other works present information-theoretic representation learning frameworks that are also based on latent variable models and variational inference, but have significant differences in their methods and assumptions Gregor et al. (2016); Mohamed and Rezende (2015). Gregor et al. (2016) aims to learn a maximally discriminative set of options by maximizing the mutual information between the final state reached by each of the options

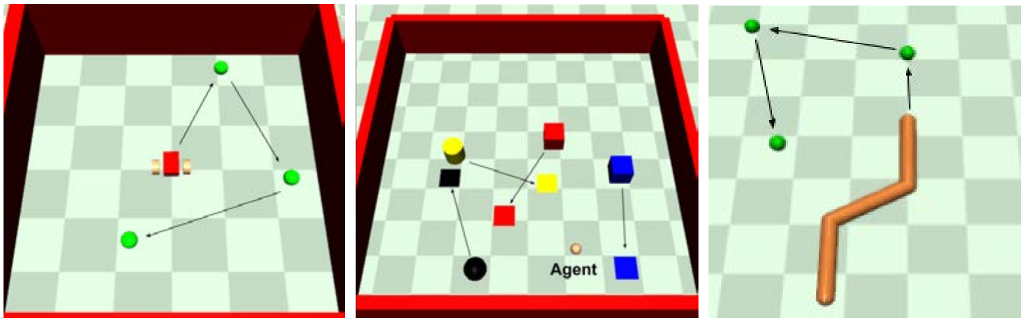


Figure 4.3: From left to right (1) the wheeled locomotion environment with the waypoints depicted in green (2) the object manipulation environment with different objects (blocks and cylinders) and their correspondingly colored goals (squares) (3) the swimmer navigation task with the first 3 waypoints depicted in green.

and the latent representation. Whereas this prior method is applied only on relatively simple gridworlds with discrete options, we learn a continuous space of primitives, together with a state decoder that can be used for model-based higher-level control.

4.3 Experiments

In our experimental evaluation, we aim to address the following questions: (1) Can we learn good exploratory behavior in the absence of task reward, using SeCTAR with our proposed exploration method? (2) Can we use the learned latent space with planning and exploration in the loop to solve hierarchical and sparse reward tasks? (3) Does the state decoder model make meaningful predictions about the outcomes of the high-level actions? We evaluate our method on four different domains: 2D navigation, object manipulation, wheeled locomotion, swimmer navigation which are shown in Figure 4.3. Details of the experimental evaluation can be found in the appendix.

Tasks

2-D Navigation In the 2-D navigation task, the agent can move a fixed distance in each of the four cardinal directions. States are continuous and are observed as the 2D location of the agent. The objective is to navigate a specific sequence of M goal waypoints which lie within a bounding box. The agent is given a reward of 1 for successfully visiting every third goal in the sequence. This evaluates our model's ability to reason over long-horizons with sparse rewards.

Wheeled Locomotion The wheeled environment consists of a two-wheeled cart that is controlled by the angular velocity of its wheels. The cart uses a differential drive system to turn and move in the plane. States include the position, velocity, rotation, and angular velocity of the cart. In this task, the cart must move to a series of goals within a bounding box and receives a reward of 1 after reaching every third goal in the sequence. This experiment tests our method's effectiveness in reasoning over a continuous action space with more complicated physics.

Object Manipulation The object manipulation environment consists of four blocks that the agent can move. The agent, which moves in 2D, can pick up nearby blocks, drop blocks, and navigate in the four cardinal directions, carrying any block it has picked up. The agent must move each block to its corresponding goal in the correct sequence and is given a reward of 1 for each correctly placed block. We designed this task to evaluate our method's ability to explore and learn useful interaction skills with objects in the environment. The sparse, sequential and discontinuous nature of this task makes it challenging.

Swimmer Navigation This task involves navigating through a number of waypoints in the correct order using a 3-link robotic swimmer. The agent is given a reward of 1 for successfully visiting every third goal. This task requires acquiring both a low-level swimming gait and a higher-level navigation strategy to visit the waypoints, and presents a more substantial exploration challenge.

Unsupervised Exploration with SeCTAR

To evaluate the effectiveness of the exploration method described in Section 4.1, we consider an unsupervised setting where we interact with environments in the absence of a task reward. We evaluate a simplified version of Algorithm 2 which alternates between (1) exploration with the explorer policy π_e, (2) model fitting with SeCTAR, (3) updating π_e via the ELBO as described in Section 4.1. This is a version of Algorithm 2, with no MPC and π_e initialized at a fixed initial state.

Our goal is to determine if alternating between exploration and SeCTAR model fitting 4.1 provides us with effective exploration behavior, which is a prerequisite for hierarchical reinforcement learning. To evaluate this, we compare the distribution of final states visited by a randomly initialized policy and the explorer policy after unsupervised training. We found that the distribution of states of the explorer policy π_e covered a significantly larger portion of the state space, indicating good exploratory behavior as seen in Figure 4.4. For the object manipulation task, the manipulator learns to pick up objects and move them around maximally while in the locomotion and 2D navigation environments, the agent learns to explore different portions of its state space.

Hierarchical Control

For the next experiment, we compare our full Algorithm 2 against several baselines methods for exploration, hierarchy, and model-based control. To provide a fair comparison, we initialize all methods from scratch, assuming no prior training in the environment. For each environment, we randomly generated 5 sets of goal configurations and compare the average reward over all goal configurations.

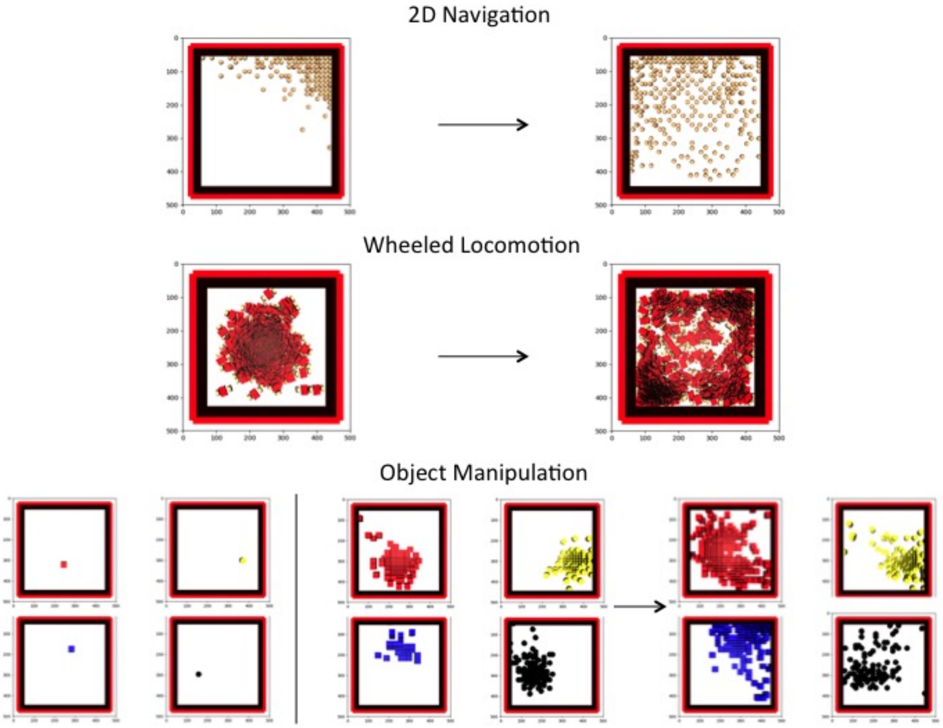


Figure 4.4: We show how our method improves exploration on three environments. On the left, we show the final agent locations for 2D navigation and wheeled location and show final block positions of 4 blocks for object manipulation from a randomly initialized policy. On the right we show the corresponding final locations from our explorer policy trained with the unsupervised exploration objective in Section 4.1. The bottom left plot shows the initial block positions. In all environments we see the agent learns to explore a more evenly distributed region of the state space.

We compare against model-free RL methods, TRPO (Schulman et al., 2015) and A3C (Mnih et al., 2016), an exploration method based on intrinsic motivation - VIME Houthooft et al. (2016), a model-based method from Nagabandi et al. (2017), and two hierarchical methods, FeUdal Networks Vezhnevets et al. (2017) and option-critic Bacon et al. (2017). For the model-based baseline, we perform the same number of random rollouts as our method, with the same planning horizon. However, due to the computational demand of planning at every time-step, we replan at the same rate as our method. We augment the state of the environment with a one-hot encoding of the goal index to enable memoryless policies to operate effectively. We did not evaluate FeUdal and A3C on the wheeled locomotion and the swimmer navigation task, as our implementations of these methods only accommodated discrete actions.

We found that our method can significantly outperform prior methods in terms of task performance and sample complexity as shown in Figure 4.5. These tasks require sequential long horizon reasoning and handling of delayed and sparse rewards. The block manipulation task is particularly challenging for all methods, since it requires the exploration process to pick up blocks and move them around, and only receives a reward when the blocks are placed in the correct locations sequentially. We found that our method is able to significantly outperform the model-based baseline, indicating the usefulness of building trajectory-level models, rather than predictive models at the state-action level. This is likely because model-based predictions at the trajectory level are less susceptible to compounding errors, and are only required to solve the simpler task of predicting the outcomes of specific closed-loop skills, rather than arbitrary actions. We also found that our method performed better than TRPO, A3C, VIME, option-critic, and FeUdal Networks on all tasks. The ability of SeCTAR to learn better on tasks which require challenging exploration and long-horizon reasoning can likely be attributed to being able to perform long-horizon planning using good trajectory representations. The model-based planner at the high level reduces sample complexity significantly, while temporally extended trajectory representations allow us to reason more effectively over longer horizons. While we find that, in the wheeled robot environment, using VIME eventually matches the performance of our method, we are significantly more sample efficient with model-based high-level planning. On the harder object manipulation and swimmer tasks, only our method achieves good performance.

Model Analysis

We visualize interpolations in latent space to see how well the model generalizes to unseen trajectories in Figure 4.6. We choose a latent in the dataset and interpolate to a random point in the latent space. For each interpolated latent we visualize the predicted trajectory from the state decoder and the rolled out trajectory from the policy decoder by plotting the position of the agent. The trajectories are mostly consistent with each other, which demonstrates the potential of SeCTAR to generalize its consistency to new behavior and provide a structured and interpretable latent space.

4.4 Discussion

We proposed a method for hierarchical reinforcement learning that combines representation learning of trajectories with model-based planning in a continuous latent space of behaviors. We describe how to train such a model and use it for long horizon planning, as well as for exploration. Experimental evaluations show that our method outperforms several prior methods and flat reinforcement learning methods in tasks that require reasoning over long horizons, handling sparse rewards, and performing multi-step compound skills.

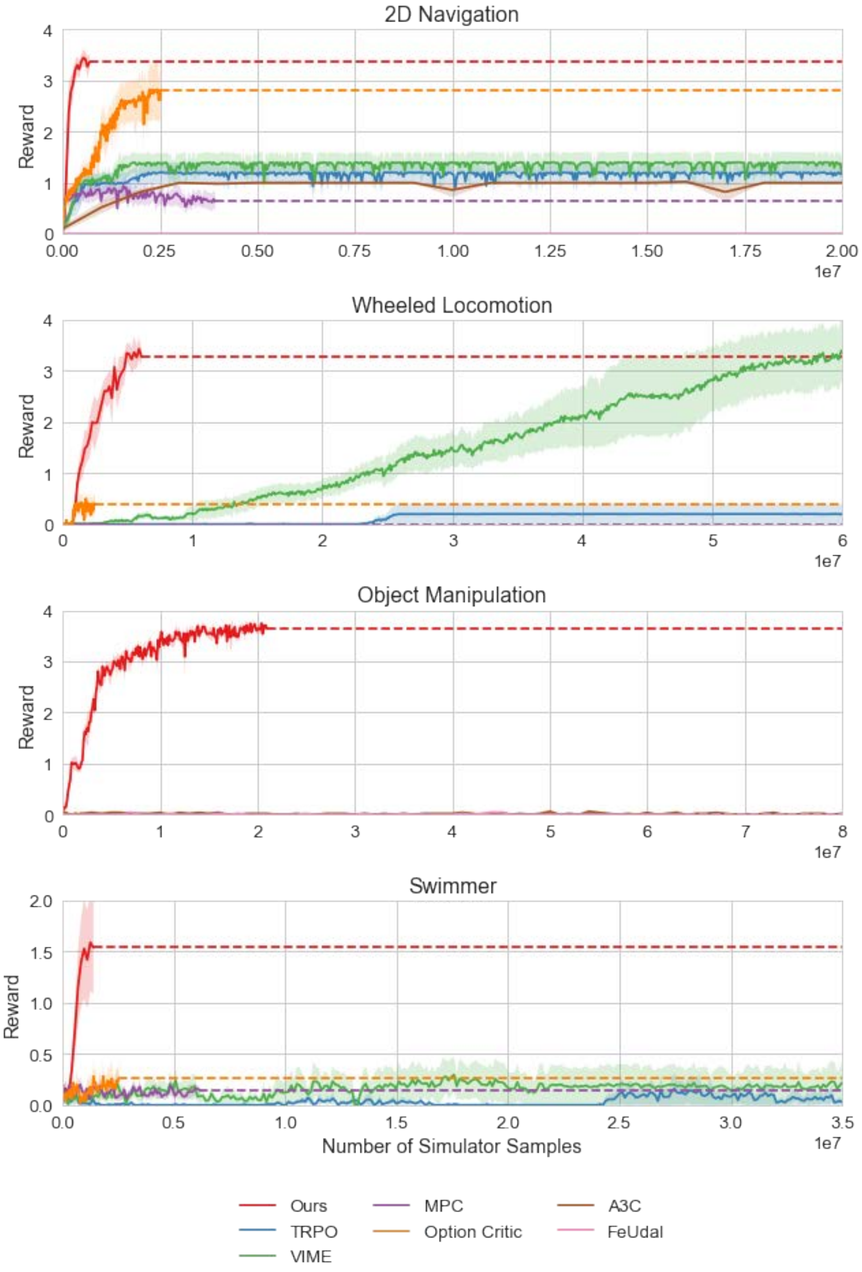


Figure 4.5: Comparison of our method with prior methods on the four tasks. Dashed lines indicate truncated execution. We find that on all tasks, our method is able to achieve higher reward much quicker than model-based, model-free and hierarchical baselines. For object manipulation and swimmer, prior methods fail to do anything meaningful.

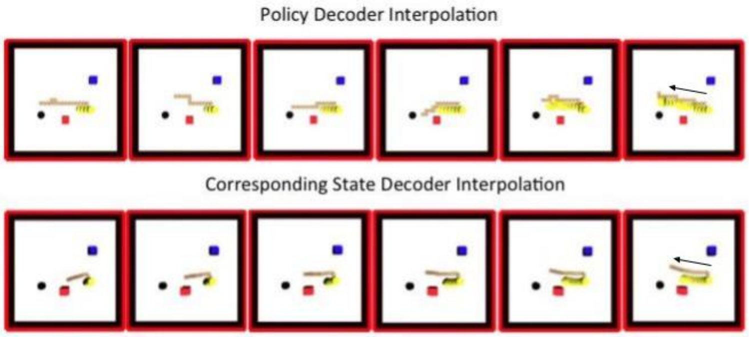


Figure 4.6: Interpolation between two latent codes on the object manipulation environment. We interpolate between two latent codes and visualize the corresponding trajectories from the policy decoder and the state decoder where each plot is a single trajectory. The agent position is in brown and the object positions are in blue, yellow, black and red. From left to right, there is a smooth interpolation between moving the yellow object a little to the left and moving it much further left.

Chapter 5

Evolutionary Based Meta-Learning to Learn General RL Algorithms

Designing new deep reinforcement learning algorithms that can efficiently solve across a wide variety of problems generally requires a tremendous amount of manual effort. Learning to design reinforcement learning algorithms or even small sub-components of algorithms would help ease this burden and could result in better algorithms than researchers could design manually. Our work might then shift from designing these algorithms manually into designing the language and optimization methods for developing these algorithms automatically.

Reinforcement learning algorithms can be viewed as a procedure that maps an agent's experience to a policy that obtains high cumulative reward over the course of training. We formulate the problem of training an agent as one of meta-learning: an outer loop searches over the space of computational graphs or programs that compute the objective function for the agent to minimize and an inner loop performs the updates using the learned loss function. The objective of the outer loop is to maximize the training return of the inner loop algorithm.

Our learned loss function should generalize across many different environments, instead of being specific to a particular domain. Thus, we design a search language based on genetic programming (Koza, 1993) that can express general symbolic loss functions which can be applied to any environment. Data typing and a generic interface to variables in the MDP allow the learned program to be domain agnostic. This language also supports the use of neural network modules as subcomponents of the program, so that more complex neural network architectures can be realized. Efficiently searching over the space of useful programs is generally difficult. For the outer loop optimization, we use regularized evolution (Real et al., 2019), a recent variant of classic evolutionary algorithms that employ tournament selection (Goldberg and Deb, 1991). This approach can scale with the number of compute nodes and has been shown to work for designing algorithms for supervised learning (Real et al., 2020). We adapt this method to automatically design algorithms for reinforcement learning.

While learning from scratch is generally less biased, encoding existing human knowledge into the learning process can speed up the optimization and also make the learned algorithm

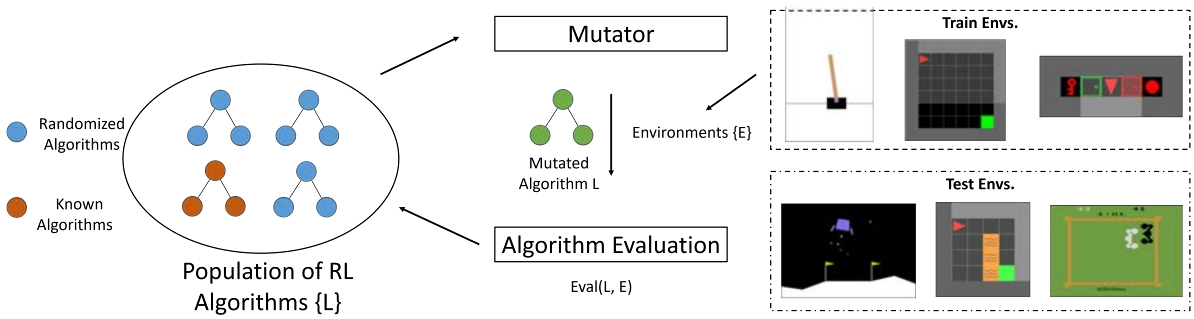


Figure 5.1: Method overview. We use regularized evolution to evolve a population of RL algorithms. A mutator alters top performing algorithms to produce a new algorithm. The performance of the algorithm is evaluated over a set of training environments and the population is updated. Our method can incorporating existing knowledge by starting the population from known RL algorithms instead of purely from scratch.

more interpretable. Because our search language expresses algorithms as a generalized computation graph, we can embed known RL algorithms in the graphs of the starting population of programs. We compare starting from scratch with bootstrapping off existing algorithms and find that while starting from scratch can learn existing algorithms, starting from existing knowledge leads to new RL algorithms which can outperform the initial programs.

We learn two new RL algorithms which outperform existing algorithms in both sample efficiency and final performance on the training and test environments. The learned algorithms are domain agnostic and generalize to new environments. Importantly, the training environments consist of a suite of discrete action classical control tasks and gridworld style environments while the test environments include Atari games and are unlike anything seen during training.

The contribution of this work is a method for searching over the space of RL algorithms, which we instantiate by developing a formal language that describes a broad class of value-based model-free reinforcement learning methods. Our search language enables us to embed existing algorithms into the starting graphs which leads to faster learning and interpretable algorithms. We highlight two learned algorithms which generalize to completely new environments. Our analysis of the meta-learned programs shows that our method automatically discovers algorithms that share structure to recently proposed RL innovations, and empirically attain better performance than deep Q-learning methods.

5.1 Related Work

Learning to learn is an established idea in in supervised learning, including meta-learning with genetic programming (Schmidhuber, 1987; Holland, 1975; Koza, 1993), learning a neural network update rule (Bengio et al., 1991), and self modifying RNNs (Schmidhuber, 1993).

Genetic programming has been used to find new loss functions (Bengio et al., 1994; Trujillo and Olague, 2006). More recently, AutoML (Hutter et al., 2018) aims to automate the machine learning training process. Automated neural network architecture search (Stanley and Miikkulainen, 2002; Real et al., 2017, 2019; Liu et al., 2017; Zoph and Le, 2016; Elsken et al., 2018; Pham et al., 2018) has made large improvements in image classification. Instead of learning the architecture, AutoML-Zero (Real et al., 2020) learns the algorithm from scratch using basic mathematical operations. Our work shares similar ideas, but is applied to the RL setting and assumes additional primitives such as neural network modules. In contrast to AutoML-Zero, we learn computational graphs with the goal of automating RL algorithm design. Our learned RL algorithms generalize to new problems, not seen in training.

Automating RL. While RL is used for AutoML (Zoph and Le, 2016; Zoph et al., 2018; Cai et al., 2018; Bello et al., 2017), automating RL itself has been somewhat limited. RL requires different design choices compared to supervised learning, including the formulation of reward and policy update rules. All of which affect learning and performance, and are usually chosen through trial and error. AutoRL addresses the gap by applying the AutoML framework from supervised learning to the MDP setting in RL. For example, evolutionary algorithms are used to mutate the value or actor network weights (Whiteson and Stone, 2006; Khadka and Tumer, 2018), learn task reward (Faust et al., 2019), tune hyperparameters (Tang and Choromanski, 2020; Franke et al., 2020), or search for a neural network architecture (Song et al., 2020; Franke et al., 2020). This work focuses on task-agnostic RL update rules in the value-based RL setting which are both interpretable and generalizable.

Meta-learning in RL. Recent work has focused on few-shot task adaptation. Finn et al. (2017); Finn and Levine (2018) meta-learns initial parameters which can quickly adapt to new tasks, while RL^2 (Duan et al., 2016) and concurrent work (Wang et al., 2017), formulates RL itself as a learning problem that is learned with an RNN. The meta-learned component of these works is tuned to a particular domain or environment, in the form of NN weights which cannot be used for completely new domains with potentially different sized inputs. Neural Programmer-Interpreters (Reed and De Freitas, 2015; Pierrot et al., 2019) overcome the environment generalization challenge by learning hierarchical neural programs with domain-specific encoders for different environments. Here, the computational graph has a flexible architecture and generalizes across different environments.

Learning RL algorithms or their components, such as a reward bonus or value update function, has been studied previously with meta-gradients (Kirsch et al., 2020; Chebotar et al., 2019; Oh et al., 2020), evolutionary strategies (Houthooft et al., 2018), and RNNs (Duan et al., 2016). Although our work also learns RL algorithms, the update rule is represented as a computation graph which includes both neural network modules and symbolic operators. One key benefit is that the resulting graph can be interpreted analytically and can optionally be initialized from known existing algorithms. Prior work that focuses on learning RL losses, generalizes to different goals and initial conditions *within* a single environment (Houthooft et al., 2018), or learns a domain invariant policy update rule that can generalize to new environments (Kirsch et al., 2020). Another approach searches over the space of curiosity programs using a similar language of DAGs with neural network modules (Alet et al., 2020a) and performs the meta-training on a single environment. In contrast, our method is applied to learn general RL update rules and meta-trained over a diverse set of environments.

5.2 Learning Reinforcement Learning Algorithms

In this section, we first describe the problem setup. An inner loop method $\text{Eval}(L, \mathcal{E})$ evaluates a learned RL algorithm L on a given environment $\mathcal{E}$. Given access to this procedure, the goal for the outer loop optimization is to learn a RL algorithm with high training return over a set of training environments. We then describe the search language which enables the learning of general loss functions and the outer loop method which can efficiently search over this space.

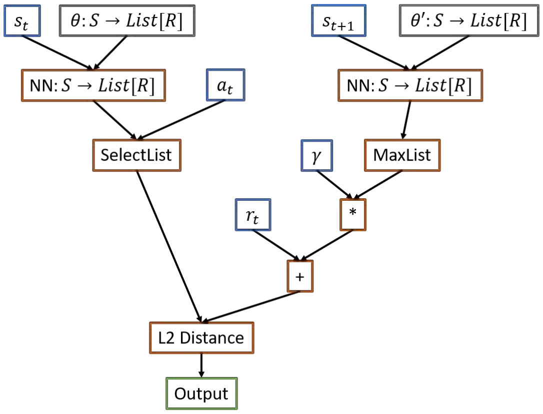


Figure 5.2: Visualization of a RL algorithm, DQN, as a computational graph which computes the loss $L = (Q(s_t, a_t) - (r_t + \gamma * \max_a Q_{targ}(s_{t+1}, a)))^2$. Input nodes are in blue, parameter nodes in gray, operation nodes in orange, and output in green.

