Solipsistic Reinforcement Learning

Anonymous authors
Paper under double-blind review

Abstract

We present a model-based reinforcement learning framework that aims to tackle environments with high dimensional state spaces. In contrast to traditional approaches, agents under our framework learn a low dimensional internal representation of the environment while avoiding the need to learn a generative model of the environment itself. This solipsistic representation is trained to encode a belief that is consistent with the dynamics of the environment and is then exploited for effective planning. We present specific cases of our framework with choices of model and corresponding planning algorithms that can deal with both discrete and continuous state environments. We demonstrate empirically gains in efficiency over existing model-free methods when learning directly from pixels and analyze the properties of our learned representations.

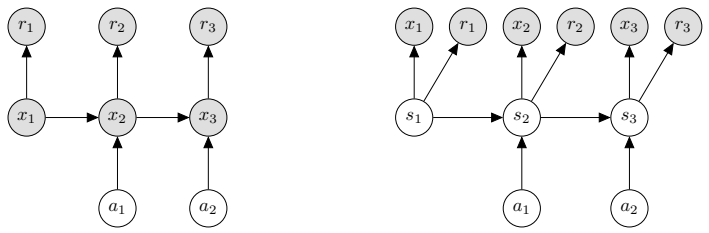
1 Introduction

The real world is complex and a learning agent must be able to recognize relevant signals to decide what actions to take towards reaching a goal. The focus of our work is to form environment representations for model-based planning and reward prediction *without* having to learn a generative model of the potentially complex environment. To motivate our approach, Pavlov's dog (Dennis and Mitterer, 2004) learns to associate the sound of a bell with the eventual reward of food, despite sensory distractions. There are classically two interpretations: (a) a model-free interpretation is that the dog learns a value $v(x_t)$ (expectation of eventual food reward) as a function of the environment state x_t at time t; (