Problem Setup

We assume that the agent parameterized with policy $\pi_\theta(a_t|s_t)$ outputs actions a_t at each time step to an environment $\mathcal{E}$ and receives reward r_t and next state s_{t+1}. Since we are focusing on discrete action value-based RL methods, θ will be the parameters for a Q-value function and the policy is obtained from the Q-value function using an ϵ-greedy strategy. The agent saves this stream of transitions (s_t, s_{t+1}, a_t, r_t) to a replay buffer and continually updates the policy by minimizing a loss function $L(s_t, a_t, r_t, s_{t+1}, \theta, \gamma)$ over these transitions with gradient descent. Training will occur for a fixed number of M training episodes where in each episode m, the agent earns episode return $R_m = \sum_{t=0}^{T} r_t$. The performance of an algorithm for a given environment is summarized by the normalized average training return, $\frac{1}{M}\sum_{m=1}^{M} \frac{R_i - R_{min}}{R_{max} - R_{min}}$, where R_{min} and R_{max} are the minimum and maximum return for that environment. We assume these are known ahead of time. This inner loop evaluation procedure $\text{Eval}(L, \mathcal{E})$ is outlined in Algorithm 3. To score an algorithm, we use the normalized average training return instead of the final behavior policy return because the former metric will factor in sample efficiency as well.

The goal of the meta-learner is to find the optimal loss function $L(s_t, a_t, r_t, s_{t+1}, \theta, \gamma)$ to optimize π_θ with maximal normalized average training return over the set of training environments. The full objective for the meta-learner is:

$$L^* = \arg\max_L \left[\sum_{\mathcal{E}} \text{Eval}(L, \mathcal{E}) \right]$$

L is represented as a computational graph which we describe in the next section.

Search Language

Our search language for the algorithm L should be expressive enough to represent existing algorithms while enabling the learning of new algorithms which can obtain good generalization performance across a wide range of environments. Similar to Alet et al. (2020a), we describe the RL algorithm as general programs with a domain specific language, but we target updates to the policy rather than reward bonuses for exploration. Algorithms will map transitions (s_t, a_t, s_{t+1}, r_t), policy parameters θ, and discount factor γ into a scalar loss to be optimized with gradient descent. We express L as a computational graph or directed acyclic graph (DAG) of nodes with typed inputs and outputs. See Figure 5.2 for a visualization of DQN expressed in this form. Nodes are of several types:

Input nodes represent inputs to the program, and include elements from transitions (s_t, a_t, s_{t+1}, r_t) and constants, such as the discount factor γ.

Parameter nodes are neural network weights, which can map between various data types. For example, the weights for the Q-value network will map an input node with state data type to a list of real numbers for each action.

Operation nodes compute outputs given inputs from parent nodes. This includes applying parameter nodes, as well as basic math operators from linear algebra, probability, and statistics. A full list of operation nodes is provided in Appendix C.1. By default, we set the last node in the graph to compute the output of the program which is the scalar loss function to be optimized. Importantly, the inputs and outputs of nodes are typed among (state, action, vector, float, list, probability). This typing allows for programs to be applied to any domain. It also restricts the space of programs to ones with valid typing which reduces the search space.

Algorithm 3 Algorithm Evaluation, Eval$(L, \mathcal{E})$

```
**Input:** RL Algorithm $L$, Environment $\mathcal{E}$, training episodes $M$
**Initialize:** Q-value parameters $\theta$, target parameters $\theta'$ empty replay buffer $\mathcal{D}$
**for** $i = 1$ **to** $M$ **do**
  **for** $t = 0$ **to** $T$ **do**
    With probability $\epsilon$, select a random action $a_t$,
    otherwise select $a_t = \arg\max_a Q(s_t, a)$
    Step environment $s_{t+1}, r_t \sim \mathcal{E}(a_t, s_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{s_t, a_t, r_t, s_{t+1}\}$
    Update parameters $\theta \leftarrow \theta - \nabla_\theta L(s_t, a_t, r_t, s_{t+1}, \theta, \gamma)$
    Update target $\theta' \leftarrow \theta$
  **end for**
  Compute episode return $R_m = \sum_{t=0}^{T} r_t$
**end for**
**Output:**
  Normalized training performance $\frac{1}{M}\sum_{m=1}^{M} \frac{R_m - R_{min}}{R_{max} - R_{min}}$
```

Algorithm 4 Evolving RL Algorithms

```
**Input:** Training environments $\{\mathcal{E}\}$, hurdle environment $\mathcal{E}_h$, hurdle threshold $\alpha$, optional existing algorithm $A$
**Initialize:** Population $P$ of RL algorithms $\{L\}$, history $H$, randomized inputs $I$. If bootstrapping, initialize $P$ with $A$.
Score each L in $P$ with $H[L].score \leftarrow \sum_{\mathcal{E}} \text{Eval}(L, \mathcal{E})$
**for** $c = 0$ **to** C **do**
  Sample tournament $T \sim Uniform(P)$
  Parent algorithm $L \leftarrow$ highest score algorithm in $T$
  Child algorithm $L' \leftarrow$ Mutate(L)
  $H[L'].hash \leftarrow \text{Hash}(L'(I))$
  **if** $H[L'].hash$ was new **and** $\text{Eval}(L', \mathcal{E}_h) > \alpha$ **then**
    $H[L'].score \leftarrow \sum_{\mathcal{E}} \text{Eval}(L', \mathcal{E})$
  **end if**
  Add $L'$ to population $P$
  Remove oldest $L$ from population
**end for**
**Output:** Algorithm L with highest score
```

Evolutionary Search Method

Evaluating thousands of programs over a range of complex environments is prohibitively expensive, especially if done serially. We adapt a genetic programming (Koza, 1993) method for the search method and use regularized evolution (Real et al., 2019), a variant of classic evolutionary algorithms that employ tournament selection (Goldberg and Deb, 1991). Regularized evolution has been shown to work for learning supervised learning algorithms (Real et al., 2020) and can be parallelized across compute nodes. Tournament selection keeps a population of P algorithms and improves the population through cycles. Each cycle picks a tournament of $T < P$ algorithms at random and selects the best algorithm in the tournament as a parent. The parent is mutated into a child algorithm which gets added to the population while the oldest algorithm in the population is removed. We use a single type of mutation which first chooses which node in the graph to mutate and then replaces it with a random operation node with inputs drawn uniformly from all possible inputs.

There exists a combinatorially large number of graph configurations. Furthermore, evaluating a single graph, which means training the full inner loop RL algorithm, can take up a large amount of time compared to the supervised learning setting. Speeding up the search and avoiding needless computation are needed to make the problem more tractable. We extend regularized evolution with several techniques, detailed below, to make the optimization more efficient. The full training procedure is outlined in Algorithm 4.

Functional equivalence check (Real et al., 2020; Alet et al., 2020b). Before evaluating a program, we check if it is functionally equivalent to any previously evaluated program. This check is done by hashing the concatenated output of the program for 10 values of randomized inputs. If a mutated program is functionally equivalent to an older program, we still add it to the population, but use the saved score of the older program. Since some nodes of the graph do not always contribute to the output, parts of the mutated program may eventually contribute to a functionally different program.

Early hurdles (So et al., 2019). We want poor performing programs to terminate early so that we can avoid unneeded computation. We use the CartPole environment as an early hurdle environment $\mathcal{E}_h$ by training a program for a fixed number of episodes. If an algorithm performs poorly, then episodes will terminate in a short number of steps (as the pole falls rapidly) which quickly exhausts the number of training episodes. We use $\text{Eval}(L, \mathcal{E}_h) < \alpha$ as the threshold for poor performance with α chosen empirically.

Program checks. We perform basic checks to rule out and skip training invalid programs. The loss function needs to be a scalar value so we check if the program output type is a float ($\text{type}(L) = \mathbb{R}$). Additionally, we check if each program is differentiable with respect to the policy parameters by checking if a path exists in the graph between the output and the policy parameter node.

Learning from Scratch and Bootstrapping. Our method enables both learning from scratch and learning from existing knowledge by bootstrapping the initial algorithm population with existing algorithms. We learn algorithms from scratch by initializing the population of algorithms randomly. An algorithm is sampled by sampling each operation node sequentially in the DAG. For each node, an operation and valid inputs to that operation are sampled uniformly over all possible options.

While learning from scratch might uncover completely new algorithms that differ substantially from the existing methods, this method can take longer to converge to a reasonable algorithm. We would like to incorporate the knowledge we do have of good algorithms to bootstrap our search from a better starting point. We initialize our graph with the loss function of DQN (Mnih et al., 2013) so that the first 7 nodes represent the standard DQN loss, while the remaining nodes are initialized randomly. During regularized evolution, the nodes are not frozen, such that it is possible for the existing sub-graph to be completely replaced if a better solution is found.

5.3 Learned RL Algorithm Results and Analysis

We discuss the training setup and results of our experiments. We highlight two learned algorithms with good generalization performance, DQNClipped and DQNReg, and analyze their structure. Additional details are available at `https://sites.google.com/view/evolvingrl`.

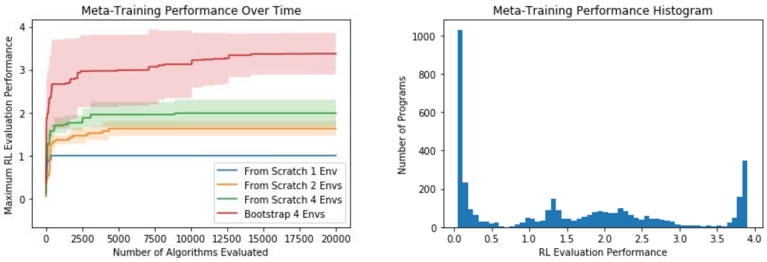
Training Setup

Meta-Training details: We search over programs with maximum 20 nodes, not including inputs or parameter nodes. A full list of node types is provided in Appendix C.1. We use a population size of 300, tournament size of 25, and choose these parameters based on the ones used in (Real et al., 2019). Mutations occur with probability 0.95. Otherwise a new random program is sampled. The search is done over 300 CPUs and run for roughly 72 hours, at which point around $20,000$ programs have been evaluated. The search is distributed such that any free CPU is allocated to a proposed individual such that there are no idle CPUs. Further meta-training details are in Appendix C.2.

Training environments: The choice of training environments greatly affects the learned algorithms and their generalization performance. At the same time, our training environments should be not too computationally expensive to run as we will be evaluating thousands of RL algorithms. We use a range of 4 classical control tasks (CartPole, Acrobat, MountainCar, LunarLander) and a set of 12 multitask gridworld style environments from MiniGrid (Chevalier-Boisvert et al., 2018). These environments are computationally cheap to run but also chosen to cover a diverse set of situations. This includes dense and sparse reward, long time horizon, and tasks requiring solving a sequence of subgoals such as picking up a key and unlocking a door. More details are in Appendix C.3.

The training environments always include CartPole as an initial hurdle. If an algorithm succeeds on CartPole (normalized training performance greater than 0.6), it then proceeds to a harder set of training environments. For our experiments, we choose these training environments by sampling a set of 3 environments and leave the rest as test environments. For learning from scratch we also compare the effect of number of training environments on the learned algorithm by comparing training on just CartPole versus training on CartPole and LunarLander.

RL Training details: For training the RL agent, we use the same hyperparameters across all training and test environments except as noted. All neural networks are MLPs of size (256, 256) with ReLU activations. We use the Adam optimizer with a learning rate of 0.0001. ϵ is decayed linearly from 1 to 0.05 over 1e3 steps for the classical control tasks and over 1e5 steps for the MiniGrid tasks.



(a) Learning curve (b) Performance histogram

Figure 5.3: Left: Meta-training performance over different number of environments from scratch, and bootstrapping. Plotted as RL evaluation performance (sum of normalized training return across the training environments) over the number of candidate algorithms. Shaded region represents one standard deviation over 10 random seeds. More training environments leads to better algorithms. Bootstrapping from DQN speeds up convergence and higher final performance. Right: Meta-training performance histogram for bootstrapped training. Many of the top programs have similar structure (Appendix C.4).

Learning Convergence

Figure 5.3a shows convergence over several training configurations. We find that at the end of training roughly 70% of proposed algorithms are functionally equivalent to a previously evaluated program, while early hurdles cut roughly another 40% of proposed non-duplicate programs.

Varying number of training environments: We compare learning from scratch with a single training environment (CartPole) versus with two training environments (CartPole and LunarLander). While both experiments reach the maximum performance on these environments (Figure 5.3a), the learned algorithms are different. The two-environment training setup learns the known TD loss

$$L_{DQN} = (Q(s_t, a_t) - (r_t + \gamma * \max_a Q_{targ}(s_{t+1}, a)))^2$$

while the single-environment training setup learns a slight variation $L = (Q(s_t, a_t) - (r_t + \max_a Q_{targ}(s_{t+1}, a)))^2$ that does not use the discount, indicating that the range of difficulty on the training environments is important for learning algorithms which can generalize.

Learning from scratch versus bootstrapping: In Figure 5.3a, we compare training from scratch versus training from bootstrapping on four training environments (CartPole, KeyCorridorS3R1, DynamicObstacle-6x6, DoorKey-5x5). The training performance does not saturate, leaving room for improvement. Bootstrapping from DQN significantly improves both the convergence and performance of the meta-training, resulting in a 40% increase in final training performance.

Learned RL Algorithms: DQNClipped and DQNReg

In this section, we discuss two particularly interesting loss functions that were learned by our method, and that have good generalization performance on the test environments. Let

$$Y_t = r_t + \gamma * \max_a Q_{targ}(s_{t+1}, a), \text{ and } \delta = Q(s_t, a_t) - Y_t.$$

The first loss function DQNClipped is

$$L_{\text{DQNClipped}} = \max\left[Q(s_t, a_t), \delta^2 + Y_t\right] + \max\left[Q(s_t, a_t) - Y_t, \gamma(\max_a Q_{targ}(s_{t+1}, a))^2\right].$$

$L_{\text{DQNClipped}}$ was trained from bootstrapping off DQN using three training environments (LunarLander, MiniGrid-Dynamic-Obstacles-5x5, MiniGrid-LavaGapS5). It outperforms DQN and double-DQN, DDQN, (van Hasselt et al., 2015) on both the training and unseen environments (Figure 5.4). The intuition behind this loss function is that, if the Q-values become too large (when $Q(s_t, a_t) > \delta^2 + Y_t$), the loss will act to minimize $Q(s_t, a_t)$ instead of the normal δ^2 loss. Alternatively, we can view this condition as $\delta = Q(s_t, a_t) - Y_t > \delta^2$. This means when δ is small enough then $Q(s_t, a_t)$ are relatively close and the loss is just to minimize $Q(s_t, a_t)$.

The second learned loss function, which we call DQNReg, is given by

$$L_{\text{DQNReg}} = 0.1 * Q(s_t, a_t) + \delta^2.$$

DQNReg was trained from bootstrapping off DQN using three training environments (KeyCorridorS3R1, Dynamic-Obstacles-6x6, DoorKey-5x5). In comparison to DQNClipped, DQNReg directly regularizes the Q values with a weighted term that is always active. We note that both of these loss functions modify the original DQN loss function to regularize the Q-values to be lower in value. While DQNReg is quite simple, it matches or outperforms the baselines on all training and test environments including from classical control and Minigrid. It does particularly well on a few test environments (SimpleCrossingS9N1, DoorKey-6x6, and Unlock) and solves the tasks when other methods fail to attain any reward. It is also much more stable with lower variance between seeds, and more sample efficient on test environments (LavaGapS5, Empty-6x6, Empty-Random-5x5).

In Table 5.1, we evaluate DQNReg on a set of Atari games. We use the same architecture as in DQN (Mnih et al., 2013) and use the same no-op evaluation procedure which evaluates a trained policy every 1 million training steps over 200 test episodes. Even though meta-training was on computationally simple, non-image based environments, we find that DQNReg can generalize to image-based environments and outperform baselines. The results for the baselines are taken from their respective papers (Mnih et al., 2013; van Hasselt et al., 2015; Schulman et al., 2017).

These algorithms are related to recently proposed RL algorithms, conservative Q-learning (CQL) (Kumar et al., 2020) and M-DQN (Vieillard et al., 2020). CQL learns a conservative Q-function by augmenting the standard Bellman error objective with a simple Q-value

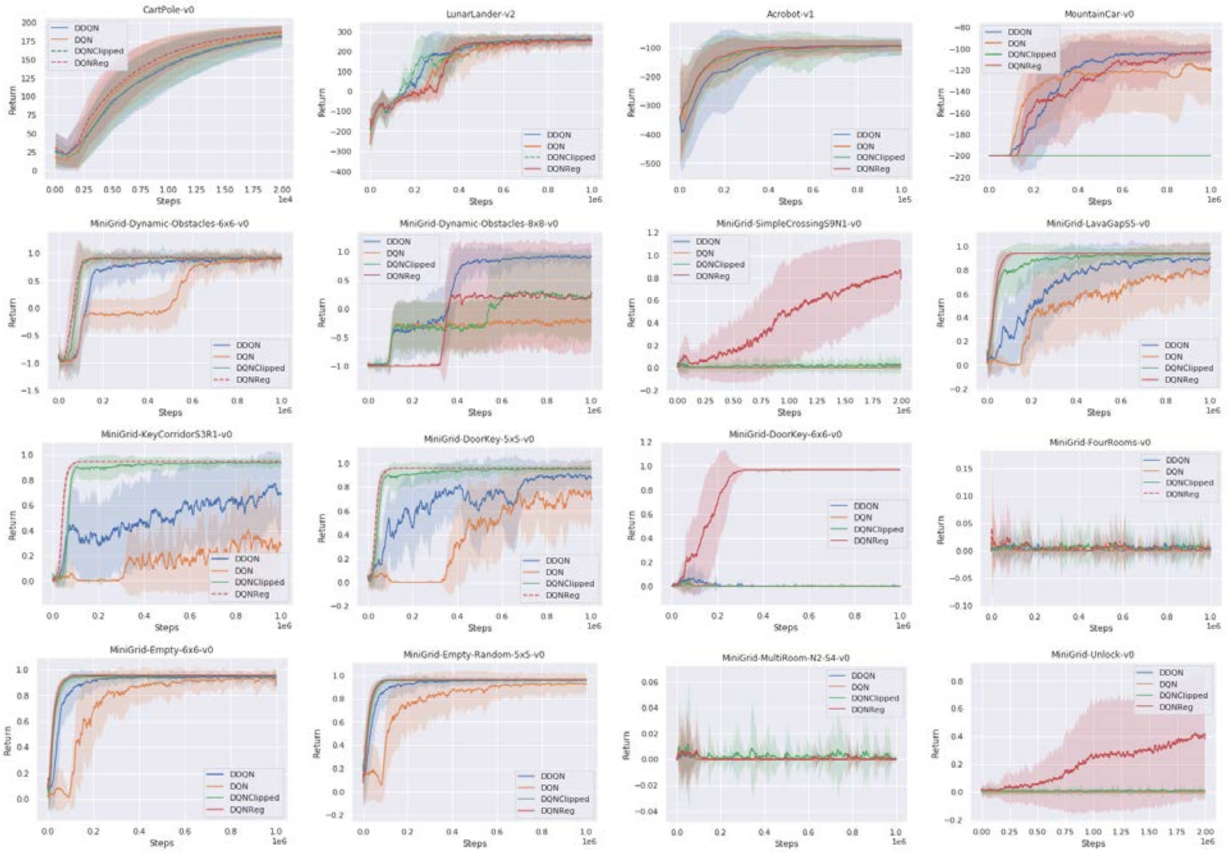


Figure 5.4: Performance of learned algorithms (DQNClipped and DQNReg) versus baselines (DQN and DDQN) on training and test environments as measured by episode return over 10 training seeds. A dashed line indicates that the algorithm was meta-trained on that environment while a solid line indicates a test environment. DQNReg can match or outperform the baselines on almost all the training and test environments. Shaded regions correspond to 1 standard deviation.

regularizer: $\log \sum_a \exp\left(Q(s_t, a)\right) - Q(s_t, a_t)$ which encourages the agent to stay close to the data distribution while maintaining a maximum entropy policy. DQNReg similarly augments the standard objective with a Q-value regularizer although does so in a different direction by preventing overestimation. M-DQN modifies DQN by adding the scaled log-policy (using the softmax Q-values) to the immediate reward. Both of these methods can be seen as ways to regularize a value-based policy. This resemblance indicates that our method can find useful structures automatically that are currently being explored manually, and could be used to propose new areas for researchers to explore.

We discover that the best performing algorithms from the experiment which learned DQNReg are consistent, and in the form $L = \delta^2 + k * Q(s_t, a_t)$. This loss could use further analysis and investigation, possibly environment-specific tuning of the parameter k. See Appendix C.2 for details.

Env	DQN	DDQN	PPO	DQNReg
Asteroid	1364.5	734.7	2097.5	**2390.4**
Bowling	50.4	68.1	40.1	**80.5**
Boxing	88.0	91.6	94.6	**100.0**
RoadRunner	39544.0	44127.0	35466.0	**65516.0**

Table 5.1: Performance of learned algorithm DQNReg against baselines on several Atari games. Baseline numbers taken from reported papers.

Analysis of Learned Algorithms

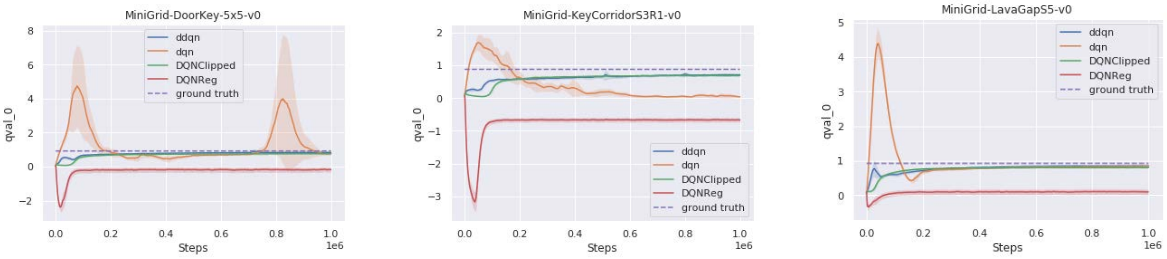


Figure 5.5: Overestimated value estimates is generally problematic in value-based RL. Our method learns algorithms which regularize the Q-values helping with overestimation. We compare the estimated Q-values for our learned algorithms and baselines with the optimal ground truth Q-values across several environments during training. Estimate is for taking action zero from the initial state of the environment. While DQN overestimates the Q-values, our learned algorithms DQNClipped and DQNReg underestimate the Q-values.

We analyze the learned algorithms to understand their beneficial effect on performance. In Figure 5.5, we compare the estimated Q-values for each algorithm. We see that DQN frequently overestimates the Q values while DDQN consistently underestimates the Q values before converging to the ground truth Q value which are computed with a manually designed optimal policy. DQNClipped has similar performance to DDQN, in that it also consistently underestimates the Q values and does so slightly more aggressively than DDQN. DQNReg significantly undershoots the Q values and does not converge to the ground truth. Various works (van Hasselt et al., 2015; Haarnoja et al., 2018; Fujimoto et al., 2018) have shown that overestimated value estimates is problematic and restricting the overestimation improves performance.

The loss function in DQNClipped is composed of the sum of two max operations, and so we can analyze when each update rule is active. We interpret DQNClipped as $\max(v_1, v_2) + \max(v_2, v_3)$ with four cases: 1) $v_1 > v_2$ and $v_3 > v_4$ 2) $v_1 > v_2$ and $v_3 < v_4$ 3) $v_1 < v_2$ and $v_3 < v_4$ 4) $v_1 < v_2$ and $v_3 > v_4$. Case 2 corresponds to minimizing the Q values. Case 3 would correspond to the normal DQN loss of δ^2 since the parameters of Q_{targ} are not updated during gradient descent.

In Figure 5.6, we plot the proportion of when each case is active during training. We see that usually case 3 is generally the most active with a small dip in the beginning but then stays around 95%. Meanwhile, case 2, which regularizes the Q-values, has a small increase in the beginning and then decreases later, matching with our analysis in Figure 5.6, which shows that DQNClipped strongly underestimates the Q-values in the beginning of training. This can be seen as a constrained optimization where the amount of Q-value regularization is tuned accordingly. The regularization is stronger in the beginning of training when overestimation is problematic $(Q(s_t, a_t) > \delta^2 + Y_t)$ and gets weaker as δ^2 gets smaller.

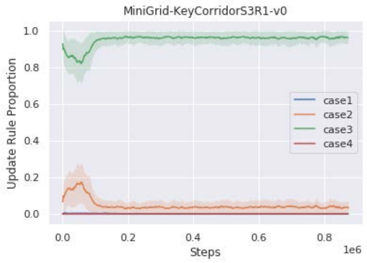


Figure 5.6: Our learned algorithm, DQNClipped, can be broken down into four update rules where each rule is active under certain conditions. Case 3 corresponds to normal TD learning while case 2 corresponds to minimizing the Q-values. Case 2 is more active in the beginning when value overestimation is a problem and then becomes less active as it is no longer needed.

5.4 Discussion

In this chapter, we presented a method for learning reinforcement learning algorithms. We designed a general language for representing algorithms which compute the loss function for value-based model-free RL agents to optimize. We highlight two learned algorithms which although relatively simple, can obtain good generalization performance over a wide range of environments. Our analysis of the learned algorithms sheds insight on their benefit as regularization terms which are similar to recently proposed algorithms. Our work is limited to discrete action and value-based RL algorithms that are close to DQN, but could easily be expanded to express more general RL algorithms such as actor-critic or policy gradient methods. How actions are sampled from the policy could also be part of the search space. The set of environments we use for both training and testing could also be expanded to include a more diverse set of problem types.

Chapter 6

Unsupervised Objectives for General Intelligence

Reinforcement learning offers a framework for learning control policies that maximize a given measure of reward – ideally, rewards that incentivize simple high-level goals, such as survival, accumulating a particular resource, or accomplishing some long-term objective. However, extrinsic rewards may be insufficiently informative to encourage an agent to explore and understand its environment, particularly when the environment is *partially-observed*: when the agent has a limited view of its environment. A generalist agent should instead acquire an understanding of its environment before a specific objective or reward is provided. This goal motivates the study of *self-supervised / unsupervised* reinforcement learning: algorithms that provide the agent with an intrinsically-grounded drive to acquire understanding and control of its environment in the absence of an extrinsic reward signal. Agents trained with intrinsic reward signals might accomplish tasks specified via simple and sparse rewards more quickly, or may acquire broadly useful skills that could be adapted to specific task objectives. Our aim is to design an embodied agent and a general-purpose intrinsic reward signal that leads to the agent controlling partially-observed environments when equipped only with a high-dimensional sensor (camera) and no prior knowledge.

A large body of prior methods for self-supervised reinforcement learning focus on attaining *coverage*, typically through novelty-seeking or skill-discovery objectives; see Hafner et al. (2020) for a survey. As argued in prior work (Friston et al., 2010; Friston, 2013; Fountas et al., 2020; Berseth et al., 2021), a compelling alternative to coverage suited to complex and dynamic environments is to minimize surprise, which incentivizes an agent to control aspects of its environment to achieve homeostasis within it – i.e. constructing and maintaining a niche where it can reliably remain despite external perturbations. We generally expect agents that succeed at minimizing surprise in complex environments to develop similarly complex behaviors; such acquired complex behaviors may be repurposed for other tasks (Berseth et al., 2021). However, these frameworks do not explicitly address the difficulty of controlling partially-observed environments: if an otherwise complex and chaotic environment contains a “dark room” (small reliable niche), an agent could minimize surprise simply by hiding in this

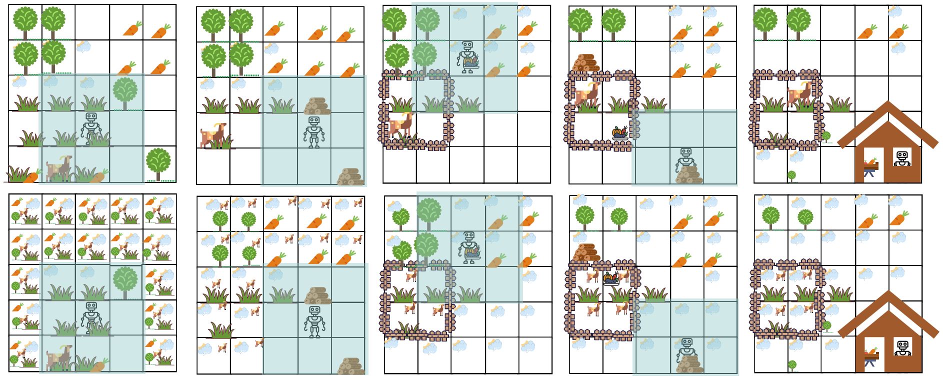


Figure 6.1: *Top row:* The environment consists of a large number of objects, some of which (e.g., the goat) move and act in unpredictable ways, and are not observed unless the agent is nearby. *Bottom row:* If the agent maintains a latent state space model of the world, it has uncertain beliefs about unobserved objects, particularly those that are dynamic (like the goat). If the agent reduces the long-horizon average entropy of its beliefs, it will first seek out information (e.g., finding the goat), and then modify the environment to limit the range of states the goat can occupy *even when it is no longer observed*, for example by building a fence around it.

room and refusing to make meaningful observations, thereby failing to explore and control the wider surrounding environment.

Consider Fig. 6.1, which depicts a partially-observed outdoor environment with various flora (trees, vegetables, and grass), fauna (a goat), weather, and an agent. We will discuss three different intrinsic incentives an agent might adopt in this environment. If the agent's incentive is to (i) minimize the entropy of its next observation, it will seek the regions with minimal unpredictable variations in flora, fauna, and weather. This is unsatisfying because it merely requires avoidance, rather than interaction. Let us assume the agent will maintain a model of its *belief* about a learned *latent state* – the agent cannot observe the *true* state, instead it learns a state representation. Further, let us assume the agent maintains a separate model of the visitation of its latent state – we will refer to this distribution as its *latent visitation*. If the agent's incentive is to (ii) minimize the entropy of belief (either at every step or at some final step), the agent will gather information and take actions to make the environment predictable: find and observe the changes in flora, fauna, and weather that are predictable and avoid those that aren't. However, once it has taken actions to make the world predictable, this agent is agnostic to future change – it will not resist predictable changes in the environment. Finally, if the agent's incentive is to (iii) minimize the entropy of its latent visitation, this will result in categorically different behavior: the agent will seek both to make the world predictable by gathering information about it and *prevent it from changing*. While both the belief and latent visitation entropy minimization objectives are worthwhile intrinsic motivation objectives to study, we speculate that an agent that is adept at preventing its environment from changing will generally learn more complex behaviors.

We present a concise and effective objective for self-supervised reinforcement learning in dynamic partially-observed environments: minimize the entropy of the agent's latent visitation under a latent state-space model learned from exploration. Our method, which we call Believer, results in an agent that learns to seek out and control factors of variation outside of its immediate observations. We instantiate this framework by simultaneously learning a state-space model as a Deep Variational Bayes Filter along with a policy that employs the model's beliefs. Our experiments show that our method learns to represent and control dynamic entities in partially-observed visual environments with *no extrinsic reward signal*, including in several 3D environments.

6.1 Maxwell's Demon and Belief Entropy

The main concept behind our approach to self-supervised reinforcement learning is that incentivizing an agent to minimize the entropy of its *beliefs* about the world is sufficient to lead it to *both* gather information about the world *and* learn to control aspects of its world. Our approach is partly inspired by a well-known connection between information theory and thermodynamics, which can be illustrated informally by a version of the Maxwell's Demon thought experiment (Maxwell and Pesic, 2001; Leff and Rex, 2014). Imagine a container separated into compartments, as shown in Fig. 6.2. Both compartments contain gas molecules that bounce off of the walls and each other in a somewhat unpredictable fashion, though short-term motion of these molecules (between collisions) is predictable. The compartments are separated by a massless door, and the agent (the eponymous "demon") can open or close the door at will to sort the particles.[1] By sorting the particles onto one side, the demon appears to reduce the disorder of the system, as measured by the thermodynamic entropy, S, which increases the energy, F available to do work, as per Helmholtz's free energy relationship, $F = U - TS$. The ability to do work affords the agent *control over the environment*. This apparent violation of the second law of thermodynamics is resolved by accounting for the agent's information processing needed to make decisions (Bennett, 1982). By concentrating the particles into a smaller region, the number of states each particle visits is reduced. Therefore, this illustrates an example environment in which reducing the entropy of the visitation distribution results in an agent gaining the ability to do work. In the same way that Maxwell's demon accumulates free energy via information-gathering and manipulating of its environment, we would expect self-supervised agents guided by belief entropy minimization to accumulate the equivalent of potential energy in their corresponding sequential decision processes, which would lead them to gain control over their environment.

[1] In Maxwell's original example, the demon sorts the particles into the two chambers based on velocity. Our example is closely related to Szilard's engine (Szilard, 1929; Magnasco, 1996).

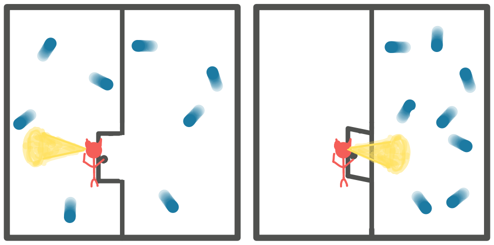


Figure 6.2: A "demon" gathering information to sort particles, reducing the entropy of the particle configuration.

6.2 Preliminaries

Our goal in this work will be to design self-supervised reinforcement learning methods in partially observed settings, which can acquire complex behaviors that both gather information and gain control over their environment. To this end, we will formulate the learning problem in the context of a discrete-time partially-observed controlled Markov process, also known as a controlled hidden Markov process (CHMP), which corresponds to a POMDP without a reward function. The CHMP is defined by a state space $\mathcal{S}$ with states $\mathbf{s} \in \mathcal{S}$, action space $\mathcal{A}$ with actions $\mathbf{a} \in \mathcal{A}$, transition dynamics $P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, observation space Ω with observations $\mathbf{o} \in \Omega$, and emission distribution $O(\mathbf{o}_t|\mathbf{s}_t)$. The agent is a policy $\pi(\mathbf{a}_t|\mathbf{o}_{\leq t})$. Note that it does *not* observe any states $\mathbf{s}$.

We denote the undiscounted finite-horizon state visitation as $d^\pi(\mathbf{s}) \doteq {}^1\!/_T \sum_{t=0}^{T-1} \Pr_\pi(\mathbf{s}_t = \mathbf{s})$, where $\Pr_\pi(\mathbf{s}_t = \mathbf{s})$ is the probability that $\mathbf{s}_t = \mathbf{s}$ after executing π for t steps. Using $d^\pi(\mathbf{s})$, we can quantify the average *disorder* of the environment with the Shannon entropy, $H(d^\pi(\mathbf{s}))$. Prior work proposes observational surprise minimization ($\min_\pi -\log \hat{p}(\mathbf{o})$) as an intrinsic control objective (Friston, 2009; Ueltzhöffer, 2018; Berseth et al., 2021); in Berseth et al. (2021) (SMiRL), the agent models the state visitation distribution, $d^\pi(\mathbf{s})$ with $\hat{p}(\mathbf{s})$, which it computes by assuming access to $\mathbf{s}$. In environments in which there are natural sources of variation outside of the agent, this incentivizes the SMiRL agent, fully aware of these variations observed through $\mathbf{s}$, to take action to control them. In a partially-observed setting, SMIRL's model becomes $\hat{p}(\mathbf{o})$, which generally will enable the agent to ignore variations that it can prevent from appearing in its observations. We observe this phenomenon in our experiments.

6.3 Control and Information Gathering via Belief Entropy Minimization

The main question that we tackle in the design of our algorithm is: how can we formulate a general and concise objective function that can enable an RL agent to gain control over its

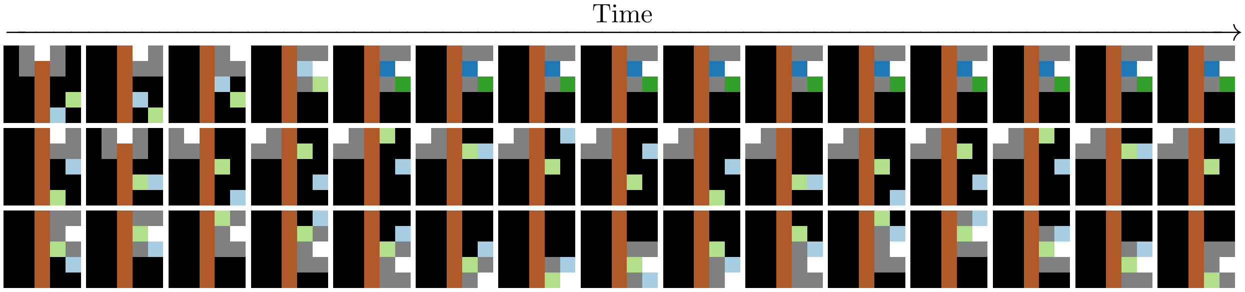


Figure 6.3: Comparison of several approaches on the TwoRoom environment. The agent, in white, can view a limited area around it, in grey, and can stop particles within its view and darken their color. The vertical wall, in brown, separates particles (blue and green) in the "busy room" (on right) from the "dark room" (on left). *Top*: Our approach seeks out the particles and stops them. *Middle:* The observational surprise minimization method in Berseth et al. (2021) leads the agent to frequently hide in the dark room, leaving the particles unstopped. *Bottom:* Latent-state infogain leads the agent to find and observe the particles, but not stop them.

partially-observed environment, in the absence of any user-provided task reward? Consider the following partially-observed "TwoRoom" environment, depicted in Fig. 6.3. The environment has two rooms: an empty ("dark") room on the left, and a "busy" room on the right, the latter containing two moving particles that move around until the agent "tags" them, which stops their motion. Intuitively, an agent that aims to gather information and gain control of its environment should search for the moving particles to find out where they are. However, it is difficult to observe both particles at the same time. A more effective strategy is to "tag" the particles – then, their position remains fixed, and the agent will know where they are at all times, even when they are not observed. This task can be seen as a simple analogy for more complex settings that can occur in natural environments, where we might want agents to arrange an environment in an orderly fashion. We can view this task as a rough analogy for a version of the previously-discussed Maxwell's Demon thought experiment.

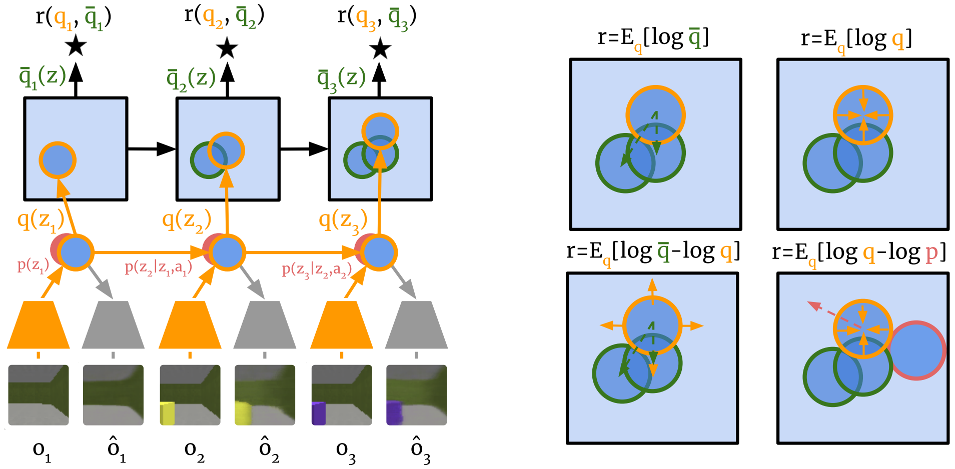
Representing variability with latent state-space models. In order for the agent to represent the dynamic components of the environment observed from images, our method involves learning a latent state-space model (LSSM) (Watter et al., 2015; Krishnan et al., 2015; Karl et al., 2016; Maddison et al., 2017; Hafner et al., 2018; Mirchev et al., 2018; Wayne et al., 2018; Vezzani et al., 2019; Lee et al., 2019; Das et al., 2019; Hafner et al., 2019; Mirchev et al., 2020; Rafailov et al., 2021). We intermittently refer to these dynamic components as "factors of variation" to distinguish the model's representation of variability in the environment (latent state) from the true variability (true state). At timestep t, the LSSM represents the agent's current *belief* as $q_\phi(\mathbf{z}_t|\mathbf{o}_{\leq t}, \mathbf{a}_{\leq t-1})$, where $\mathbf{z}_t$ is the model's latent state. We defer the description of the LSSM learning and architecture to Section 6.4, and now motivate how we will use the LSSM for constructing an intrinsic control objective.

Belief entropy and latent visitation entropy. Consider a policy that takes actions to minimize the entropy of the belief $H(q_\phi(\mathbf{z}_t|\mathbf{o}_{\leq t}, \mathbf{a}_{\leq t-1}))$. This corresponds to the agent performing *active state estimation* (Feder et al., 1999; Williams, 2007; Kreucher et al., 2005), and is equivalent to taking actions to maximize expected latent-state information gain $I(\mathbf{o}_t, \mathbf{z}_t|\mathbf{o}_{<t}, \mathbf{a}_t)$ (Aoki et al., 2011). However, active state estimation is satisfied by a policy that simply collects informative observations, as it does not further incentivize actions to "stabilize" the environment by preventing the latent state from changing. Analogous to the standard definition of undiscounted state visitation, consider the undiscounted latent visitation: $d^\pi(\mathbf{z}) \doteq {}^1\!/_T \sum_{t=0}^{T-1} \Pr_\pi(\mathbf{z}_t = \mathbf{z})$, where $\Pr_\pi(\mathbf{z}_t = \mathbf{z}) = \mathbb{E}_\pi q_\phi(\mathbf{z}_t|\mathbf{o}_{\leq t}, \mathbf{a}_{\leq t-1})$ (the expected belief after executing π for t timesteps). Our goal is to minimize $H(d^\pi(\mathbf{z}))$, as this corresponds to *stabilizing the agent's beliefs*, which incentivizes both reducing uncertainty in each $q_\phi(\mathbf{z}_t|\mathbf{o}_{\leq t}, \mathbf{a}_{\leq t-1})$, as well as constructing a niche such that each $q_\phi(\mathbf{z}_t|\mathbf{o}_{\leq t}, \mathbf{a}_{\leq t-1})$ concentrates probability on the same latent states.

Discovering factors of variation. In order for belief entropy minimization to incentivize the agent to control entities in the environment, the LSSM's belief must represent the underlying state variables in some way and model their uncertain evolution until either observed or controlled. For example, the demon in the thought experiment in Section 6.1 would have no incentive to gather the particles if it did not know that they existed. While sufficient random exploration may result in a good-enough LSSM, making this approach generally practical requires a suitable exploration strategy to collect the experience necessary to train an LSSM that represents all of the underlying factors of variation. To this end, we learn a separate exploratory policy to maximize *expected model information gain*, similar to Schmidhuber (2010); Houthooft et al. (2016); Gheshlaghi Azar et al. (2019); Sekar et al. (2020). Expected information gain about model parameters $\boldsymbol{\theta}$ is relative to a set of prior experience $\mathcal{D}$ and a partial trajectory $\mathbf{h}_{t-1}$, given as $I(\mathbf{o}_t; \boldsymbol{\theta}|\mathbf{h}_{t-1}, \mathcal{D}) = \mathbb{E}_{\mathbf{o}_t}\mathrm{KL}(p(\boldsymbol{\theta}|\mathbf{o}_t, \mathbf{h}_{t-1}, \mathcal{D}) \mid\mid p(\boldsymbol{\theta}|\mathbf{h}_{t-1}, \mathcal{D}))$. Note that model information gain is *distinct* from the information an agent may gather to reduce its belief entropy $H(q_\phi(\mathbf{z}_t|\mathbf{o}_{\leq t}, \mathbf{a}_{\leq t-1}))$ within the current episode. Computing the full model parameter prior $p(\boldsymbol{\theta}|\mathbf{h}_{t-1}, \mathcal{D})$ and posterior $p(\boldsymbol{\theta}|\mathbf{o}_t, \mathbf{h}_{t-1}, \mathcal{D})$ is generally computationally expensive, and also requires evaluating an expectation over observations – instead, we approximate this expected information gain following a method similar to Sekar et al. (2020): we use an ensemble of latent dynamics models, $\mathcal{E} = \{p_{\theta_i}(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{a}_{t-1})\}_{i=1}^K$ to compute the variance of latent states estimated by $q_\phi(\mathbf{z}_t|\mathbf{o}_{\leq t}, \mathbf{a}_{\leq t-1})$. We build the ensemble throughout training using the method of Izmailov et al. (2018). Thus, the exploration reward is given as: $r_e = \mathrm{Var}_{\{\theta_i\}}[\log p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{a}_{t-1})|\mathbf{z}_t \sim q_\phi]..$

6.4 The Believer Algorithm

Now we describe how we implement and combine the various components of our method into a learning algorithm for minimizing belief visitation entropy in CHMPs (reward-less



(a) Latent state-space and visitation models.

(b) Intrinsic rewards.

Figure 6.4: Figure of latent-state space model and rewards. *Left:* The model observes images, $\mathbf{o}_t$ to inform beliefs about latent states, $q_\phi(\mathbf{z}_t|\mathbf{o}_{\leq t}, \mathbf{a}_{\leq t-1})$, and observes actions to make one-step predictions $p(\mathbf{z}_{t+1}|\mathbf{z}_t, \mathbf{a}_t)$. Each belief is used to update the latent visitation, $\bar{q}_{t'}(\mathbf{z})$. *Right:* The beliefs and latent visitations can be combined into various reward functions. The solid arrows denote directions of belief expansion and contraction incentivized by rewards; the dotted arrows denote directions of belief translation incentivized by rewards.

POMDPs). The main components are the latent-state space model, the latent visitation model, and the exploration and control policies.

Latent state-space model. In our CHMP setting, the agent only has access to partial observations $\mathbf{o}$ of the true state \. In order to estimate a representation of states and beliefs, we employ a sequence-based Variational Autoencoder to learn latent variable belief, dynamics, and emission models. We formulate the variational posterior to be $q_\phi(\mathbf{z}_{1:T}|\mathbf{o}_{1:T}, \mathbf{a}_{1:T}) = \prod_{t=1}^{T} q_\phi(\mathbf{z}_t|\mathbf{o}_{\leq t}, \mathbf{a}_{\leq t-1})$ and the generative model as $p(\mathbf{z}_{1:T}, \mathbf{o}_{1:T}|\mathbf{a}_{1:T}) = \prod_{t=1}^{T} p_\theta(\mathbf{o}_t|\mathbf{z}_t) p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{a}_{t-1})$. Denoting $\mathbf{h}_t \doteq (\mathbf{o}_{\leq t}, \mathbf{a}_{\leq t-1})$, the log-evidence of the model and its lower-bound are:

$$\log\ p(\mathbf{o}_{1:T}|\mathbf{a}_{1:T-1}) = \log \mathbb{E}_{\mathbf{z}_{1:T} \sim p(\mathbf{z}_{1:T}|\mathbf{a}_{1:T})} \left[\prod_{t=1}^{T} \log p(\mathbf{o}_t|\mathbf{z}_t) \right]$$

$$\geq \mathcal{L}(\phi, \theta) = \sum_{t=1}^{T} \mathop{\mathbb{E}}_{q_\phi(\mathbf{z}_t|\mathbf{h}_t)} \left[p_\theta(\mathbf{o}_t|\mathbf{z}_t) \right] - \mathop{\mathbb{E}}_{q_\phi(\mathbf{z}_{t-1}|\mathbf{h}_{t-1})} \left[\mathrm{KL}(q_\phi(\mathbf{z}_t|\mathbf{h}_t) \mid\mid p_\theta(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{a}_{t-1}) \right] \quad (6.1)$$

Given a dataset $\mathcal{D} = \{(\mathbf{o}_{1:T}, \mathbf{a}_{1:T})_i\}_{i=1}^N$, Eq. 6.1 is used to train the model via $\max_{\phi,\theta} \mathbb{E}_{U(\mathcal{D})}\mathcal{L}(\phi, \theta)$. The focus of our work is not to further develop the performance of LSSMs; our method could be further improved with advances in the particular LSSM employed. In practice, we implemented an LSSM in PyTorch (Paszke et al., 2019) similar to the categorical LSSM architecture described in (Hafner et al., 2020). In this case, both the belief prior and belief posterior are distributions formed from products of K_1 categorical distributions over K_2 categories: $g(\mathbf{z}; \mathbf{v}) = \prod_{\kappa_1=1}^{K_1} \prod_{\kappa_2=1}^{K_2} \mathbf{v}_{\kappa_1,\kappa_2}^{\mathbf{z}_{\kappa_1,\kappa_2}}$, where the vector of $K_1 \cdot K_2$ parameters, $\mathbf{v}$, is predicted by neural networks: $\mathbf{v}_{\text{posterior}} = f_\phi(\mathbf{o}_{\leq t}, \mathbf{a}_t)$ for the posterior, and $\mathbf{v}_{\text{prior}} = f_\theta(\mathbf{o}_{<t}, \mathbf{a}_t)$ for the prior. We found it effective to construct f_ϕ and f_θ following the distribution-sharing decomposition described in (Karl et al., 2016, 2017; Das et al., 2019), in which the posterior is produced by transforming the parameters of the prior with a measurement update. We implemented the prior as an RNN, and the posterior as an MLP, and defer further details to the supplementary material.

Latent visitation model. Our agent cannot, in general, evaluate $d^\pi(\mathbf{z})$ or $H(d^\pi(\mathbf{z}))$; at best, it can approximate them. We do so by maintaining a within-episode estimate of $d^\pi(\mathbf{z})$ by constructing a mixture across the belief history samples $\bar{q}_{t'}(\mathbf{z}) = {}^1\!/_{t'} \sum_{t=0}^{t'} q_\phi(\mathbf{z}_t|\mathbf{o}_{\leq t}, \mathbf{a}_{\leq t-1})$. This corresponds to a mixture (across time) of single-sample estimates of each $\mathbb{E}_\pi q_\phi(\mathbf{z}_t|\mathbf{o}_{\leq t}, \mathbf{a}_{\leq t-1})$. Given $\bar{q}_{t'}(\mathbf{z})$, we have an estimate of the visitation of the policy, and we can use this as part or all of the reward signal of the agent. To implement $\bar{q}_{t'}(\mathbf{z})$, we experimented both approximating it by averaging each $\mathbf{v}$ and with recording every belief, and found that simply recording each belief sufficed.

Belief-based objectives

We now describe several rewards that can be constructed with the LSSM, and how they connect to our main objective. In what follows, we denote $q_\phi(\mathbf{z}_t|\mathbf{o}_{\leq t}, \mathbf{a}_{\leq t-1}) = q_t(\mathbf{z}_t)$ for brevity. While our primary objective is "niche creation" by minimizing the expected surprise of the latent visitation model, we also explore other intrinsic objectives that can be computed in terms of the LSSM and the latent visitation model. These rewards are visualized in Fig. 6.4.

Certainty. The entropy of the current belief measures the agent's certainty of the latent state, and by extension, about the aspects of the environment captured by the latent. Minimizing the belief entropy increases the agent's certainty. It is agnostic to predictable changes, and penalizes unpredictable changes. We define the *certainty reward* as the negative belief entropy:

$$r_t^c \doteq -\mathrm{H}(q_t(\mathbf{z}_t)) = \mathbb{E}_{q_t(\mathbf{z}_t)}[\log q_t(\mathbf{z}_t)] \tag{6.2}$$

Niche Creation: Our primary objective is the expected surprise of the latent visitation distribution, which measures how many states the agent believes it could have been in during the episode so far. Minimizing this reduces the number of visited environment states and increases the agent's certainty. We define the *niche creation reward* as the negative cross entropy of the visitation under the belief:

$$r_t^{nc} \doteq -\mathrm{H}(q_t(\mathbf{z}_t), \bar{q}_{t'}(\mathbf{z})) = \mathbb{E}_{q_t(\mathbf{z}_t)}[\log \bar{q}_{t'}(\mathbf{z})]. \tag{6.3}$$

Niche Expansion. Instead of minimizing the latent visitation entropy altogether, we can add an entropy bonus for the current belief to encourage exploration and thus potentially find a broader niche. The results in bringing the current belief towards the current latent visitation distribution. We define the *niche expansion reward* as the negative KL divergence:

$$r_t^{ne} \doteq -\mathrm{KL}(q_t(\mathbf{z}_t)||\bar{q}_{t'}(\mathbf{z})) = \mathbb{E}_{q_t(\mathbf{z}_t)}[\log \bar{q}_{t'}(\mathbf{z}) - \log q_t(\mathbf{z}_t)]. \tag{6.4}$$

State Infogain. The latent state information gain (not to be confused with the model information gain described in section 6.3) measures how much more certain the belief is compared to its temporal prior. Gaining information does not always coincide with being certain, because an infogain agent may cause chaotic events in the environment with outcomes that it only understands partially. As a result, it has gained more information than standing still but has also become less certain. We define the *infogain reward* as the KL divergence:

$$r_t^{i} \doteq \mathrm{KL}(q_t(\mathbf{z}_t)||p(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{a}_{t-1})) = \mathbb{E}_{q_t(\mathbf{z}_t)q_{t-1}(\mathbf{z}_{t-1})}[\log q_t(\mathbf{z}_t) - \log p(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{a}_{t-1})]. \tag{6.5}$$

Algorithm summary

Conceptual pseudocode for our belief-entropy based method, which we call Believer, is presented in Alg. 5. The algorithm begins by initializing the LSSM, q_ϕ and p_θ, as well as two separate policies: one trained to collect difficult-to-predict data with the exploration objective with rewards defined by r_e, π_e, and one trained to maximize one of the intrinsic control objectives as defined by eqs. (6.2) to (6.5).

We represent each policy $\pi(\mathbf{a}_t|\mathbf{v}_{\text{posterior},t})$ as a two-layer fully-connected MLP with 128 units. Recall that $\mathbf{v}_{\text{posterior}}$ is the vector of posterior parameters, which enables the policy to use the memory represented by the LSSM. We do not back-propagate the policies' losses to the LSSM for simplicity of implementation, although prior work does so in the case of a single policy (Lee et al., 2019). Our method is agnostic to the subroutine used to improve the policies. In our implementation, we employ PPO (Schulman et al., 2017).

6.5 Experiments

Our experiments are designed to answer the following questions: **Q1: Intrinsic control capability**: Does our latent visitation-based self-supervised reward signal cause the agent to stabilize partially-observed visual environments with dynamic entities more effectively than prior self-supervised stabilization objectives? **Q2: Properties of Believer objectives**: What types of emergent behaviors does each belief-based objective described in section 6.4 evoke?

Algorithm 5 Believer

```
procedure BELIEVER(Env; $L$, $M$, $N$, $L$)
Initialize $\pi_c$, $\pi_e, q_\phi, p_{\{\theta_i\}_{i=1}^K}, \mathcal{D} \leftarrow \emptyset$.
for episode $= 0, \dots, M$ do
    $\mathcal{D}_e \leftarrow \text{Collect}(N, \text{Env}, \pi_e); \mathcal{D}_c \leftarrow \text{Collect}(N, \text{Env}, \pi_c); \mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_e \cup \mathcal{D}_c$
    // LSSM fitting step.
    Update $q_\phi, p_\theta$ using SGD on Eq. 6.1 with $\mathcal{D}$ for $L$ rounds.
    Update $\pi_e$ using SGD on PPO's objective with rewards $r_e$
    // Intrinsic control optimization step.
    Update $\pi_c$ using SGD on PPO's objective with rewards defined by one of eqs. (6.2) to (6.5)
end for
```

In order to answer these questions, we identified environments with the following properties **(i):** partial observability, **(ii)**: dynamic entities that the agent can affect, and **(iii)**: high-dimensional observations. Because many standard RL benchmarks do not contain the significant partial-observability that is prevalent in the real world, it is challenging to answer these questions with them. Instead, we create several environments, and employ several existing environments we identified to have these properties. In what follows, we give an overview of the experimental settings and conclusions. We defer comprehensive details to the supplementary material.

Environments

TwoRoom. As previously described, this environment has two rooms: an empty ("dark") room on the left, and a "busy" room on the right, the latter containing moving particles that move around unless the agent "tags" them, which permanently stops their motion, as shown in Fig. 6.5. The agent can observe a small area around it, which it receives as an image. In this environment, control corresponds to finding and stopping the particles. The action space is $\mathcal{A} = \{\text{left, right, up, down, tag, no-op}\}$, and the observation space is normalized RGB-images: $\Omega = [0, 1]^{3\times30\times30}$. An agent that has significant control over this environment should tag particles to reduce the uncertainty over future states. To evaluate policies, we use the average fraction of total particles locked, the average fraction of particles visible, and the discrete true-state visitation entropy of the positions of the particles, $H(d^\pi(s_d))$. We employed two versions of this environment, with details provided in the supplementary material. In the large environment, the agent observes a 5x5 area around it as an image, and the busy room contains 5 particles.

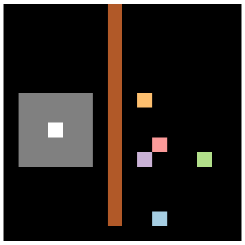


Figure 6.5: TwoRoom Large Environment.

VizDoom DefendTheCenter. The VizDoom DefendTheCenter environment shown in Fig. 6.6 is a circular arena in which a stationary agent, equipped with a partial field-of-view of the arena and a weapon, can rotate and shoot encroaching monsters (Kempka et al., 2016). The action space is $\mathcal{A} = \{\text{turn left, turn right, shoot}\}$, and the observation space is normalized RGB-images: $\Omega = [0, 1]^{3\times64\times64}$. In this environment, control corresponds to reducing the number of monsters by finding and shooting them. We use the average original environment return, (for which *no policy* has access to during training), the average number of killed monsters at the end of an episode, and the average number of visible monsters to measure the agent's control.

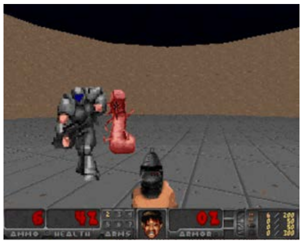


Figure 6.6: Vizdoom Defend The Center.

OneRoomCapture3D The MiniWorld framework is a customizable 3D environment simulator in which an agent perceives the world through a perspective camera (Chevalier-Boisvert, 2018). We used this framework to build the environment in Fig. 6.7 which an agent and a bouncing box both inhabit a large room; the agent can lock the box to stop it from moving if it is nearby, as well as constrain the motion of the box by standing nearby it. In this environment, control corresponds to finding the box and either trapping it near a wall, or tagging it. The action space is $\mathcal{A} = \{\text{turn left }\ 20^\circ, \text{turn right }\ 20^\circ, \text{move forward, move backward, tag}\}$, and observation space is normalized RGB-images: $\Omega = [0, 1]^{3\times64\times64}$. We use the average fraction of time the box is captured, the average time the box is visible, and continuous (Gaussian-estimated) true-state visitation entropy of the box's position $H(d^\pi(s_d))$ to measure the agent's ability to reduce entropy of the environment.

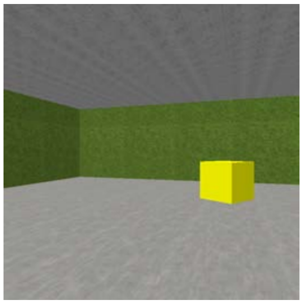


Figure 6.7: One Room Capture 3D.

Comparisons

In order to answer **Q1**, we compare to SMiRL (Berseth et al., 2021) and a recent empowerment method (Zhao et al., 2021) that estimates the current empowerment of the agent from an image. We use the empowerment estimate as a reward signal for training a policy. In order to answer **Q2**, we ensured each environment was instrumented with the aforementioned metrics of visibility and control, and deployed algorithm 5 separately with each of eqs. (6.2) to (6.5), as well a with simple sum of eq. (6.3) and eq. (6.5). Finally, we compare to a "random policy" that chooses actions by sampling from a uniform distribution over the action space, as well as an "oracle policy" that has access to privileged information about the environment state in each environment. We perform evaluation at $5e6$ environment steps with 50 policy rollouts per random seed, with 3 random seeds for each method (150 rollouts total).

Our primary results are presented in tables 6.1 and 6.2. We observe the Niche Creation+Infogain and Niche Expansion rewards to yield policies that exhibit a high degree

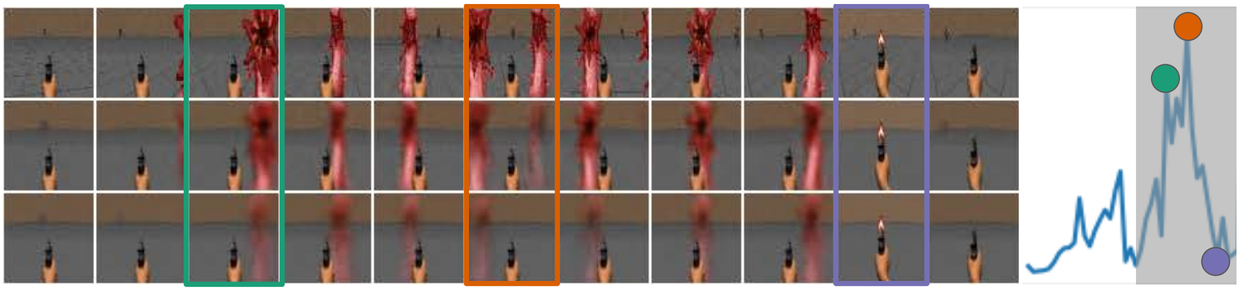


Figure 6.8: Visualization of a sequence in the VizDoom DefendTheLine environment. *Row 1:* The image provided to the agent. *Row 2:* The agent's reconstruction of a sample from q. *Row 3:* The agent's one-step image forecast. *Right:* The state infogain signal, $\mathbb{E}_q[\log q - \log p]$. Each colored rectangle identifies a keyframe that corresponds to a colored circle on the infogain plot. The infogain signal measures how much more certain the belief is compared to its temporal prior; when stochastic events happen (monster appears nearby), the signal is high; when the next image is predictable (monster disappears when shot), the signal is low.

of control over each environment – finding and stopping the moving objects, and finding and shooting the monsters, in the *complete absence of any extrinsic reward signal*. The Infogain-based agent generally seeks out and observes, but does not interfere with, the dynamic objects; the high Visibility metric and low control metrics (Lock, Capture, Kill) in each environment illustrates this. Qualitative results of this phenomenon are illustrated in fig. 6.8, in which the infogain signal is high when stochastically-moving monsters are visible, and low when they are not. We observe that in these partially observed environments the method of Berseth et al. (2021) tends to learn policies that hide from the dynamic objects in the environment, as indicated by the low control and visibility values of the final policy. Furthermore, we observe the method of Zhao et al. (2021) not to exhibit controlling behavior in these partially-observed environments, instead it views the dynamic objects in the TwoRoom and OneRoomCapture3D Environments somewhat more frequently than the random policy. We present videos of all policies in the supplementary material.

6.6 Related Work

Much of the previous work on learning without extrinsic rewards has been based either on (i) exploration (Chentanez et al., 2005; Oudeyer et al., 2007; Oudeyer and Kaplan, 2009), or (ii) some notion of intrinsic control, such as empowerment (Klyubin et al., 2005; Mohamed and Jimenez Rezende, 2015; Karl et al., 2017). Exploration approaches include those that maximize model prediction error or improvement (Schmidhuber, 1991; Lopes et al., 2012; Stadie et al., 2015; Pathak et al., 2017), maximize model uncertainty (Houthooft et al., 2016; Still and Precup, 2012; Shyam et al., 2018; Pathak et al., 2019; Gheshlaghi Azar et al., 2019), maximize state visitation (Bellemare et al., 2016; Fu et al., 2017; Tang et al., 2017;

Method	TwoRoom Environment			OneRoomCapture3D Environment			
	Obj. Lock↑	Obj. Visible↑	$H(d^\pi(s_d))\downarrow$	Obj. Captured↑	Obj. Tagged↑	Obj. Visible↑	$H(d^\pi(s_d))\downarrow$
Niche Creation+Infogain	0.92$_{\pm0.01}$	**0.94**$_{\pm0.01}$	0.33$_{\pm0.03}$	**0.84**$_{\pm0.08}$	0.00$_{\pm0.00}$	**0.88**$_{\pm0.02}$	$-0.06_{\pm0.16}$
Niche Expansion, eq. (6.4)	**0.95**$_{\pm0.00}$	0.66$_{\pm0.03}$	**0.22**$_{\pm0.02}$	0.73$_{\pm0.03}$	**0.71**$_{\pm0.03}$	0.67$_{\pm0.04}$	**-0.96**$_{\pm0.21}$
Niche Creation, eq. (6.3)	0.50$_{\pm0.05}$	0.46$_{\pm0.05}$	1.06$_{\pm0.09}$	0.39$_{\pm0.04}$	0.36$_{\pm0.04}$	0.01$_{\pm0.00}$	1.15$_{\pm0.16}$
Certainty, eq. (6.2)	0.06$_{\pm0.02}$	0.01$_{\pm0.00}$	1.86$_{\pm0.04}$	0.29$_{\pm0.04}$	0.27$_{\pm0.04}$	0.16$_{\pm0.03}$	0.66$_{\pm0.24}$
Infogain, eq. (6.5)	0.00$_{\pm0.00}$	0.55$_{\pm0.00}$	1.95$_{\pm0.02}$	0.57$_{\pm0.03}$	0.00$_{\pm0.00}$	0.54$_{\pm0.04}$	0.97$_{\pm0.14}$
SMiRL (Berseth et al., 2021)	0.25$_{\pm0.04}$	0.27$_{\pm0.03}$	1.52$_{\pm0.06}$	0.46$_{\pm0.04}$	0.45$_{\pm0.04}$	0.20$_{\pm0.02}$	$-0.02_{\pm0.28}$
Empowerment (Zhao et al., 2021)	0.00$_{\pm0.00}$	0.46$_{\pm0.03}$	1.95$_{\pm0.03}$	0.49$_{\pm0.04}$	-	0.22$_{\pm0.02}$	0.21$_{\pm0.25}$
Random policy	0.61$_{\pm0.03}$	0.28$_{\pm0.01}$	1.19$_{\pm0.06}$	0.54$_{\pm0.03}$	0.52$_{\pm0.04}$	0.16$_{\pm0.01}$	0.12$_{\pm0.22}$
Oracle lock policy	0.98$_{\pm0.00}$	0.91$_{\pm0.01}$	0.13$_{\pm0.01}$	0.95$_{\pm0.00}$	0.95$_{\pm0.00}$	0.96$_{\pm0.00}$	$-2.58_{\pm0.23}$

Table 6.1: Policy evaluation in TwoRoom and OneRoomCapture3D. Means and their standard errors are reported; grey shading denotes a variant of our method, bolding denotes where a method achieves the best mean performance under a metric. We observe that the Niche Expansion and Niche Creation+Infogain objectives lead the agent to seek out and stabilize the dynamic objects substantially more effectively than other methods.

Hazan et al., 2019), maximize surprise (Schmidhuber, 1991; Achiam and Sastry, 2017; Sun et al., 2011), and employ other novelty-based exploration bonuses (Lehman and Stanley, 2011; Burda et al., 2018; Kim et al., 2018, 2019). Our method can be combined with prior exploration techniques to aid in optimizing our proposed objective, and in that sense our work is largely orthogonal to prior exploration methods.

Prior works on intrinsic control include empowerment maximization (Klyubin et al., 2005,?; Mohamed and Rezende, 2015), observational surprise minimization (Friston, 2009; Friston et al., 2009; Ueltzhöffer, 2018; Berseth et al., 2021; Parr and Friston, 2019), and skill discovery (Barto et al., 2004; Konidaris and Barto, 2009; Gregor et al., 2016; Eysenbach et al., 2018; Sharma et al., 2019; Xu et al., 2020). Observational surprise minimization seeks policies that make *observations* predictable and controllable, and is closely connected to entropy minimization, as entropy is defined to be the expected surprise. In Friston et al. (2010), the notion of Free Energy Minimization corresponds to minimizing *observational* entropy, and states that the entropy of hidden states in the environment is bounded by the entropy of sensory observations. However, the proof assumes a diffeomorphism to hold between states and observations, which is explicitly violated in CHMPs and any real-world setting, as agents cannot perceive the state of anything outside their egocentric sensory observations. Similarly, empowerment (Klyubin et al., 2005; Karl et al., 2015, 2017; Zhao et al., 2021) is a measure of the degree of control an agent *could have* over future *observations*, whereas state visitation entropy is a measure of the degree of the control an agent *has* over the *underlying environment state*. Our approach seeks to infer and gain control over a representation of the environment's state, as opposed to the agent's observations. We demonstrate environments where minimizing observational surprise and maximizing empowerment leads to degenerate solutions that ignore important factors of variation, whereas our approach identifies and controls them.

Representation learning methods have been explored in a variety of prior work, including, but not limited to, (Lange and Riedmiller, 2010; Watter et al., 2015; Karl et al., 2016; Finn

Method	DefendTheCenter Environment		
	Env. Return↑	Monster Kills↑	Visible↑
Niche Creation+Infogain	1964.4±05.42	0.00±0.00	**1.16**±0.005
Niche Expansion, eq. (6.4)	**2506.4**±36.26	**28.2**±1.50	0.94±0.015
Niche Creation, eq. (6.3)	2182.2±14.12	10.0±0.55	0.76±0.017
Certainty, eq. (6.2)	1958.0±35.63	07.2±0.73	0.92±0.028
Infogain, eq. (6.5)	1928.6±35.51	00.2±00.2	1.15±0.008
SMiRL (Berseth et al., 2021)	1918.7±53.80	7.58±0.15	0.89±0.009
Empowerment (Zhao et al., 2021)	2161.8±29.06	16.2±2.35	0.86±0.064
Random policy	2113.8±37.60	14.2±0.73	0.90±0.026
Oracle policy	2550.8±55.90	28.0±1.52	0.80±0.035
	TwoRoom-Large Environment		
	Obj. Lock↑	Obj. Visible↑	$H(d^\pi(s_d))\downarrow$
Niche Creation+Infogain	**0.665**±0.013	**0.501**±0.013	**2.831**±0.083
SMiRL (Berseth et al., 2021)	0.110±0.020	0.087±0.016	3.417±0.022
Random policy	0.271±0.024	0.138±0.011	3.404±0.063
Oracle policy	0.885±0.005	0.464±0.017	1.016±0.043

Table 6.2: Policy evaluation in VizDoom and TwoRoom-Large. Means and their standard errors are reported; grey shading denotes a variant of our method, bolding denotes where a method achieves the best mean performance under a metric. We observe that the Niche Expansion objective in VizDoom and Niche Creation+Infogain objective in TwoRoom-Large lead the agent to seek out and stabilize the dynamic objects substantially more effectively than other methods.

et al., 2016; Nair et al., 2018; Zhang et al., 2018; Hafner et al., 2018; Lee et al., 2019). Our approach employs a representation learning method to build a latent state-space model (Watter et al., 2015; Krishnan et al., 2015; Karl et al., 2016; Hafner et al., 2018; Mirchev et al., 2018; Wayne et al., 2018; Vezzani et al., 2019; Lee et al., 2019; Das et al., 2019; Hafner et al., 2019; Mirchev et al., 2020; Rafailov et al., 2021).

6.7 Discussion

We presented a method, Believer, for intrinsically motivating an agent to discover, represent, and exercise control of dynamic objects in a partially-observed environments sensed with visual observations. We found that our method approached expert-level performance on several environments and substantially surpassed prior work in its unsupervised control capability. While our experiments represent a proof-of-concept that illustrates how latent state belief entropy minimization can incentivize an agent to both gather information and gain control over its environment, there are a number of exciting future directions. First, our method is inspired by a connection between thermodynamics and information theory, but the treatment of this connection is informal. Formalizing this connection could lead to an improved theoretical understanding of how Believer and other intrinsic motivation methods can lead to desirable behavior, and perhaps allow deriving conditions on environments under which such desirable behaviors would emerge. Second, Believer and other surprise-minimizing intrinsic motivation objectives are designed to work well in complex environments with unpredictable phenomena: a particularly interesting direction for future work is to scale up such methods in order to study the behavior that emerges.

Chapter 7

Conclusion

In this thesis, our goal was to make progress towards building general purpose agents that can efficiently solve any problem. Reinforcement learning provides a general framework to design these agents, and in particular we view internal latent dynamics models with model-based RL as a key component for building this generalization capability. We summarise our findings and discuss future work and open challenges.

In Chapter 3, we incorporated the concept of entities into a factorized latent dynamics model and showed that such a model can generalize and transfer knowledge about physics in a scenes with a few objects to new scenes with many objects. However, the concept of an entity or object is not fixed and is instead context dependent. For example, if opening a water bottle, we might consider the cap and the bottle as seperate objects, but when transporting the bottle, we might consider both pieces as a single object. This flexibility to view objects at different levels of abstraction depending on the task is still lacking in our agents and could enable representations that are less brittle and can adapt faster to new dynamics or tasks.

While Chapter 3 covered abstractions in space, Chapter 4, explored abstractions in space in the form of chaining together abstract skills for solving temporally extended problems. Chapter 4 did not operate on raw visual observations compared to Chapter 3 and an open challenge is to combine the strengths of both state abstraction and temporal abstraction. An agent that can build flexible state abstractions with the concept of entities and then reason over long horizon interactions between objects, could generalize over both complex visual input and temporally extended tasks.

While Chapter 3 and Chapter 4 showed that we can manually build in structural priors that can enable generalization, Chapter 5 discussed an evolutionary based method for automatically learning structure within our RL algorithms that can generalize across very different environments. Chapter 5 was limited in scope to value-based RL methods and an immediate future work would be to extend it to a broader family of RL algorithms. However, we could also extend this work for high level architectures to automatically learn the abstractions that were manually designed in Chapter 3 and Chapter 4. Entity abstractions and hierarchical RL might emerge among other structures as key components for general RL agents. Longer term, we might see machine guided algorithm and architecture design that is iterative and interactive with humans where the machine might suggest new promising areas of research to explore further.

Finally, while the previous chapters covered building in structure into latent dynamics models and a meta-optimization method for general RL agents, Chapter 6 covered the missing piece of the optimization objective. Latent state belief entropy minimization led to an agent gathering information and gaining control over its environment. An open question is whether such an approach might scale to more complex open ended world environments. For example, we might see an agent learn to build a house and acquire resources in Minecraft such that it is more empowered to control the environment and achieve any downstream task.

The role of the environment in developing complex behavior is still an underexplored area. While some progress has been made in characterizing properties of the environment that would make RL more tractable (Co-Reyes et al., 2020), an exciting area of research would be to combine the unsupervised objectives in Chapter 6 with large scale meta-learning in Chapter 5 but in open-ended environments that are structured for fast learning. A kind of curriculum or shaped environment could help make unsupervised RL and meta-learning efficient enough to create general purpose agents that exhibit complex behavior and adaptation to solve any task.

Bibliography

D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. *arXiv:1811.04551*, 2018.

C. Finn, I. Goodfellow, and S. Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pages 64–72, 2016.

T. Kulkarni, A. Gupta, C. Ionescu, S. Borgeaud, M. Reynolds, A. Zisserman, and V. Mnih. Unsupervised learning of object keypoints for perception and control. *arXiv:1906.11883*, 2019.

Z. Xu, Z. Liu, C. Sun, K. Murphy, W. T. Freeman, J. B. Tenenbaum, and J. Wu. Modeling parts, structure, and system dynamics via predictive learning. 2018.

N. Watters, L. Matthey, M. Bosnjak, C. P. Burgess, and A. Lerchner. Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration. *arXiv:1905.09275*, 2019.

P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261*, 2018.

M. Janner, S. Levine, W. T. Freeman, J. B. Tenenbaum, C. Finn, and J. Wu. Reasoning about physical interactions with object-oriented prediction and planning. *arXiv:1812.10972*, 2018.

F. Ebert, S. Dasari, A. X. Lee, S. Levine, and C. Finn. Robustness via retrying: Closed-loop robotic manipulation with self-supervised learning. *arXiv:1810.03043*, 2018.

G. Berseth, D. Geng, C. M. Devin, N. Rhinehart, C. Finn, D. Jayaraman, and S. Levine. {SM}irl: Surprise minimizing reinforcement learning in unstable environments. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=cPZOyoDloxl`.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik,

I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.*, 17(1):1334–1373, Jan. 2016.

D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

R. Paulus, C. Xiong, and R. Socher. A deep reinforced model for abstractive summarization. *ArXiv*, abs/1705.04304, 2018.

A. Mirhoseini, A. Goldie, M. Yazgan, J. Jiang, E. M. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi, J. Pak, A. Tong, K. Srinivasa, W. Hang, E. Tuncer, Q. V. Le, J. Laudon, R. Ho, R. Carpenter, and J. Dean. A graph placement methodology for fast chip design. *Nature*, 594 7862:207–212, 2021.

J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL `http://arxiv.org/abs/1502.05477`.

A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *CoRR*, abs/1708.02596, 2017.

I. Clavera, A. Nagabandi, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn. Learning to adapt: Meta-learning for model-based control. In *International Conference on Learning Representations*, 2019.

E. S. Spelke and K. D. Kinzler. Core knowledge. *Developmental science*, 10(1):89–96, 2007.

R. Baillargeon, E. Spelke, and S. Wasserman. Object permanence in five-month-old infants. *Cognition*, 20:191–208, 1985.

L. Acredolo and S. Goodwyn. Symbolic gesturing in normal infants. *Child development*, 59 2: 450–66, 1988.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60:84 – 90, 2012.

G. E. Hinton, L. Deng, D. Yu, G. E. Dahl, A. rahman Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29:82–97, 2012.

J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? *ArXiv*, abs/1411.1792, 2014.

K. Kansky, T. Silver, D. A. Mély, M. Eldawy, M. Lázaro-Gredilla, X. Lou, N. Dorfman, S. Sidor, S. Phoenix, and D. George. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. *arXiv:1706.04317*, 2017.

P. Tsividis, T. Pouncy, J. L. Xu, J. Tenenbaum, and S. Gershman. Human learning in atari. In *AAAI Spring Symposia*, 2017.

B. Lake and M. Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *ICML*, 2018.

K. Greff, S. van Steenkiste, and J. Schmidhuber. On the binding problem in artificial neural networks. *ArXiv*, abs/2012.05208, 2020.

P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *AAAI*, pages 1726–1734, 2017.

R. Veerapaneni, J. D. Co-Reyes, M. Chang, M. Janner, C. Finn, J. Wu, J. Tenenbaum, and S. Levine. Entity abstraction in visual model-based reinforcement learning. *ArXiv*, abs/1910.12827, 2019.

J. D. Co-Reyes, Y. Liu, A. Gupta, B. Eysenbach, P. Abbeel, and S. Levine. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In *ICML*, 2018.

J. D. Co-Reyes, Y. Miao, D. Peng, E. Real, S. Levine, Q. V. Le, H. Lee, and A. Faust. Evolving reinforcement learning algorithms. *ArXiv*, abs/2101.03958, 2021.

N. Rhinehart, J. Wang, G. Berseth, J. D. Co-Reyes, D. Hafner, C. Finn, and S. Levine. Intrinsic control of variational beliefs in dynamic partially-observed visual environments. In *ICML 2021 Workshop on Unsupervised Reinforcement Learning*, 2021. URL `https://openreview.net/forum?id=gb5rUScIY5x`.

M. D. Hoffman and D. M. Blei. Stochastic structured variational inference. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015, San Diego, California, USA, May 9-12, 2015*, 2015.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

J. Marino, Y. Yue, and S. Mandt. Iterative amortized inference. *arXiv:1807.09356*, 2018a.

J. Marino, M. Cvitkovic, and Y. Yue. A general method for amortizing variational filtering. In *Advances in Neural Information Processing Systems*, pages 7857–7868, 2018b.

K. Greff, R. L. Kaufmann, R. Kabra, N. Watters, C. Burgess, D. Zoran, L. Matthey, M. Botvinick, and A. Lerchner. Multi-object representation learning with iterative variational inference. *arXiv:1903.00450*, 2019.

N. Wichers, R. Villegas, D. Erhan, and H. Lee. Hierarchical long-term video prediction without supervision. *arXiv:1806.04768*, 2018.

E. L. Denton et al. Unsupervised learning of disentangled representations from video. In *Advances in neural information processing systems*, pages 4414–4423, 2017.

A. X. Lee, R. Zhang, F. Ebert, P. Abbeel, C. Finn, and S. Levine. Stochastic adversarial video prediction. *arXiv:1804.01523*, 2018.

J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in neural information processing systems*, pages 2863–2871, 2015.

C. Finn and S. Levine. Deep visual foresight for planning robot motion. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2786–2793. IEEE, 2017.

M. Zhang, S. Vikram, L. Smith, P. Abbeel, M. J. Johnson, and S. Levine. Solar: Deep structured latent representations for model-based reinforcement learning. *arXiv:1808.09105*, 2018.

J. Oh, V. Chockalingam, S. Singh, and H. Lee. Control of memory, active perception, and action in minecraft. *arXiv:1605.09128*, 2016.

K. Greff, R. K. Srivastava, and J. Schmidhuber. Binding via reconstruction clustering. *arXiv:1511.06418*, 2015.

F. Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, CORNELL AERONAUTICAL LAB INC BUFFALO NY, 1961.

W. F. Whitney, M. Chang, T. Kulkarni, and J. B. Tenenbaum. Understanding visual concepts with continuation learning. *arXiv:1602.06822*, 2016.

X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2172–2180, 2016.

T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum. Deep convolutional inverse graphics network. In *Advances in Neural Information Processing Systems*, pages 2539–2547, 2015.

V. Goel, J. Weng, and P. Poupart. Unsupervised video object segmentation for deep reinforcement learning. *arXiv:1805.07780*, 2018.

Z. Xu, Z. Liu, C. Sun, K. Murphy, W. T. Freeman, J. B. Tenenbaum, and J. Wu. Unsupervised discovery of parts, structure, and dynamics. *arXiv:1903.05136*, 2019.

M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv:1612.00341*, 2016.

P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems*, pages 4502–4510, 2016.

J. B. Hamrick, A. J. Ballard, R. Pascanu, O. Vinyals, N. Heess, and P. W. Battaglia. Metacontrol for adaptive imagination-based optimization. *arXiv:1705.02670*, 2017.

M. Janner, K. Narasimhan, and R. Barzilay. Representation learning for grounded spatial reasoning. *Transactions of the Association for Computational Linguistics*, 6:49–61, 2018.

K. Narasimhan, R. Barzilay, and T. Jaakkola. Grounding language for transfer in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 63:849–874, 2018.

V. Bapst, A. Sanchez-Gonzalez, C. Doersch, K. L. Stachenfeld, P. Kohli, P. W. Battaglia, and J. B. Hamrick. Structured agents for physical construction. *arXiv:1904.03177*, 2010.

A. Ajay, M. Bauza, J. Wu, N. Fazeli, J. B. Tenenbaum, A. Rodriguez, and L. P. Kaelbling. Combining physical simulators and object-based networks for control. *arXiv:1904.06580*, 2019.

R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

D. Wang, C. Devin, Q.-Z. Cai, F. Yu, and T. Darrell. Deep object centric policies for autonomous driving. *arXiv:1811.05432*, 2018.

W. Yang, X. Wang, A. Farhadi, A. Gupta, and R. Mottaghi. Visual semantic navigation using scene priors. *arXiv:1810.06543*, 2018.

K. Fragkiadaki, P. Agrawal, S. Levine, and J. Malik. Learning visual predictive models of physics for playing billiards. *arXiv:1511.07404*, 2015.

J. Wu, E. Lu, P. Kohli, B. Freeman, and J. Tenenbaum. Learning to see physics via visual de-animation. In *Advances in Neural Information Processing Systems*, pages 153–164, 2017.

A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap. A simple neural network module for relational reasoning. *arXiv:1706.01427*, 2017.

V. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. Reichert, T. Lillicrap, E. Lockhart, et al. Deep reinforcement learning with relational inductive biases. 2018.

Y. Du and K. Narasimhan. Task-agnostic dynamics priors for deep reinforcement learning. *arXiv:1905.04819*, 2019.

K. Greff, S. van Steenkiste, and J. Schmidhuber. Neural expectation maximization. 2017.

S. van Steenkiste, M. Chang, K. Greff, and J. Schmidhuber. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. *arXiv:1802.10353*, 2018.

G. Zhu, J. Wang, Z. Ren, and C. Zhang. Object-oriented dynamics learning through multi-level abstraction. *arXiv:1904.07482*, 2019.

S. A. Eslami, N. Heess, T. Weber, Y. Tassa, D. Szepesvari, G. E. Hinton, et al. Attend, infer, repeat: Fast scene understanding with generative models. In *Advances in Neural Information Processing Systems*, pages 3225–3233, 2016.

C. P. Burgess, L. Matthey, N. Watters, R. Kabra, I. Higgins, M. Botvinick, and A. Lerchner. Monet: Unsupervised scene decomposition and representation. *arXiv:1901.11390*, 2019.

A. R. Kosiorek, H. Kim, I. Posner, and Y. W. Teh. Sequential attend, infer, repeat: Generative modelling of moving objects. *arXiv:1806.01794*, 2018.

S. D. Levy and R. Gayler. Vector symbolic architectures: A new building material for artificial general intelligence. In *Conference on Artificial General Intelligence*, 2008.

P. Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation*, 2009.

P. Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence*, 46(1-2):159–216, 1990.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

T. Kipf, E. van der Pol, and M. Welling. Contrastive learning of structured world models. *arXiv preprint arXiv:1911.12247*, 2019.

Y. Zhang, J. Hare, and P.-B. Adam. Deep set prediction networks. *arXiv:1906.06565*, 2019.

S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

J. Fu, A. Singh, D. Ghosh, L. Yang, and S. Levine. Variational inverse control with events: A general framework for data-driven reward definition. In *Advances in Neural Information Processing Systems*, pages 8538–8547, 2018.

F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv:1812.00568*, 2018.

E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

R. Y. Rubinstein and D. P. Kroese. The cross-entropy method. In *Information Science and Statistics*, 2004.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

R. S. Sutton, D. Precup, and S. P. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2):181–211, 1999. doi: 10.1016/S0004-3702(99)00052-1.

P. Dayan and G. E. Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992]*, pages 271–278, 1992.

T. G. Dietterich. The MAXQ method for hierarchical reinforcement learning. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998), Madison, Wisconsin, USA, July 24-27, 1998*, pages 118–126, 1998.

R. Parr and S. J. Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems 10, [NIPS Conference, Denver, Colorado, USA, 1997]*, pages 1043–1049, 1997.

C. Florensa, Y. Duan, and A. Pieter. Stochastic neural networks for hierarchical reinforcement learning. In *ICLR*, 2017.

K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman. Meta learning shared hierarchies. *CoRR*, abs/1710.09767, 2017.

A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 3540–3549, 2017.

P. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 1726–1734, 2017.

K. Gregor, D. J. Rezende, and D. Wierstra. Variational intrinsic control. *CoRR*, abs/1611.07507, 2016.

R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. D. Turck, and P. Abbeel. VIME: variational information maximizing exploration. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 1109–1117, 2016.

J. Schulman, P. Abbeel, and X. Chen. Equivalence between policy gradients and soft q-learning. *CoRR*, abs/1704.06440, 2017a.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017b.

C. E. García, D. M. Prett, and M. Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335 – 348, 1989.

T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. Reinforcement learning with deep energy-based policies. *CoRR*, abs/1702.08165, 2017.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

J. Schmidhuber. Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes. *CoRR*, abs/0812.4360, 2008.

R. Fox, S. Krishnan, I. Stoica, and K. Goldberg. Multi-level discovery of deep options. *CoRR*, abs/1703.08294, 2017.

B. C. Stadie, S. Levine, and P. Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *CoRR*, abs/1507.00814, 2015.

A. L. Strehl and M. L. Littman. An analysis of model-based interval estimation for markov decision processes. *J. Comput. Syst. Sci.*, 74(8):1309–1331, 2008. doi: 10.1016/j.jcss.2007.08.009.

M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 1471–1479, 2016.

R. I. Brafman and M. Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, Mar. 2003. ISSN 1532-4435. doi: 10.1162/153244303765208377.

J. Fu, J. D. Co-Reyes, and S. Levine. EX2: exploration with exemplar models for deep reinforcement learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 2574–2584, 2017.

N. Heess, G. Wayne, Y. Tassa, T. P. Lillicrap, M. A. Riedmiller, and D. Silver. Learning and transfer of modulated locomotor controllers. *CoRR*, abs/1610.05182, 2016.

K. Hausman, J. T. Springenberg, N. H. Ziyu Wang, and M. Riedmiller. Learning an embedding space for transferable robot skills. In *Proceedings of the International Conference on Learning Representations, ICLR*, 2018.

N. Mishra, P. Abbeel, and I. Mordatch. Prediction and control with temporal segment models. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 2459–2468, 2017.

S. Mohamed and D. J. Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2125–2133, 2015.

J. Koza. Genetic programming - on the programming of computers by means of natural selection. In *Complex adaptive systems*, 1993.

E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *AAAI*, volume abs/1802.01548, 2019.

D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. *FOGA*, 1991.

E. Real, C. Liang, D. So, and Q. V. Le. Automl-zero: Evolving machine learning algorithms from scratch. In *ICML*, volume abs/2003.03384, 2020.

J. Schmidhuber. Evolutionary principles in self-referential learning. 1987.

J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975. second edition, 1992.

Y. Bengio, S. Bengio, and J. Cloutier. Learning a synaptic learning rule. *IJCNN-91-Seattle International Joint Conference on Neural Networks*, ii:969 vol.2–, 1991.

J. Schmidhuber. A self-referential weight matrix. 1993.

S. Bengio, Y. Bengio, and J. Cloutier. Use of genetic programming for the search of a new learning rule for neural networks. *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pages 324–327 vol.1, 1994.

L. Trujillo and G. Olague. Synthesis of interest point detectors through genetic programming. In *GECCO '06*, 2006.

F. Hutter, L. Kotthoff, and J. Vanschoren, editors. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2018. In press, available at http://automl.org/book.

K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 2902–2911. JMLR.org, 2017.

C. Liu, B. Zoph, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. L. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. *CoRR*, abs/1712.00559, 2017. URL `http://arxiv.org/abs/1712.00559`.

B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv:1611.01578*, 2016.

T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. In *ICML*, 2018.

B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. Efficient architecture search by network transformation. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 2787–2794, 2018.

I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le. Neural optimizer search with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 459–468. JMLR. org, 2017.

S. Whiteson and P. Stone. Evolutionary function approximation for reinforcement learning. *J. Mach. Learn. Res.*, 7:877–917, 2006.

S. Khadka and K. Tumer. Evolution-guided policy gradient in reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 1188–1200. Curran Associates, Inc., 2018.

A. Faust, A. Francis, and D. Mehta. Evolving rewards to automate reinforcement learning. *arXiv preprint arXiv:1905.07628*, 2019.

Y. Tang and K. Choromanski. Online hyper-parameter tuning in off-policy learning via evolutionary strategies, 2020.

J. K. H. Franke, G. Köhler, A. Biedenkapp, and F. Hutter. Sample-efficient automated deep reinforcement learning. 2020.

X. Song, K. Choromanski, J. Parker-Holder, Y. Tang, W. Gao, A. Pacchiano, T. Sarlos, D. Jain, and Y. Yang. Reinforcement learning with chromatic networks for compact architecture search. 2020.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *ArXiv*, abs/1703.03400, 2017.

C. Finn and S. Levine. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. *ArXiv*, abs/1710.11622, 2018.

Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. Rl2: Fast reinforcement learning via slow reinforcement learning. *ArXiv*, abs/1611.02779, 2016.

J. X. Wang, Z. Kurth-Nelson, H. Soyer, J. Z. Leibo, D. Tirumala, R. Munos, C. Blundell, D. Kumaran, and M. M. Botvinick. Learning to reinforcement learn. *ArXiv*, abs/1611.05763, 2017.

S. Reed and N. De Freitas. Neural programmer-interpreters. *arXiv:1511.06279*, 2015.

T. Pierrot, G. Ligner, S. Reed, O. Sigaud, N. Perrin, A. Laterre, D. Kas, K. Beguir, and N. de Freitas. Learning compositional neural programs with recursive tree search and planning. *NeurIPS*, 2019.

L. Kirsch, S. van Steenkiste, and J. Schmidhuber. Improving generalization in meta reinforcement learning using learned objectives. *ArXiv*, abs/1910.04098, 2020.

Y. Chebotar, A. Molchanov, S. Bechtle, L. Righetti, F. Meier, and G. S. Sukhatme. Meta-learning via learned loss. *ArXiv*, abs/1906.05374, 2019.

J. Oh, M. Hessel, W. M. Czarnecki, Z. Xu, H. van Hasselt, S. Singh, and D. Silver. Discovering reinforcement learning algorithms. *ArXiv*, abs/2007.08794, 2020.

R. Houthooft, R. Y. Chen, P. Isola, B. C. Stadie, F. Wolski, J. Ho, and P. Abbeel. Evolved policy gradients. *ArXiv*, abs/1802.04821, 2018.

F. Alet, M. F. Schneider, T. Lozano-Perez, and L. Kaelbling. Meta-learning curiosity algorithms. *ArXiv*, abs/2003.05325, 2020a.

F. Alet, M. F. Schneider, T. Lozano-Perez, and L. P. Kaelbling. Meta-learning curiosity algorithms. In *International Conference on Learning Representations*, 2020b. URL `https://openreview.net/forum?id=BygdyxHFDS`.

D. So, C. Liang, and Q. V. Le. The evolved transformer. *ArXiv*, abs/1901.11117, 2019.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602, 2013.

M. Chevalier-Boisvert, L. Willems, and S. Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *AAAI*, 2015.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *ArXiv*, abs/2006.04779, 2020.

N. Vieillard, O. Pietquin, and M. Geist. Munchausen reinforcement learning. *ArXiv*, abs/2007.14430, 2020.

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

S. Fujimoto, H. V. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. *ArXiv*, abs/1802.09477, 2018.

D. Hafner, P. A. Ortega, J. Ba, T. Parr, K. Friston, and N. Heess. Action and perception as divergence minimization. *arXiv preprint arXiv:2009.01791*, 2020.

K. J. Friston, J. Daunizeau, J. Kilner, and S. J. Kiebel. Action and behavior: a free-energy formulation. *Biological cybernetics*, 102(3):227–260, 2010.

K. Friston. Life as we know it. *Journal of the Royal Society Interface*, 10(86):20130475, 2013.

Z. Fountas, N. Sajid, P. A. Mediano, and K. Friston. Deep active inference agents using monte-carlo methods. *arXiv preprint arXiv:2006.04176*, 2020.

J. C. Maxwell and P. Pesic. *Theory of heat*. Courier Corporation, 2001.

H. S. Leff and A. F. Rex. *Maxwell's demon: entropy, information, computing*. Princeton University Press, 2014.

L. Szilard. Über die entropieverminderung in einem thermodynamischen system bei eingriffen intelligenter wesen. *Zeitschrift für Physik*, 53(11):840–856, 1929.

M. Magnasco. Szilard's heat engine. *EPL (Europhysics Letters)*, 33(8):583, 1996.

C. H. Bennett. The thermodynamics of computation—a review. *International Journal of Theoretical Physics*, 21(12):905–940, 1982.

K. Friston. The free-energy principle: a rough guide to the brain? *Trends in cognitive sciences*, 13(7):293–301, 2009.

K. Ueltzhöffer. Deep active inference. *Biological Cybernetics*, 112(6):547–573, 2018.

M. Watter, J. T. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. *arXiv preprint arXiv:1506.07365*, 2015.

R. G. Krishnan, U. Shalit, and D. Sontag. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

M. Karl, M. Soelch, J. Bayer, and P. van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. *arXiv preprint arXiv:1605.06432*, 2016.

C. J. Maddison, D. Lawson, G. Tucker, N. Heess, M. Norouzi, A. Mnih, A. Doucet, and Y. W. Teh. Filtering variational objectives. *arXiv preprint arXiv:1705.09279*, 2017.

A. Mirchev, B. Kayalibay, M. Soelch, P. van der Smagt, and J. Bayer. Approximate bayesian inference in spatial environments. *arXiv preprint arXiv:1805.07206*, 2018.

G. Wayne, C.-C. Hung, D. Amos, M. Mirza, A. Ahuja, A. Grabska-Barwinska, J. Rae, P. Mirowski, J. Z. Leibo, A. Santoro, et al. Unsupervised predictive memory in a goal-directed agent. *arXiv preprint arXiv:1803.10760*, 2018.

G. Vezzani, A. Gupta, L. Natale, and P. Abbeel. Learning latent state representation for speeding up exploration. *arXiv preprint arXiv:1905.12621*, 2019.

A. X. Lee, A. Nagabandi, P. Abbeel, and S. Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *arXiv preprint arXiv:1907.00953*, 2019.

N. Das, M. Karl, P. Becker-Ehmck, and P. van der Smagt. Beta dvbf: Learning state-space models for control from high dimensional observations. *arXiv preprint arXiv:1911.00756*, 2019.

D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

A. Mirchev, B. Kayalibay, P. van der Smagt, and J. Bayer. Variational state-space models for localisation and dense 3d mapping in 6 dof. *arXiv preprint arXiv:2006.10178*, 2020.

R. Rafailov, T. Yu, A. Rajeswaran, and C. Finn. Offline reinforcement learning from images with latent space models. *Learning for Decision Making and Control (L4DC)*, 2021.

H. J. S. Feder, J. J. Leonard, and C. M. Smith. Adaptive mobile robot navigation and mapping. *The International Journal of Robotics Research*, 18(7):650–668, 1999.

J. L. Williams. *Information Theoretic Sensor Management*. PhD thesis, Massachusetts Institute of Technology, 2007.

C. Kreucher, K. Kastella, and A. O. Hero Iii. Sensor management using an active sensing approach. *Signal Processing*, 85(3):607–624, 2005.

E. H. Aoki, A. Bagchi, P. Mandal, and Y. Boers. A theoretical look at information-driven sensor management criteria. In *14th International Conference on Information Fusion*, pages 1–8. IEEE, 2011.

J. Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247, 2010.

R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. VIME: Variational Information Maximizing Exploration. 2016. URL `http://arxiv.org/abs/1605.09674`.

M. Gheshlaghi Azar, B. Piot, B. Avila Pires, J.-B. Grill, F. Altché, and R. Munos. World discovery models. *arXiv e-prints*, pages arXiv–1902, 2019.

R. Sekar, O. Rybkin, K. Daniilidis, P. Abbeel, D. Hafner, and D. Pathak. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pages 8583–8592. PMLR, 2020.

P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

M. Karl, M. Soelch, P. Becker-Ehmck, D. Benbouzid, P. van der Smagt, and J. Bayer. Unsupervised real-time control through variational empowerment. *arXiv preprint arXiv:1710.05101*, 2017.

M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2016.

M. Chevalier-Boisvert. gym-miniworld environment for openai gym. `https://github.com/maximecb/gym-miniworld`, 2018.

R. Zhao, K. Lu, P. Abbeel, and S. Tiomkin. Efficient empowerment estimation for unsupervised stabilization. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=u2YNJPcQlwq`.

N. Chentanez, A. G. Barto, and S. P. Singh. Intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pages 1281–1288, 2005.

P.-Y. Oudeyer, F. Kaplan, and V. V. Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, 11(2):265–286, 2007.

P.-Y. Oudeyer and F. Kaplan. What is intrinsic motivation? a typology of computational approaches. *Frontiers in neurorobotics*, 1:6, 2009.

A. S. Klyubin, D. Polani, and C. L. Nehaniv. All else being equal be empowered. In M. S. Capcarrère, A. A. Freitas, P. J. Bentley, C. G. Johnson, and J. Timmis, editors, *Advances in Artificial Life*, pages 744–753, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31816-3.

S. Mohamed and D. Jimenez Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2125–2133. Curran Associates, Inc., 2015.

J. Schmidhuber. Curious model-building control systems. In *Proc. international joint conference on neural networks*, pages 1458–1463, 1991.

M. Lopes, T. Lang, M. Toussaint, and P.-Y. Oudeyer. Exploration in model-based reinforcement learning by empirically estimating learning progress. In *Advances in neural information processing systems*, pages 206–214, 2012.

B. C. Stadie, S. Levine, and P. Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven Exploration by Self-supervised Prediction. 2017.

S. Still and D. Precup. An information-theoretic approach to curiosity-driven reinforcement learning. *Theory in Biosciences*, 131(3):139–148, 2012.

P. Shyam, W. Jaśkowski, and F. Gomez. Model-based active exploration. *arXiv preprint arXiv:1810.12162*, 2018.

D. Pathak, D. Gandhi, and A. Gupta. Self-Supervised Exploration via Disagreement. 2019.

M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

J. Fu, J. Co-Reyes, and S. Levine. Ex2: Exploration with exemplar models for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2577–2587, 2017.

H. Tang, R. Houthooft, D. Foote, A. Stooke, O. X. Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in neural information processing systems*, pages 2753–2762, 2017.

E. Hazan, S. Kakade, K. Singh, and A. Van Soest. Provably efficient maximum entropy exploration. In *International Conference on Machine Learning*, pages 2681–2691. PMLR, 2019.

J. Achiam and S. Sastry. Surprise-based intrinsic motivation for deep reinforcement learning. *arXiv preprint arXiv:1703.01732*, 2017.

Y. Sun, F. Gomez, and J. Schmidhuber. Planning to be surprised: Optimal bayesian exploration in dynamic environments. In *International Conference on Artificial General Intelligence*, pages 41–51. Springer, 2011.

J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros. Large-Scale Study of Curiosity-Driven Learning. 2018. URL `http://arxiv.org/abs/1808.04355`.

H. Kim, J. Kim, Y. Jeong, S. Levine, and H. O. Song. Emi: Exploration with mutual information. *arXiv preprint arXiv:1810.01176*, 2018.

Y. Kim, W. Nam, H. Kim, J.-H. Kim, and G. Kim. Curiosity-bottleneck: Exploration by distilling task-specific novelty. In *International Conference on Machine Learning*, pages 3379–3388, 2019.

A. S. Klyubin, D. Polani, and C. L. Nehaniv. Empowerment: A universal agent-centric measure of control. In *2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 128–135. IEEE, 2005.

S. Mohamed and D. J. Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pages 2125–2133, 2015.

K. J. Friston, J. Daunizeau, and S. J. Kiebel. Reinforcement learning or active inference? *PLOS ONE*, 4(7):1–13, 07 2009. doi: 10.1371/journal.pone.0006421. URL `https://doi.org/10.1371/journal.pone.0006421`.

T. Parr and K. J. Friston. Generalised free energy and active inference. *Biological Cybernetics*, 113(5):495–513, Dec 2019. ISSN 1432-0770. doi: 10.1007/s00422-019-00805-w. URL `https://doi.org/10.1007/s00422-019-00805-w`.

A. G. Barto, S. Singh, and N. Chentanez. Intrinsically motivated learning of hierarchical collections of skills. In *Proceedings of the 3rd International Conference on Development and Learning*, pages 112–19. Piscataway, NJ, 2004.

G. Konidaris and A. G. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in neural information processing systems*, pages 1015–1023, 2009.

K. Gregor, D. J. Rezende, and D. Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

A. Sharma, S. Gu, S. Levine, V. Kumar, and K. Hausman. Dynamics-aware unsupervised discovery of skills. *arXiv preprint arXiv:1907.01657*, 2019.

K. Xu, S. Verma, C. Finn, and S. Levine. Continual learning of control primitives: Skill discovery via reset-games. *arXiv preprint arXiv:2011.05286*, 2020.

M. Karl, J. Bayer, and P. van der Smagt. Efficient empowerment. *arXiv preprint arXiv:1509.08455*, 2015.

S. Lange and M. Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2010.

C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016.

A. V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems*, pages 9191–9200, 2018.

J. D. Co-Reyes, S. Sanjeev, G. Berseth, A. Gupta, and S. Levine. Ecological reinforcement learning. *ArXiv*, abs/2006.12478, 2020.

C. Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv:1312.6114*, 2013.

D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

R. Pascanu, T. Mikolov, and Y. Bengio. Understanding the exploding gradient problem. *ArXiv*, abs/1211.5063, 2012.

I. Sutskever. *Training recurrent neural networks*. University of Toronto Toronto, Canada, 2013.

L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

Appendix A

OP3 Details

A.1 Observation Model

The observation model $\mathcal{G}$ models how the objects $H_{1:K}$ cause the image observation $X \in \mathbb{R}^{N \times M}$. Here we provide a mechanistic justification for our choice of observation model by formulating the observation model as a probabilistic approximation to a deterministic rendering engine.

Deterministic rendering engine: Each object H_k is rendered independently as the sub-image I_k and the resulting K sub-images are combined to form the final image observation X. To combine the sub-images, each pixel $I_{k(ij)}$ in each sub-image is assigned a depth $\delta_{k(ij)}$ that specifies the distance of object k from the camera at coordinate (ij). of the image plane. Thus the pixel $X_{(ij)}$ takes on the value of its corresponding pixel $I_{k(ij)}$ in the sub-image I_k if object k is closest to the camera than the other objects, such that

$$X_{(ij)} = \sum_{k=1}^{K} Z_{k(ij)} \cdot I_{k(ij)}, \tag{A.1}$$

where $Z_{k(ij)}$ is the indicator random variable $\mathbb{1}[k = \arg\min_{k \in K} \delta_{k(ij)}]$, allowing us to intuitively interpret Z_k as segmentation masks and I_k as color maps.

Modeling uncertainty with the observation model: In reality we do not directly observe the depth values, so we must construct a probabilistic model to model our uncertainty:

$$\mathcal{G}\left(X|H_{1:K}\right) = \prod_{i,j=1}^{N,M} \sum_{k=1}^{K} m_{(ij)}(H_k) \cdot g\left(X_{(ij)} \,|\, H_k\right), \tag{A.2}$$

where every pixel (ij) is modeled through a set of mixture components $g\left(X_{(ij)} \,|\, H_k\right) := p\left(X_{ij}|Z_{k(ij)} = 1, H_k\right)$ that model how pixels of the individual sub-images I_k are generated, as well as through the mixture weights $m_{ij}(H_k) := p\left(Z_{k(ij)} = 1|H_k\right)$ that model which point of each object is closest to the camera.

A.2 Evidence Lower Bound

Here we provide a derivation of the evidence lower bound. We begin with the log probability of the observations $X^{(1:T)}$ conditioned on a sequence of actions $a^{(0:T-1)}$:

$$\begin{aligned}
\log p\left(X^{(0:T)} \,\middle|\, a^{(0:T-1)}\right) &= \log \int_{h_{1:K}^{(0:T)}} p\left(X^{(0:T)}, h_{1:K}^{(0:T)} \,\middle|\, a^{(0:T-1)}\right) \, dh_{1:K}^{(0:T)}. \\
&= \log \int_{h_{1:K}^{(0:T)}} p\left(X^{(0:T)}, h_{1:K}^{(0:T)} \,\middle|\, a^{(0:T-1)}\right) \frac{q\left(h_{1:K}^{(0:T)} \mid \cdot\right)}{q\left(h_{1:K}^{(0:T)} \mid \cdot\right)} \, dh_{1:K}^{(0:T)}. \\
&= \log \mathbb{E}_{h_{1:K}^{(0:T)} \sim q\left(H_{1:K}^{(0:T)} \mid \cdot\right)} \left[\frac{p\left(X^{(0:T)}, h_{1:K}^{(0:T)} \,\middle|\, a^{(0:T-1)}\right)}{q\left(h_{1:K}^{(0:T)} \mid \cdot\right)} \right] \\
&\geq \mathbb{E}_{h_{1:K}^{(0:T)} \sim q\left(H_{1:K}^{(0:T)} \mid \cdot\right)} \log \left[\frac{p\left(X^{(0:T)}, h_{1:K}^{(0:T)} \,\middle|\, a^{(0:T-1)}\right)}{q\left(h_{1:K}^{(0:T)} \mid \cdot\right)} \right]. \qquad \text{(A.3)}
\end{aligned}$$

We have freedom to choose the approximating distribution $q\left(H_{1:K}^{(0:T)} \mid \cdot\right)$ so we choose it to be conditioned on the past states and actions, factorized across time:

$$q\left(H_{1:K}^{(0:T)} \mid x^{(0:T)}, a^{(0:T)}\right) = q\left(H_{1:K}^{(0)} \mid x^{(0)}\right) \prod_{t=1}^{T} q\left(H_{1:K}^{(t)} \mid H_{1:K}^{(t-1)}, x^{(t)}, a^{(t-1)}\right)$$

With this factorization, we can use linearity of expectation to decouple Equation A.3 across timesteps:

$$\mathbb{E}_{h_{1:K}^{(0:T)} \sim q\left(H_{1:K}^{(0:T)} \mid x^{(0:T)}, a^{(0:T)}\right)} \log \left[\frac{p\left(X^{(0:T)}, h_{1:K}^{(0:T)} \,\middle|\, a^{(0:T-1)}\right)}{q\left(h_{1:K}^{(0:T)} \mid x^{(0:T)}, a^{(0:T)}\right)} \right] = \sum_{t=0}^{(t)} \mathcal{L}_{\mathrm{r}}^{(t)} - \mathcal{L}_{\mathrm{c}}^{(t)},$$

where at the first timestep

$$\begin{aligned}
\mathcal{L}_{\mathrm{r}}^{(0)} &= \mathbb{E}_{h_{1:K}^{(0)} \sim q\left(H_{1:K}^{(0)} \mid X^{(0)}\right)} \left[\log p\left(X^{(0)} \mid h_{1:K}^{(0)}\right) \right] \\
\mathcal{L}_{\mathrm{c}}^{(0)} &= D_{KL}\left(q\left(H_{1:K}^{(0)} \mid X^{(0)}\right) || \, p\left(H_{1:K}^{(0)}\right) \right)
\end{aligned}$$

and at subsequent timesteps

$$\begin{aligned}
\mathcal{L}_{\mathrm{r}}^{(t)} &= \mathbb{E}_{h_{1:K}^{(t)} \sim q\left(H_{1:K}^{(t)} \mid h_{1:K}^{(0:t-1)}, X^{(0:t)}, a^{(0:t-1)}\right)} \left[\log p\left(X^{(t)} \mid h_{1:K}^{(t)}\right) \right] \\
\mathcal{L}_{\mathrm{c}}^{(t)} &= \mathbb{E}_{h_{1:K}^{(t-1)} \sim q\left(H_{1:K}^{(t-1)} \mid h_{1:K}^{(0:t-2)}, X^{(1:t-1)}, a^{(0:t-2)}\right)} \left[D_{KL}\left(q\left(H_{1:K}^{(t)} \mid h_{1:K}^{(t-1)}, X^{(t)}, a^{(t-1)}\right) || \, p\left(H_{1:K}^{(t)} \mid h_{1:K}^{(t-1)}, a^{(t-1)}\right) \right) \right].
\end{aligned}$$

By the Markov property, the marginal $q(H_{1:K}^{(t)} \mid h_{1:K}^{(0:t-1)}, X^{(0:t)}, a^{(0:t-1)})$ is computed recursively as

$$\mathbb{E}_{h^{(t-1)} \sim q\left(H_{1:K}^{(t-1)} \mid h_{1:K}^{(0:t-2)}, X^{(0:t-1)}, a^{(0:t-2)}\right)} \left[q\left(H_{1:K}^{(t)} \mid h_{1:K}^{(t-1)}, X^{(t)}, a^{(t-1)}\right) \right]$$

whose base case is $q\left(H^{(0)} \mid X^{(0)}\right)$ when $t = 0$.

We approximate observation distribution $p(X \mid H_{1:K})$ and the dynamics distribution $p(H'_{1:K} \mid H_{1:K}, a)$ by learning the parameters of the observation model $\mathcal{G}$ and dynamics model $\mathcal{D}$ respectively as outputs of neural networks. We approximate the recognition distribution $q(H_{1:K}^{(t)} \mid h_{1:K}^{(t-1)}, X^{(t)}, a^{(t-1)})$ via an inference procedure that refines better estimates of the posterior parameters, computed as an output of a neural network. To compute the expectation in the marginal $q(H_{1:K}^{(t)} \mid h_{1:K}^{(0:t-1)}, X^{(0:t)}, a^{(0:t-1)})$, we follow standard practice in amortized variational inference by approximating the expectation with a single sample of the sequence $h_{1:K}^{(0:t-1)}$ by sequentially sampling the latents for one timestep given latents from the previous timestep, and optimizing the ELBO via stochastic gradient ascent (Doersch, 2016; Kingma and Welling, 2013; Rezende et al., 2014).

A.3 Posterior Predictive Distribution

Here we provide a derivation of the posterior predictive distribution for the dynamic latent variable model with multiple latent states. Section A.2 described how we compute the distributions $p(X \mid H_{1:K})$, $p(H'_{1:K} \mid H_{1:K}, a)$, $q(H_{1:K}^{(t)} \mid h_{1:K}^{(t-1)}, X^{(t)}, a^{(t-1)})$, and $q(H_{1:K}^{(0:T)} \mid x^{(1:T)}, a^{(1:T)})$. Here we show that these distributions can be used to approximate the predictive posterior distribution $p(X^{(T+1:T+d)} \mid x^{(0:T)}, a^{(0:T+d)})$ by maximizing the following lower bound:

$$\begin{aligned}
\log p\left(X^{(T+1:T+d)} \mid x^{(0:T)}, a^{(0:T+d)}\right) &= \int_{h_{1:K}^{(0:T+d)}} p\left(X^{(T+1:T+d)}, h_{1:K}^{(0:T+d)} \mid x^{(0:T)}, a^{(0:T+d)}\right)\, dh_{1:K}^{(0:T+d)} \\
&= \int_{h_{1:K}^{(0:T+d)}} p\left(X^{(T+1:T+d)}, h_{1:K}^{(0:T+d)} \mid x^{(0:T)}, a^{(0:T+d)}\right) \frac{q\left(h_{1:K}^{(0:T+d)} \mid \cdot\right)}{q\left(h_{1:K}^{(0:T+d)} \mid \cdot\right)}\, dh_{1:K}^{(0:T+d)} \\
&= \log \mathbb{E}_{h_{1:K}^{(0:T+d)} \sim q\left(H_{1:K}^{(0:T+d)} \mid \cdot\right)} \left[\frac{p\left(X^{(T+1:T+d)}, h_{1:K}^{(0:T+d)} \mid x^{(0:T)}, a^{(0:T+d)}\right)}{q\left(h_{1:K}^{(0:T+d)} \mid \cdot\right)}\right] \\
&\geq \mathbb{E}_{h_{1:K}^{(0:T+d)} \sim q\left(H_{1:K}^{(0:T+d)} \mid \cdot\right)} \log \left[\frac{p\left(X^{(T+1:T+d)}, h_{1:K}^{(0:T+d)} \mid x^{(0:T)}, a^{(0:T+d)}\right)}{q\left(h_{1:K}^{(0:T+d)} \mid \cdot\right)}\right].
\end{aligned} \tag{A.4}$$

The numerator $p(X^{(T+1:T+d)}, h_{1:K}^{(0:T+d)} \mid x^{(0:T)}, a^{(0:T+d)})$ can be decomposed into two terms, one of which involving the posterior $p(h_{1:K}^{(0:T+d)} \mid x^{(0:T)}, a^{(0:T+d)})$:

$$p\left(X^{(T+1:T+d)}, h_{1:K}^{(0:T+d)} \mid x^{(0:T)}, a^{(0:T+d)}\right) = p\left(X^{(T+1:T+d)} \mid h_{1:K}^{(0:T+d)}\right) p\left(h_{1:K}^{(0:T+d)} \mid x^{(0:T)}, a^{(0:T+d)}\right),$$

This allows Equation A.4 to be broken up into two terms:

$$\mathbb{E}_{h_{1:K}^{(0:T+d)} \sim q\left(H_{1:K}^{(0:T+d)} \mid \cdot\right)} \log p\left(X^{(T+1:T+d)} \mid h_{1:K}^{(0:T+d)}\right) - D_{KL}\left(q\left(H_{1:K}^{(0:T+d)} \mid \cdot\right) || \, p\left(H_{1:K}^{(0:T+d)} \mid x^{(0:T)}, a^{(0:T+d)}\right)\right)$$

Maximizing the second term, the negative KL-divergence between the variational distribution $q(H_{1:K}^{(0:T+d)} \mid \cdot)$ and the posterior $p(H_{1:K}^{(0:T+d)} \mid x^{(0:T)}, a^{(0:T+d)})$ is the same as maximizing the following lower bound:

$$\mathbb{E}_{h_{1:K}^{(0:T)} \sim q\left(h_{1:K}^{(0:T)} \mid \cdot\right)} \log p\left(x^{(0:T)} \mid h_{1:K}^{(0:T)}, a^{(0:T-1)}\right) - D_{KL}\left(q\left(H_{1:K}^{(0:T+d)} \mid \cdot\right) || \, p\left(H_{1:K}^{(0:T+d)} \mid a^{(0:T+d)}\right)\right) \tag{A.5}$$

where the first term is due to the conditional independence between $X^{(0:T)}$ and the future states $H_{1:K}^{(T+1:T+d)}$ and actions $A^{(T+1:T+d)}$. We choose to express $q\left(H_{1:K}^{(0:T+d)} \mid \cdot\right)$ as conditioned on past states and actions, factorized across time:

$$q\left(H^{(0:T+d)} \mid x^{(0:T)}, a^{(0:T+d-1)}\right) = q\left(H_{1:K}^{(0)} \mid x^{(0)}\right) \prod_{t=1}^{T+d} q\left(H_{1:K}^{(t)} \mid H_{1:K}^{(t-1)}, x^{(t)}, a^{(t-1)}\right).$$

In summary, Equation A.4 can be expressed as

$$\begin{aligned}
&\mathbb{E}_{h_{1:K}^{(0:T+d)} \sim q\left(H^{(0:T+d)} \mid x^{(0:T)}, a^{(0:T+d-1)}\right)} \log p\left(X^{(T+1:T+d)} \mid h_{1:K}^{(0:T+d)}\right) \\
&+\mathbb{E}_{h_{1:K}^{(0:T)} \sim q\left(H^{(0:T)} \mid x^{(0:T)}, a^{(0:T-1)}\right)} \log p\left(x^{(0:T)} \mid h_{1:K}^{(0:T)}, a^{(0:T-1)}\right) \\
&-D_{KL}\left(q\left(H^{(0:T+d)} \mid x^{(0:T)}, a^{(0:T+d-1)}\right) || \, p\left(H_{1:K}^{(0:T+d)} \mid a^{(0:T+d)}\right)\right)
\end{aligned}$$

which can be interpreted as the standard ELBO objective for timesteps $0 : T$, plus an addition reconstruction term for timesteps $T + 1 : T + d$, a reconstruction term for timesteps $0 : T$. We can maximize this using the same techniques as maximizing Equation A.3.

Whereas approximating the ELBO in Equation A.4 can be implemented by rolling out OP3 to predict the next observation via teacher forcing (Williams and Zipser, 1989), approximating the posterior predictive distribution in Equation A.4 can be implemented by rolling out the dynamics model d steps beyond the last observation and using the observation model to predict the future observations.

A.4 Interactive Inference

Algorithms 6 and 7 detail M steps of the interactive inference algorithm at timestep 0 and $t \in [1, T]$ respectively. Algorithm 6 is equivalent to the IODINE algorithm described in (Greff et al., 2019). Recalling that $\lambda_{1:K}$ are the parameters for the distribution of the random variables $H_{1:K}$, we consider in this paper the case where this distribution is an isotropic Gaussian (e.g. $\mathcal{N}(\lambda_k)$ where $\lambda_k = (\mu_k, \sigma_k)$), although OP3 need not be restricted to the Gaussian distribution. The *refinement network* f_q produces the parameters for the distribution $q(H_k^{(t)} \mid h_k^{(t-1)}, x^{(t)}, a^{(t)})$. The *dynamics network* $f_{\mathscr{d}}$ produces the parameters for the distribution $\mathscr{d}(H_k^{(t)} \mid h_k^{(t-1)}, h_{[\neq k]}^{(t-1)}, a^{(t)})$. To implement q, we repurpose the dynamics model to transform $h_k^{(t-1)}$ into the initial posterior estimate $\lambda_k^{(0)}$ and then use f_q to iteratively update this parameter estimate. β_k indicates the auxiliary inputs into the refinement network used in (Greff et al., 2019). We mark the major areas where the algorithm at timestep t differs from the algorithm at timestep 0 in blue.

Algorithm 6 Interactive Inference: Timestep 0

```
1: **Input:** observation $x^{(0)}$
2: **Initialize:** parameters $\lambda^{(0,0)}$
3: **for** $i = 0$ **to** $M - 1$ **do**
4:     Sample $h_k^{(0,i)} \sim \mathcal{N}\left(\lambda_k^{(0,i)}\right)$ for each entity $k$
5:     Evaluate $\mathcal{L}^{(0,i)} \approx \log \mathscr{G}\left(x^{(0)} \mid h_{1:K}^{(0,i)}\right) - D_{KL}\left(\mathcal{N}\left(\lambda_{1:K}^{(0,i)}\right) || \mathcal{N}(0, I)\right)$
6:     Calculate $\nabla_{\lambda_k} \mathcal{L}^{(0,i)}$ for each entity $k$
7:     Assemble auxiliary inputs $\beta_k$ for each entity $k$
8:     Update $\lambda_k^{(0,i+1)} \leftarrow f_{\text{refine}}\left(x^{(0)}, \nabla_\lambda \mathcal{L}^{(0,i)}, \lambda^{(0,i)}, \beta_k^{(0,i)}\right)$ for each entity $k$
9: **end for**
10: **Return** $\lambda^{(0,M)}$
```

Algorithm 7 Interactive Inference: Timestep t

```
1: **Input:** observation $x^{(t)}$, previous action $a^{(t-1)}$, previous entity states $h_{1:K}^{(t-1)}$
2: **Predict** $\lambda_k^{(t,0)} \leftarrow f_{\mathscr{d}}\left(h_k^{(t-1)}, h_{[\neq k]}^{(t-1)}, a^{(t-1)}\right)$ for each entity $k$
3: **for** $i = 0$ **to** $M - 1$ **do**
4:     Sample $h_k^{(t,i)} \sim \mathcal{N}\left(\lambda^{(t,i)}\right)$ for each entity $k$
5:     Evaluate $\mathscr{L}^{(t,i)} \approx \log \mathscr{G}\left(x^{(t)} \mid h_{1:K}^{(t)}\right) - D_{KL}\left(\mathcal{N}\left(\lambda_{1:K}^{(t,i)}\right) || \mathcal{N}\left(\lambda_{1:K}^{(t,0)}\right)\right)$
6:     Calculate $\nabla_{\lambda_k} \mathscr{L}^{(t,i)}$ for each entity $k$
7:     Assemble auxiliary inputs $\beta_k$ for each entity $k$
8:     Update $\lambda_k^{(t,i+1)} \leftarrow f_q\left(x^{(t)}, \nabla_{\lambda_k} \mathscr{L}^{(t,i)}, \lambda_k^{(t,i)}, \beta_k^{(t,i)}\right)$ for each entity $k$
9: **end for**
10: **Return** $\lambda^{(t,M)}$
```

Training: We can train the entire OP3 system end-to-end by backpropagating through the entire inference procedure, using the ELBO at every timestep as a training signal for the parameters of $\mathcal{G}$, $\mathcal{D}$, $\mathcal{Q}$ in a similar manner as (van Steenkiste et al., 2018). However, the interactive inference algorithm can also be naturally be adapted to predict rollouts by using the dynamics model to propagate the $\lambda_{1:K}$ for multiple steps, rather than just the one step for predicting $\lambda_{1:K}^{(t,0)}$ in line 2 of Algorithm 7. To train OP3 to rollout the dynamics model for longer timescales, we use a curriculum that increases the prediction horizon throughout training.

A.5 Cost Function

Let $\hat{I}(H_k) := m(H_k) \cdot g\left(X \mid H_k\right)$ be a *masked* sub-image (see Appdx: A.1). We decompose the cost of a particular configuration of objects into a distance function between entity states, $c(H_a, H_b)$. For the first environment with single-step planning we use L_2 distance of the corresponding masked subimages: $c(H_a, H_b) = L_2(\hat{I}(H_a), \hat{I}(H_b))$. For the second environment with multi-step planning we a different distance function since the previous one may care more about if a shape matches than if the color matches. We instead use a form of intersection over union but that counts intersection if the mask aligns and pixel color values are close $c(H_a, H_b) = 1 - \frac{\sum_{i,j} m_{ij}(H_a)>0.01 \text{ and } m_{ij}(H_b)>0.01 \text{ and } L_2(g(H_a)_{(ij)}, g(H_b)_{(ij)})<0.1}{\sum_{i,j} m_{ij}(H_a)>0.01 \text{ or } m_{ij}(H_b)>0.01}$. We found this version to work better since it will not give low cost to moving a wrong color block to the position of a different color goal block.

A.6 Architecture and Hyperparameter Details

We use similar model architectures as in (Greff et al., 2019) and so have rewritten some details from their appendix here. Differences include the dynamics model, inclusion of actions, and training procedure over sequences of data. Like (Hafner et al., 2018), we define our latent distribution of size R to be divided into a deterministic component of size R_d and stochastic component of size R_s. We found that splitting the latent state into a deterministic and stochastic component (as opposed to having a fully stocahstic representation) was helpful for convergence. We parameterize the distribution of each H_k as a diagonal Gaussian, so the output of the refinement and dynamics networks are the parameteres of a diagonal Gaussian. We parameterize the output of the observation model also as a diagonal Gaussian with means μ and global scale $\sigma = 0.1$. The observation network outputs the μ and mask m_k.

Training: All models are trained with the ADAM optimizer (Kingma and Ba, 2014) with default parameters and a learning rate of 0.0003. We use gradient clipping as in (Pascanu et al., 2012) where if the norm of global gradient exceeds 5.0 then the gradient is scaled down to that norm.

Inputs: For all models, we use the following inputs to the refinement network, where LN means Layernorm and SG means stop gradients. The following image-sized inputs are concatenated and fed to the corresponding convolutional network:

Description	Formula	LN	SG	Ch.
image	X			3
means	μ			3
mask	m_k			1
mask-logits	$\hat{m}_k$			1
mask posterior	$p(m_k \mid X, \mu)$			1
gradient of means	$\nabla_{\mu_k}\mathcal{L}$	✓	✓	3
gradient of mask	$\nabla_{m_k}\mathcal{L}$	✓	✓	1
pixelwise likelihood	$p(X \mid H)$	✓	✓	1
leave-one-out likelih.	$p(X \mid H_{i \neq k})$	✓	✓	1
coordinate channels				2
			total:	17

Observation and Refinement Networks

The posterior parameters $\lambda_{1:K}$ and their gradients are flat vectors, and we concatenate them with the output of the convolutional part of the refinement network and use the result as input to the refinement LSTM:

Description	Formula	LN	SG
gradient of posterior	$\nabla_{\lambda_k}\mathcal{L}$	✓	✓
posterior	λ_k		

All models use the ELU activation function and the convolutional layers use a stride equal to 1 and padding equal to 2 unless otherwise noted. For the table below $R_s = 64$ and $R = 128$.

Observation Model Decoder

Type	Size/*Ch.*	Act. Func.	Comment
Input: H_i	R		
Broadcast	R+*2*		+ coordinates
Conv 5 × 5	*32*	ELU	
Conv 5 × 5	*32*	ELU	
Conv 5 × 5	*32*	ELU	
Conv 5 × 5	*32*	ELU	
Conv 5 × 5	*4*	Linear	RGB + Mask

Refinement Network

Type	Size/*Ch.*	Act. Func.	Comment
MLP	128	Linear	
LSTM	128	Tanh	
Concat $[\lambda_i, \nabla_{\lambda_i}]$	$2R_s$		
MLP	128	ELU	
Avg. Pool	R_s		
Conv 3 × 3	R_s	ELU	
Conv 3 × 3	*32*	ELU	
Conv 3 × 3	*32*	ELU	
Inputs	*17*		

Dynamics Model

The dynamics model $\mathcal{D}$ models how each entity H_k is affected by action A and the other entity $H_{[\neq k]}$. It applies the same function $d(H'_k \mid H_k, H_{[\neq k]}, A)$ to each state, composed of several functions illustrated and described in Fig. 3.4:

$$\tilde{H}_k = d_o(H_k) \qquad \tilde{A} = d_a(A^t) \qquad \tilde{H}_k^{\text{act}} = d_{ao}(\tilde{H}_k\tilde{A})$$

$$H_k^{\text{interact}} = \sum_{i\neq k}^{K} d_{oo}(\tilde{H_i}^{\text{act}}, \tilde{H_k}^{\text{act}}) \qquad H_k^{'} = d_{\text{comb}}(\tilde{H}_k^{\text{act}}, H_k^{\text{interact}}),$$

where for a given entity k, $d_{ao}(\tilde{H}_k\tilde{A}) := d_{\text{act-eff}}(\tilde{H}_k, \tilde{A}) \cdot d_{\text{act-att}}(\tilde{H}_k, \tilde{A})$ computes how ($d_{\text{act-eff}}$) and to what degree ($d_{\text{act-att}}$) an action affects the entity and $d_{oo}(\tilde{H_i}^{\text{act}}, \tilde{H_k}^{\text{act}}) := d_{\text{obj-eff}}(\tilde{H}_i^{\text{act}}, \tilde{H}_k^{\text{act}}) \cdot d_{\text{obj-att}}(\tilde{H}_i^{\text{act}}, \tilde{H}_k^{\text{act}})$ computes how ($f_{\text{obj-eff}}$) and to what degree ($d_{\text{obj-att}}$) other entities affect that entity. $d_{\text{obj-eff}}$ and $d_{\text{obj-att}}$ are shared across all entity pairs. The other functions are shared across all entities. The dynamics network takes in a sampled state and outputs the parameters of the posterior distribution. Similar to (Hafner et al., 2018) the output H_k' is then split into deterministic and stochastic components each of size 64 with separate networks f_{det} and f_{sto}. All functions are parametrized by single layer MLPs.

Dynamics Network

Function	Output	Act. Func.	MLP Size
$d_o(H_k)$	$\tilde{H}_k$	ELU	128
$d_a(A)$	$\tilde{A}$	ELU	32
$d_{\text{act-eff}}(\tilde{H}_k, \tilde{A})$		ELU	128
$d_{\text{act-att}}(\tilde{H}_k, \tilde{A})$		Sigmoid	128
$d_{\text{obj-eff}}(\tilde{H}_i^{\text{act}}, \tilde{H}_j^{\text{act}})$		ELU	256
$d_{\text{obj-att}}(\tilde{H}_i^{\text{act}}, \tilde{H}_k^{\text{act}})$		Sigmoid	256
$d_{\text{comb}}(\tilde{H}_i^{\text{act}}, \tilde{H}_k^{\text{interact}})$	H_k'	ELU	256
$f_{\text{det}}(H_k')$	$H_{\text{k,det}}'$		128
$f_{\text{sto}}(H_k')$	$H_{\text{k,sto}}'$		128

This architectural choice for the dynamics model is an action-conditioned modification of the interaction function used in Relational Neural Expectation Maximization (RNEM) (van Steenkiste et al., 2018), which is a latent-space attention-based modification of the Neural Physics Engine (NPE) (Chang et al., 2016), which is one of a broader class of architectures known as graph networks (Battaglia et al., 2018).

A.7 Experiment Details

Single-Step Block-Stacking

The training dataset has 60,000 trajectories each containing before and after images of size 64x64 from Janner et al. (2018). Before images are constructed with actions which consist of choosing a shape (cube, rectangle, pyramid), color, and an (x, y, z) position and orientation for the block to be dropped. At each time step, a block is dropped and the simulation runs until the block settles into a stable position. The model takes in an image containing the block to be dropped and must predict the steady-state effect. Models were trained on scenes with 1 to 5 blocks with $K = 7$ entity variables. The cross entorpy method (CEM) begins from a uniform distribution on the first iteration, uses a population size of 1000 samples per iteration, and uses 10% of the best samples to fit a Gaussian distribution for each successive iteration.

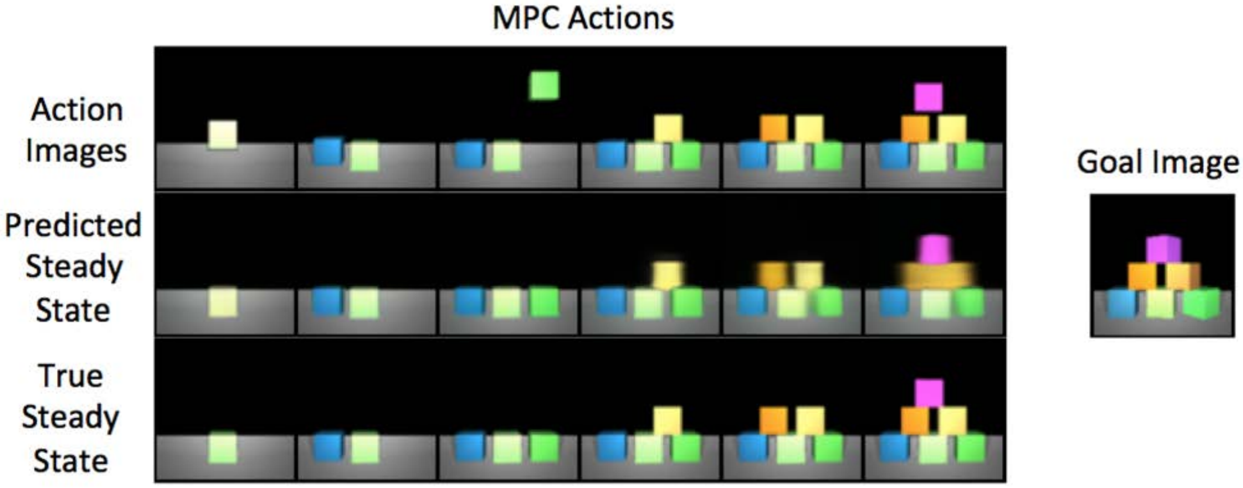


Figure A.1: Qualitative results on building a structure from the dataset in (Janner et al., 2018). The input is an "action image," which depicts how an action intervenes on the state by raising a block in the air. OP3 is trained to predict the steady-state outcome of dropping the block. We see how OP3 is able to accurately and consistently predict the steady state effect, successively capturing the effect of inertial dynamics (gravity) and interactions with other objects.

Multi-Step Block-Stacking

The training dataset has 10,000 trajectories each from a separate environment with two different colored blocks. Each trajectory contains five frames (64x64) of randomly picking and placing blocks. We bias the dataset such that 30% of actions will pick up a block and place it somewhere randomly, 40% of actions will pick up a block and place it on top of a another random block, and 30% of actions contain random pick and place locations. Models were trained with K = 4 slots. We optimize actions using CEM but we optimize over multiple consecutive actions into the future executing the sequence with lowest cost. For a goal with n blocks we plan n steps into the future, executing n actions. We repeat this procedure $2n$ times or until the structure is complete. Accuracy is computed as $\frac{\text{\# blocks in correct position}}{\text{\# goal blocks}}$, where a correct position is based on a threshold of the distance error.

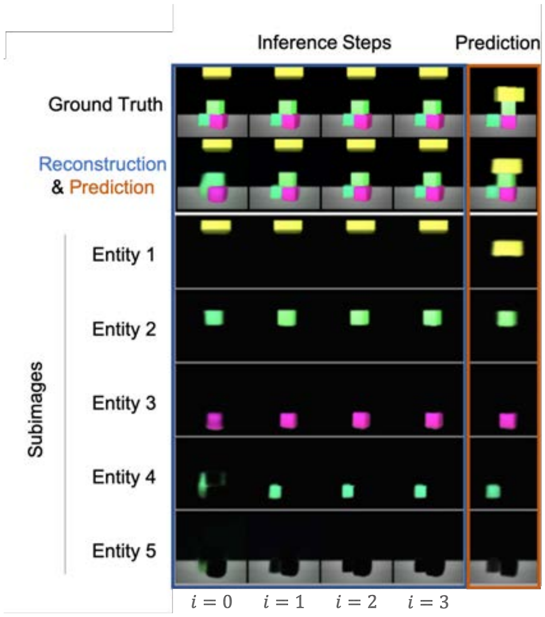


Figure A.2: We show a demonstration of a rollout for the dataset from (Janner et al., 2018). The first four columns show inference iterations (refinement steps) on the single input image, while the last column shows the predicted results using the dynamics module on the learnt hidden states. The bottom 5 rows show the subimages of each entity at each iteration, demonstrating how the model is able to capture individual objects, and the dynamics afterwards. Notice that OP3 only predicts a change in the yellow block while leaving the other latents unaffected. This is a desriable property for dynamics models that operate on scenes with multiple objects.

For MPC we use two difference action spaces:

Coordinate Pick Place: The normal action space involves choosing a pick (x,y) and place (x,y) location.

Entity Pick Place: A concern with the normal action space is that successful pick locations are sparse (2%) given the current block size. Therefore, the probability of picking n blocks consecutively becomes 0.02^n which becomes improbable very fast if we just sample pick locations uniformly. We address this by using the pointers to the entity variables to create an action space that involves directly choosing one of the latent entities to move and then a place (x, y) location. This allows us to easily pick blocks consecutively if we can successfully map a latent `entity_id` of a block to a corresponding successful pick location. In order to determine the pick (x, y) from an `entity_id` k, we sample coordinates uniformly over the pick (x, y) space and then average these coordinates weighted by their attention

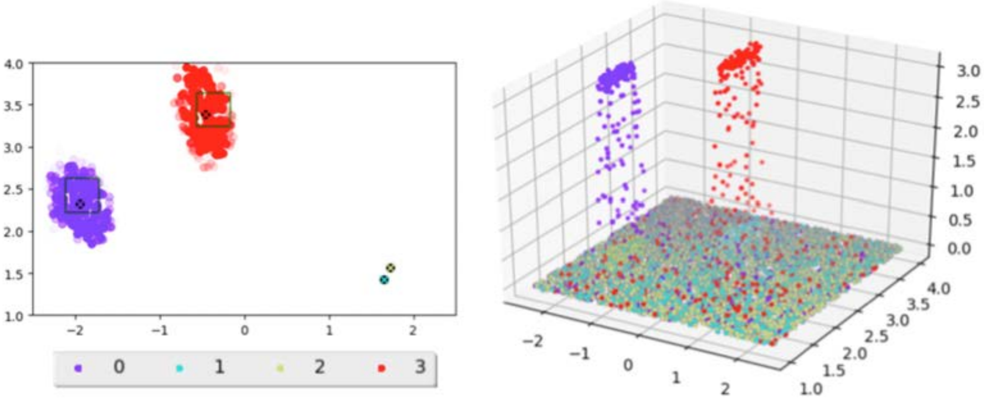


Figure A.3: Two-dimensional (left) and three-dimensional (right) visualization of attention values where colors correspond to different latents. The blocks are shown as the green squares in the 2D visualizatio; picking anywhere within the square automatically picks the block up. The black dots with color crosses denote the computed `pick_xy` for a given h_k. We see that although the individual values are noisy, the means provide good estimates of valid pick locations. In the right plot we see that attention values for all objects are mostly 0, except in the locations corresponding to the objects (purple and red).

coefficient on that latent:

$$\texttt{pick\_xy}|h_k = \frac{\sum_{x',y'} p(h_k|x,y) * \texttt{pick\_x'y'}}{\sum_{x',y'} p(h_k|x',y')}$$

where $p(h_k|x,y)$ are given by the attention coefficients produced by the dynamics model given h_k and the pick location (x,y) and x', y' are sampled from a uniform distribution. The attention coefficient of H_k is computed as $\sum_{i \neq k}^{K} d_{\text{obj-att}}(\tilde{H}_i^{\text{act}}, \tilde{H}_k^{\text{act}})$ (see Appdx. A.6)

A.8 Ablations

We perform ablations on the block stacking task from Janner et al. (2018) examining components of our model. Table A.1 shows the effect of non-symmetrical models or cost functions. The “Unfactorized Model” and “No Weight Sharing” follow (c) and (d) from Figure 3.1 and are unable to sufficiently generalize. “Unfactorized Cost” refers to simply taking the mean-squared error of the compositie prediction image and the goal image, rather than decomposing the cost per entity masked subimage. We see that with the same OP3 model trained on the same data, not using an entity-centric factorization of the cost significantly underperforms a cost function that does decompose the cost per entity (c.f. Table 3.1).

No Weight Sharing	Unfactorized Model	Unfactorized Cost
0 %	0 %	5%

Table A.1: Accuracy of ablations. The no weight sharing model did not converge during training.

A.9 Interpretability

We do not explicitly explore interpretability in this work, but we see that an entity-factorized model readily lends itself to be interpretable by construction. The ability to decompose a scene into specific latents, view latents invididually, and explicitly see how these latents interact with each other could lead to significantly more interpretable models than current unfactorized models. Our use of attention values to determine the pick locations of blocks scratches the surface of this potential. Additionally, the ability to construct cost functions based off individual latents allows for more interpretable and customizable cost functions.

Appendix B

SeCTAR Details

B.1 Experimental Details

For all experiments, we parameterize $\pi_{\theta_{PD}}$ and π_e as a three-layer fully connected neural networks with 400, 300, 200 hidden units and ReLU activations. The policies output either categorical or Gaussian distributions. The encoder is a two-layer bidirectional-LSTM with 300 hidden units, and we mean-pool over LSTM outputs over time before applying a linear transform to produce parameters of a Gaussian distribution. We use an 8-dimensional diagonal Gaussian distribution for z. The state decoder is a single-layer LSTM with 256 hidden units that conditions on the initial state and latent z, to output a Gaussian distribution over trajectories. We use trajectories of length $T = 19$, and plan over $K = 2048$ random latent sequences. We use horizons $H = 380$, $H_{MPC} = 5$, $H_e = 5$ for the 2D navigation task, $H = 950$, $H_{MPC} = 20$, $H_e = 10$ for the wheeled locomotion task, and $H = 950$, $H_{MPC} = 10$, $H_e = 10$ for the object manipulation task. These values were chosen empirically with a hyperparameer sweep.

B.2 Baseline Details

TRPO / VIME We used the rllab TRPO implementation, OpenAI VIME implementation with a batch size of 100 * task horizon and step size of 0.01.

MPC We use a learning rate of 0.001 and batch size of 512. The MPC policy simulates 2048 paths each time it is asked for an action. We verified correctness on half-cheetah.

Option Critic We use a version of Option Critic that uses PPO instead of DQN. We swept over number of options, reward multiplier, and entropy bonuses. We verified correctness on cartpole, hopper, and cheetah.

Feudal / A3C The Feudal and A3C implementations are based on chainerRL. We swept over the parameters β, t_{max}, and gradient clipping.

Appendix C

Evolving RL Details

C.1 Search Language Details

Inputs and outputs to nodes in the computational graph have data types which include state $\mathbb{S}$, action $\mathbb{Z}$, float $\mathbb{R}$, list $List[\mathbb{X}]$, probability $\mathbb{P}$, vector $\mathbb{V}$. The symbol $\mathbb{X}$ indicates it can be of $\mathbb{S}, \mathbb{R}$, or $\mathbb{V}$. We assume that vectors are of fixed length 32 and actions are integers. Operations will broadcast so that for example adding a float variable to a state variable will result in the float being added to each element of the state. This typing allows the learned program to be domain agnostic. The full list of operators is listed below.

Operation	Input Types	Output Type
Add	$\mathbb{X}$, $\mathbb{X}$	$\mathbb{X}$
Subtract	$\mathbb{X}$, $\mathbb{X}$	$\mathbb{X}$
Max	$\mathbb{X}$, $\mathbb{X}$	$\mathbb{X}$
Min	$\mathbb{X}$, $\mathbb{X}$	$\mathbb{X}$
DotProduct	$\mathbb{X}$, $\mathbb{X}$	$\mathbb{R}$
Div	$\mathbb{X}$, $\mathbb{X}$	$\mathbb{X}$
L2Distance	$\mathbb{X}$, $\mathbb{X}$	$\mathbb{R}$
MaxList	$List[\mathbb{R}]$	$\mathbb{R}$
MinList	$List[\mathbb{R}]$	$\mathbb{R}$
ArgMaxList	$List[\mathbb{R}]$	$\mathbb{Z}$
SelectList	$List[\mathbb{X}]$, $\mathbb{Z}$	$\mathbb{X}$
MeanList	$List[\mathbb{X}]$	$\mathbb{X}$
VarianceList	$List[\mathbb{X}]$	$\mathbb{X}$
Log	$\mathbb{X}$	$\mathbb{X}$
Exp	$\mathbb{X}$	$\mathbb{X}$
Abs	$\mathbb{X}$	$\mathbb{X}$
(C)NN:$\mathbb{S} \rightarrow List[\mathbb{R}]$	$\mathbb{S}$	$List[\mathbb{R}]$
(C)NN:$\mathbb{S} \rightarrow \mathbb{R}$	$\mathbb{S}$	$\mathbb{R}$
(C)NN:$\mathbb{S} \rightarrow \mathbb{V}$	$\mathbb{V}$	$\mathbb{V}$
Softmax	$List[\mathbb{R}]$	$\mathbb{P}$
KLDiv	$\mathbb{P}$, $\mathbb{P}$	$\mathbb{R}$
Entropy	$\mathbb{P}$	$\mathbb{R}$
Constant		1, 0.5, 0.2, 0.1, 0.01
MultiplyTenth	$\mathbb{X}$	$\mathbb{X}$
Normal(0, 1)		$\mathbb{R}$
Uniform(0, 1)		$\mathbb{R}$

C.2 Training Details

We describe the training details and hyperparameters used. For all environments we use the Adam optimzier with a learning rate of 0.0001.

Common RL training details. All neural networks are MLPs of size (256, 256) with ReLU activations. For optimizing the Q-function parameters we use the Adam optimizer with a learning rate of 0.0001. Target update period is 100. These settings are used for all training and test environments.

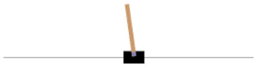
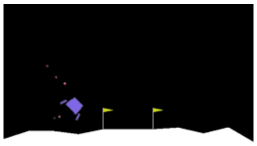
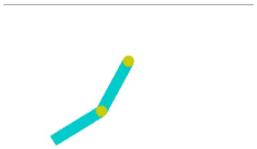
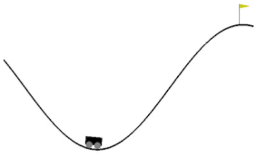
Classical control environments. The value of ϵ is decayed linearly from 1 to 0.05 over 1000 steps. CartPole, Acrobat, and MountainCar are trained for 400 episodes and LunarLander is trained for 1000 episodes.

MiniGrid environments. The value of ϵ is decayed linearly from 1 to 0.05 over 10^5 steps. During meta-training, MiniGrid environments are trained for $5 * 10^5$ steps.

Atari environments. We use the same neural network architecture as in Mnih et al. (2013). Target update period is $1,000$. The value of ϵ is decayed linearly from 1 to 0.1 over 10^6 steps. For evaluation, we use the no-op start condition as in Mnih et al. (2013) where the agent will output the no-op action for x steps where x is a random integer drawn between $[1, 30]$. The evaluation policy uses an ϵ of 0.001 and is evaluated every 10^6 steps for 100 episodes. The best training snapshot is reported.

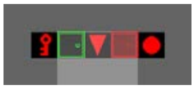
C.3 Environment Details

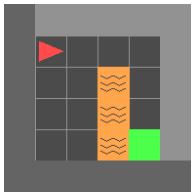
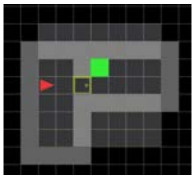
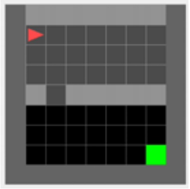
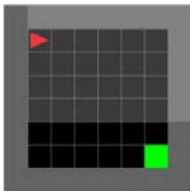
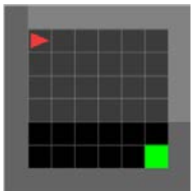
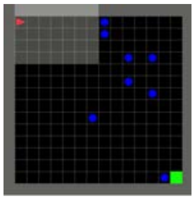
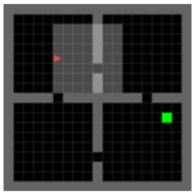
We describe the classical control environments below. CartPole and LunarLander are dense reward while Acrobat and MountainCar are sparse reward.

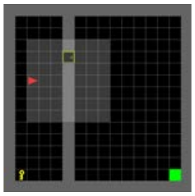
	Task ID	Description
	CartPole-v0	The agent must balance a pole on top of a cart by applying a force of +1 or −1 to the cart. A reward of +1 is provided for each timestep the pole remains upright.
	LunarLander-v2	The agent controls a lander by firing one of four thrusters and must land it on the landing pad.
	Acrobat-v1	The goal is to swing a 2-link system upright to a given height by applying 1, 0, or -1 torque on the join between the two links.
	MountainCar-v0	The goal is to drive up the mountain on the right by first driving back and forth to build up momentum.

We describe the MiniGrid environments below. The input to the agent is a fully observed grid which is encoded as an NxNx3 size array where N is the grid size. The 1st channel contains the index of the object type at that location (out of 11 possible objects), the 2nd channel contains the color of the object (out of 6 possible colors), and the 3rd channel contains the orientation of the agent out of 4 cardinal directions. This encoding is then flattened and fed into an MLP. There are 7 possible actions (turn left, turn right, forward, pickup, drop, toggle, done).

Unless stated otherwise, all tasks are sparse reward tasks with a reward of 1 for completing the task. Max steps is set to 100. A size such as 5x5 in the environment name refers to a grid size with width and height of 5 cells.

	Task ID	Description
	KeyCorridorS3R1-v0	The agent has to find a key hidden in one room and then use it to pickup an object behind a locked door in another room. This tests sequential subgoal completion.

	LavaGapS5-v0	The agent has to reach the green goal square without touching the lava which will terminate the episode with zero reward. This tests safety and safe exploration.
	MultiRoom-N2-S4-v0	The agent must open a door to get to the green goal square in the next room.
	SimpleCrossingS9N1-v0	The agent has to reach the green goal square on the other corner of the room and navigate around walls.
	Empty-v0	The agent has to reach the green goal square in an empty room.
	EmptyRandom-v0	The agent has to reach the green goal square in an empty room but is initialized to a random location.
	Dynamic-Obstacles-v0	The agent has to reach the green goal square without colliding with any blue obstacles which move around randomly. If the agent collides with an obstacle it receives a reward of -1 and the episode terminates.
	FourRooms-v0	The agent must navigate in a maze composed of four rooms. Both the agent and goal square are randomly placed in any of the four rooms.

	DoorKey-v0	The agent must pick up a key to unlock a door to enter another room and get to the green goal square.

C.4 Graph Distribution Analysis

We look at the distribution of top performing graphs and find similarities in their structure. This is summarized in Table C.2 where we describe the equations of learned algorithms for differing ranks (if sorted by score). The best performing algorithms from the experiment which learned DQNReg are all variants of adding $Q(s_t, a_t)$ to the standard TD loss in some form, $\delta^2 + k * Q(s_t, a_t)$. We think this kind of loss could use further investigation and that while we did not tune the value of k, this could also be tuned per environment. In Figure 5.3b, we show the distribution of scores for all non-duplicate programs that have been evaluated. We provide a full list of top performing algorithms from a few of our experiments at `https://github.com/jcoreyes/evolvingrl`.

Raw Equation	Simplified Equation	Score	Rank
$\delta^2 + 0.1 * Q(s_t, a_t) + r_t - (\gamma * Q_{targ} - 0.1 * Q(s_t, a_t))$	$\delta^2 + 0.2 * Q(s_t, a_t)$	3.905	2
$\delta^2 + 0.1 * Q(s_t, a_t) - \gamma + Q_{targ}$	$\delta^2 + 0.1 * Q(s_t, a_t)$	3.904	3
$\delta^2 - (\gamma * Q_{targ} - 0.1 * Q(s_t, a_t))$	$\delta^2 + 0.1 * Q(s_t, a_t)$	3.903	4
$\delta^2 + Q_{targ} + 0.1 * Q(s_t, a_t) - \gamma$	$\delta^2 + 0.1 * Q(s_t, a_t)$	3.902	5
$\delta^2 - (0.1 * Q(s_t, a_t) - Y_t)^2$	$\delta^2 - (0.1 * Q(s_t, a_t) - Y_t)^2$	3.898	6
$\delta^2 + ((r_t + \gamma * Q_{targ} + Q(s_t, a_t)) * (\gamma - \max(\gamma, 0.1 * Q(s_t, a_t))$ $-\gamma * Q_{targ} - 0.1 * Q(s_t, a_t))$	NA	3.846	11146
$\delta^2 + (\delta^2 + 0.1 * Q(s_t, a_t))^2$	NA	3.65	12146
$\delta^2 + Q(s_t, a_t)$	$\delta^2 + Q(s_t, a_t)$	2.8	12446
δ^2	δ^2	2.28	13246

Table C.2: Other programs learned in learning DQNReg which is rank 1 with score 3.907. Rank is if scores are sorted in decreasing order. Score is the sum of normalized RL training performance across four environments. The simplified equations contains only the relevant parts for minimizing the equation output. Q_{targ} refers to $\max_a Q_{targ}(s_{t+1}, a)$.

C.5 Repeatability of Meta Training

In Figure C.1, we plot the meta-training performance for bootstrapping from DQN with four training environments (CartPole, KeyCorridorS3R1, Dynamic-Obstacles-6x6, DoorKey-5x5) over ten trials. Four out of the ten trials reach the max training performance. Two out of 10 of these trials learns the same algorithm DQNReg while the other top two trials find other less interpretable algorithms.

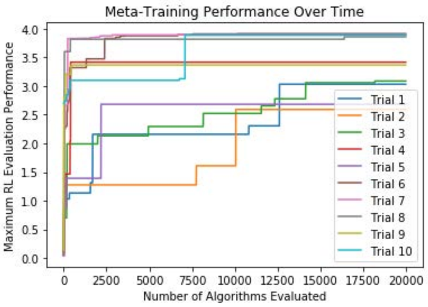


Figure C.1: Meta-training performance for boot-strapping on 4 training environments for 10 random seeds.

Appendix D

Believer Details

D.1 Experimental Details

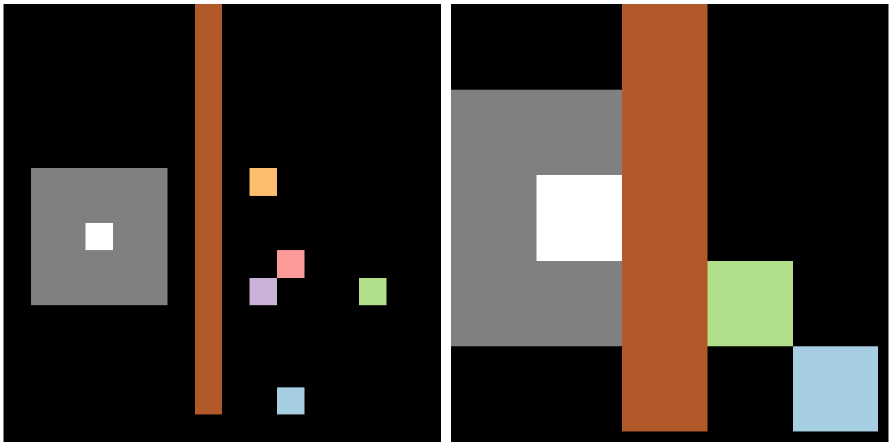
Computational resources. Most experimental results were computed on a Linux Destkop with 32 GiB of RAM, equipped with an AMD Ryzen 7 3800X 8-core CPU and an RTX 2080 Ti GPU.

True-state entropy metric. We approximated $H(d^\pi(s_d))$ by computing an estimated $d^\pi(s_d)$ during the episode. We report the entropy at the final step of the episode, which is when the estimate of $d^\pi(s_d)$ is most precise. Recall that s_d represents the positions of the dynamic objects in the environment. In the TwoRoom environment, $d^\pi(s_d)$ is computed by recording counts. In the OneRoomCapture3D environment, s_d is continuous, and $d^\pi(s_d)$ is computed by fitting a diagonal Gaussian.

Environment details.

In this section, we elaborate on the details of the environments described in the main text.

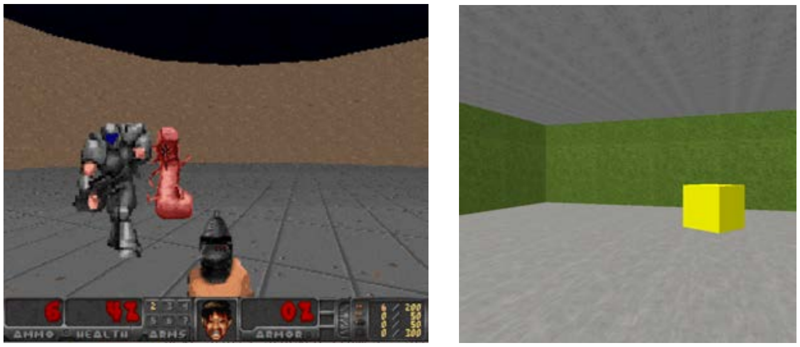
TwoRoom environment details. As previously described, this environment has two rooms: an empty ("dark") room on the left, and a "busy" room on the right, the latter containing moving particles that move around unless the agent "tags" them, which permanently stops their motion, as shown in Fig. D.1. The agent can observe a small area around it, which it receives as an image. In this environment, control corresponds to finding and stopping the particles. The action space is $\mathcal{A} = \{\text{left, right, up, down, tag, no-op}\}$, and the observation space is normalized RGB-images: $\Omega = [0, 1]^{3\times30\times30}$. An agent that has significant control over this environment should tag particles to reduce the uncertainty over future states. To evaluate policies, we use the average fraction of total particles locked, the average fraction of particles visible, and the discrete true-state visitation entropy of the positions of the particles, $H(d^\pi(s_d))$. We employed two versions of this environment. In the large environment, the



(a) TwoRoom Large Environment (b) TwoRoom Environment

Figure D.1: TwoRoom Environments. In the large environment, the agent observes a 5x5 area around it as an image, and the busy room contains 5 particles. In the normal environment, the agent observes a 3x3 around it as an image, and the busy room contains 2 particles. In both settings, the particles are initialized to random positions in the busy room at the beginning of each episode.

agent observes a 5x5 area around it as an image, and the busy room contains 5 particles. In the normal environment, the agent observes a 3x3 around it as an image, and the busy room contains 2 particles. The large environment consists of an area of 15x15 cells, and the normal environment consists of an area of 5x5 cells. In both settings, the particles are initialized to random positions in the busy room at the beginning of each episode, and episodes last for $T = 100$ timesteps. The particles bounce off the walls, but not each other.



(a) Vizdoom DefendTheCenter. (b) OneRoomCapture3D

Figure D.2: Vizdoom Defend The Center and OneRoomCapture3D

VizDoom DefendTheCenter. The VizDoom DefendTheCenter environment shown in Fig. D.2 is a circular arena in which an agent, equipped with a partial field-of-view of the arena and a weapon, can rotate and shoot encroaching monsters (Kempka et al., 2016). The action space is $\mathcal{A} = \{\text{turn left, turn right, shoot}\}$, and the observation space is normalized RGB-images: $\Omega = [0, 1]^{3\times64\times64}$. In this environment, significant control corresponds to reducing the number of monsters by finding and shooting them. We use the average original environment return (for which *no method* has access to during training) the average number of killed monsters at the end of an episode, and the average number of visible monsters to measure the agent's control. Episodes last for $T = 500$ timesteps.

OneRoomCapture3D The MiniWorld framework is a customizable 3D environment simulator in which an agent perceives the world through a perspective camera (Chevalier-Boisvert, 2018). We used this framework to build the environment in Fig. D.2 which an agent and a bouncing box both inhabit a large room; the agent can lock the box to stop it from moving if it is nearby, as well as constrain the motion of the box by standing nearby it. In this environment, significant control corresponds to finding the box and either trapping it near a wall, or tagging it. When the box is tagged, its color changes from yellow to purple. The action space is $\mathcal{A} = \{\text{turn left } 20°, \text{turn right } 20°, \text{move forward, move backward, tag}\}$, and observation space is normalized RGB-images: $\Omega = [0, 1]^{3\times64\times64}$. We use the average fraction of time the box is captured, the average time the box is visible, and continuous (Gaussian-estimated) true-state visitation entropy of the box's position $H(d^{\pi}(s_d))$ to measure the agent's ability to reduce entropy of the environment. Episodes last for $T = 150$ timesteps.

D.2 Implementation Details

Hyperparameters. In table D.1, we provide values of the hyperparameters used in algorithm 5 and the LSSM architecture described in table D.2.

Architectural details. We provide detailed information on the architectural implementation of the learners in table D.2. The value function used for generalized advantage estimation in PPO uses the same architecture as the policy with decoupled weights, except with a final output size of 1 (scalar).

Optimization. We use RAdam to optimize the policies and LSSM (Liu et al., 2019). Following Hafner et al. (2020), we use straight-through gradient estimation of samples of the Categorical distributions.

Hyperparameter	Value	Meaning
		algorithm 5 hyperparameters
K	7	Ensemble size
B	32	Minibatch size of LSSM
L	$0.05\lvert\mathcal{D}\rvert/B$	Number of minibatches to step the model
M	$1e7/2NT$	Maximum number of total rounds
N	20	Number of episodes to collect per policy per round
T	{100,150,500} (varies)	Episode length in the environment
		LSSM hyperparameters
H	50	LSSM model training horizon
P	5	Number of latent particles
K_1	16	Number of component categoricals in latent distributions
K_2	16	Number of categories in each component categorical
		Optimization hyperparameters
α_0	$0.5 \cdot 10^{-4}$	LSSM learning rate with Adam optimizer
α_1	$1.0 \cdot 10^{-4}$	PPO learning rate with Adam optimizer
ϵ_{PPO}	0.2	PPO advantage clipping
γ_0	0.99	PPO discount factor
γ_1	0.90	PPO GAE discount factor
β	1.0	KL loss scaling factor (implicit in eq. (6.1))
b_0	True	Whether truncated BPTT (Sutskever, 2013) is used to train the model
j_1, j_2	50	Truncated BPTT horizons

Table D.1: Hyperparameters of algorithm 5, models, and optimization.

.

D.3 OneRoomCapture3d visualization.

In fig. D.3, we analyze the behavior of our intrinsic reward signals over several different sub-episodes in the OneRoomCapture3D environment.

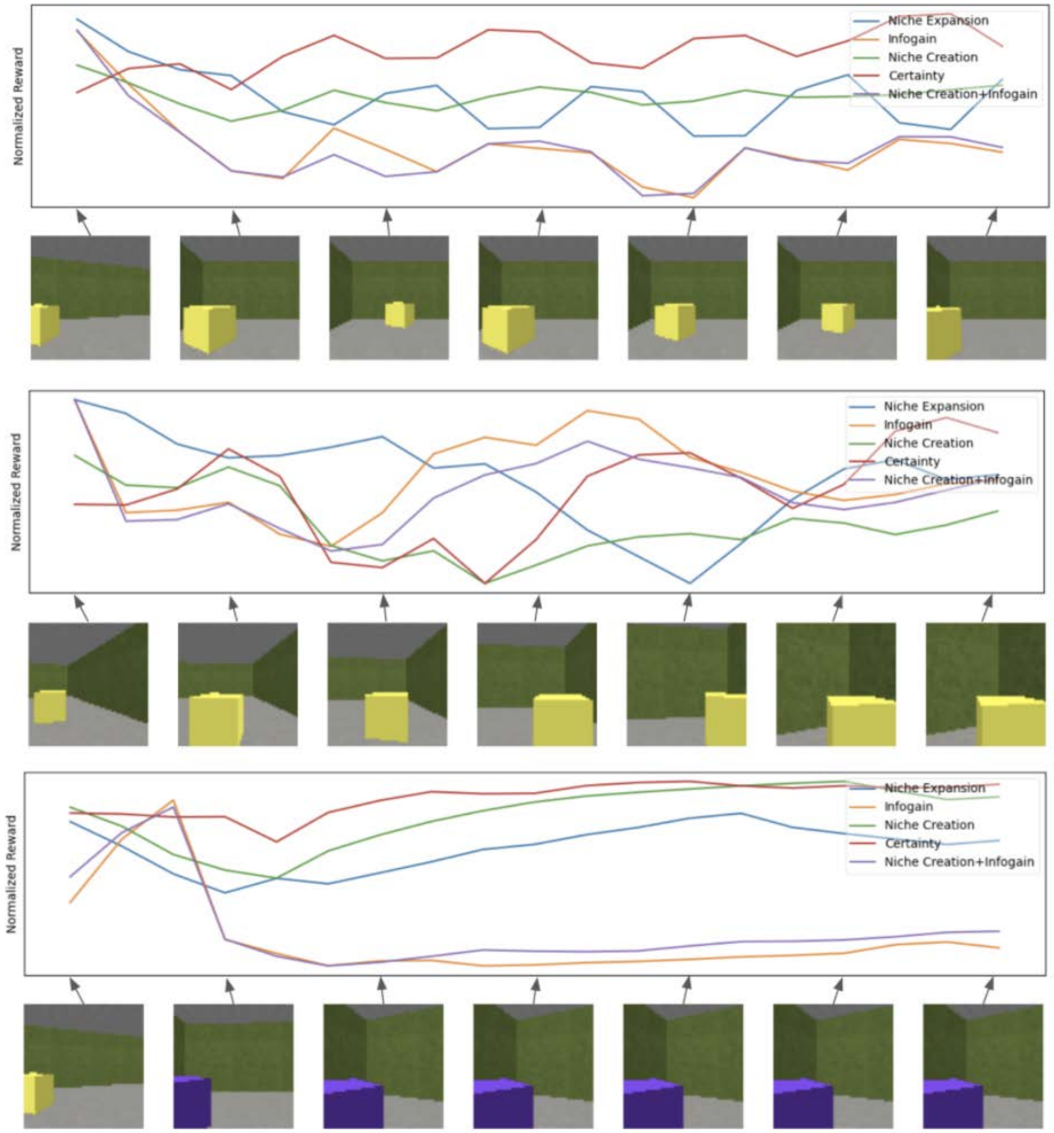


Figure D.3: Niche Expansion, Infogain, Niche Creation, Certainty, and Niche Creation+Infogain rewards are plotted for the first 20 steps of select episodes. Rewards are normalized to [0, 1] for each reward across the figures. *Top:* When the agent turns to look at the box without taking actions to capture it, all rewards other than Certainty are relatively low throughout the episode. *Middle:* When the agent moves towards the box to trap it against a wall, Niche Creation and Niche Expansion decrease until the box is trapped, and then they increase; the resulting stable configuration eventually outweighs the preparations needed to trap the box if the episode length is sufficiently long. Infogain and Certainty increase as the box is in view and able to move. *Bottom:* Freezing the box results in low Infogain throughout the episode, however it is highly rewarded by the other rewards.

Layer	Input [Dimensionality]	Output [Dimensionality]
Prior $p_\theta(\mathbf{z}_{t+1} \mid \mathbf{z}_t, \mathbf{a}_t, g_t) = \prod_{\kappa_1=1}^{K_1} \prod_{\kappa_2=1}^{K_2} \mathbf{v}_{\text{prior},\kappa_1,\kappa_2}^{\mathbf{z}_{t+1,\kappa_1,\kappa_2}} = \text{MultiCat}(\cdot; \mathbf{v}_{\text{prior}})$		
	//Embed action with affine layer	
1	$\mathbf{a}_t$ $[A]$	L_a=$\texttt{Linear}(\mathbf{a}_t)$, $[K_1 \cdot K_2]$
	//Combine action embedding and latent state	
2	$\mathbf{z}_t$ $[K_1, K_2]$; L_a $[K_1 \cdot K_2]$	L_{az}=$\texttt{Concat(Flatten(}\mathbf{z}_t\texttt{), }L_a\texttt{)}$, $[2K_1 \cdot K_2]$
	//RNN transformation of action embedding and latent state	
3	L_{az}, $[2K_1 \cdot K_2]$; g_t, $[K_1 \cdot K_2]$	g_{t+1}=$\texttt{GRU}(L_{az}, g_t)$, $[K_1 \cdot K_2]$
	//Logits of independent Categorical prior (‘‘MultiCat’’)	
4	g_{t+1}, $[K_1 \cdot K_2]$	$\mathbf{l}_{\text{prior}}$=$\texttt{Linear}(g_{t+1})$, $[K_1 \cdot K_2]$
	//Transform logits of all K_1 component dists.	
5	$\mathbf{l}_{\text{prior}}$,$[K_1 \cdot K_2]$	$\mathbf{v}_{\text{prior}}$=$\texttt{softmax}(\mathbf{l}_{\text{prior}}, \text{axis} = K_2)$
Posterior $q_\phi(\mathbf{z}_{t+1} \mid \mathbf{o}_{\leq t+1}, \mathbf{a}_{\leq t}, \mathbf{z}_t, g_t) = \text{MultiCat}(\cdot; \mathbf{v}_{\text{posterior}})$		
	//Apply CNN to observation.	
1	$\mathbf{o}_t$, $[3, 64, 64]$	L_{o0}=$\texttt{ELU(BN(Conv2d(3, 32, 4, 2)))}(\mathbf{o}_t)$
2	L_{o0}, $[32, 31, 31]$	L_{o1}=$\texttt{ELU(BN(Conv2d(32, 64, 4, 2)))}(L_{o0})$
3	L_{o1}, $[64, 14, 14]$	L_{o2}=$\texttt{ELU(BN(Conv2d(64, 128, 4, 2)))}(L_{o1})$
4	L_{o2}, $[128, 6, 6]$	L_{o3}=$\texttt{ELU(BN(Conv2d(128, 256, 4, 2)))}(L_{o2})$
5	L_{o3}, $[256, 2, 2]$	E_o=$\texttt{Flatten}(L_{o2})$, $[1024]$
	//Produce observation-specific posterior parameters.	
6	E_o, $[1024]$; $\mathbf{a}_t$, [A]	$\mathbf{l'}_{\text{posterior}}$=$\texttt{Linear(Concat(}E_o\texttt{, }\mathbf{a}_t\texttt{))}$, $[K_1 \cdot K_2]$
	//Produce final posterior parameters as log-space addition to prior parameters.	
7	$\mathbf{l'}_{\text{posterior}}$, $[K_1 \cdot K_2]$; $\mathbf{l}_{\text{prior}}$, $[K_1 \cdot K_2]$	$\mathbf{l}_{\text{posterior}} = \mathbf{l'}_{\text{posterior}} + \mathbf{l}_{\text{prior}}$, $[K_1 \cdot K_2]$
	//Transform logits of all K_1 component dists.	
8	$\mathbf{l}_{\text{posterior}}$,$[K_1 \cdot K_2]$	$\mathbf{v}_{\text{posterior}}$=$\texttt{softmax}(\mathbf{l}_{\text{posterior}}, \text{axis} = K_2)$, $[K_1 \cdot K_2]$
Observation Likelihood $p_\theta(\mathbf{o}_t \mid \mathbf{z}_t) = \mathcal{N}(\cdot; \mu_o, I)$		
	//Embed latent state with affine layer to consistently-sized vector.	
1	$\mathbf{z}_t$, $[K_1 \cdot K_2]$	E_z=$\texttt{Linear(}\mathbf{z}_t\texttt{)}$, $[1024]$
	//Apply transposed-CNN to decode to observation dimensionality.	
2	E_z, $[1024]$	L_{z0}=$\texttt{ELU(BN(ConvTranspose2d(1024, 128, 5, 2)))}(E_z)$
3	L_{z0}, $[128, 5, 5]$	L_{z1}=$\texttt{ELU(BN(ConvTranspose2d(128, 64, 5, 2)))}(L_{z0})$
4	L_{z1}, $[64, 13, 13]$	L_{z2}=$\texttt{ELU(BN(ConvTranspose2d(64, 32, 6, 2)))}(L_{z1})$
5	L_{z2}, $[32, 30, 30]$	L_{z3}=$\texttt{ELU(BN(ConvTranspose2d(32, 3, 6, 2)))}(L_{z2})$
6	L_{z3}, $[3, 64, 64]$	L_{z4}=$\texttt{ELU(BN(ConvTranspose2d(32, 3, 6, 2)))}(L_{z3})$
	//Output of final layer is the mean of the observation likelihood.	
7	L_{z4}, $[3, 64, 64]$	μ_o=$\texttt{ELU(BN(ConvTranspose2d(3, 3, 1, 1)))}(L_{z4})$
Belief $q_\phi(\mathbf{z}_{t+1} \mid \mathbf{o}_{\leq t+1}, \mathbf{a}_{\leq t}) = q(\mathbf{z}_{t+1} \mid \mathbf{h}_{t+1}) = {}^1\!/\!_P \sum_{p=0}^{P} w_p q_\phi(\mathbf{z}_{t+1,p} \mid \mathbf{o}_{\leq t+1}, \mathbf{a}_{\leq t}, \mathbf{z}_{tp}, g_t)$		
	//Compute unnormalized log-space particle weights of P latent particles	
1	$\{(\mathbf{z}_{t,p}, \mathbf{z}_{t-1,p})\}_{p=1}^{P}$	$\log \hat{w} = \log \frac{p_\theta(\mathbf{z}_{t+1,p} \mid \mathbf{z}_{t,p}, \mathbf{a}_t) p_\theta(\mathbf{o}_t \mid \mathbf{z}_{t,p})}{q_\phi(\mathbf{z}_{t+1,p} \mid \mathbf{o}_{\leq t+1}, \mathbf{a}_{\leq t}, \mathbf{z}_{t,p})}$, $[P]$
	//Form belief from weighted mixture over particles.	
2	w, $[P]$	$\texttt{MixtureSameFamily}(\{q_\phi(\mathbf{z}_{t+1,p} \mid \mathbf{o}_{\leq t+1}, \mathbf{a}_{\leq t}, \mathbf{z}_{t,p})\}_{p=1}^{P}, w)$
Latent Visitation Model $\bar{q}_{t'}(\mathbf{z}) = {}^1\!/\!_{t'} \sum_{t=0}^{t'} q_\phi(\mathbf{z}_t \mid \mathbf{o}_{\leq t}, \mathbf{a}_{\leq t-1})$		
	//Define uniform mixture weights.	
1	$\emptyset$	$c_0 = {}^1\!/\!_{t'}\texttt{torch.ones((1,t’))}$, $[1, t']$
	//Form uniform mixture over previous beliefs.	
2	$\{q_\phi(\mathbf{z}_t \mid \mathbf{o}_{\leq t}, \mathbf{a}_{\leq t-1})\}_{t=0}^{t'}; c_0, [1, t']$	$\bar{q}_{t'}(\mathbf{z})$ =$\texttt{MixtureSameFamily}(\{q_\phi(\mathbf{z}_t \mid \mathbf{o}_{\leq t}, \mathbf{a}_{\leq t-1})\}_{t=0}^{t'}, c_0)$
Policy $\pi(\mathbf{a}_t \mid \mathbf{v}_{\text{posterior}}) =$ $\texttt{Categorical}(\cdot; p_a)$		
1	$\mathbf{v}_{\text{posterior}}$, $[K_1 \cdot K_2]$	h_0=$\texttt{tanh(Linear(}\mathbf{v}_{\text{posterior}}\texttt{))}$, $[128]$
2	h_1, $[128]$	h_1=$\texttt{tanh(Linear(}h_0\texttt{))}$, $[128]$
	//Compute categorical action distribution parameters.	
3	h_2, $[128]$	$\hat{p}_a$=$\texttt{Linear(}h_2\texttt{)}$, $[\lvert\mathcal{A}\rvert]$
4	$\hat{p}_a$, $[\lvert\mathcal{A}\rvert]$	p_a=$\texttt{sum(}\hat{p}_a\texttt{, -1)}$, $[\lvert\mathcal{A}\rvert]$

Table D.2: **Latent state-space model, visitation model, and policy architectural details:** The inputs to the latent state-space model are RGB images $\mathbf{o}_t \in [0, 1]^{3 \times 64 \times 64}$ and actions $\mathbf{a}_t \in \{0, 1\}^A$ (one-hot). Pytorch layer notation is used as shorthand. g_t represents the GRU state at t.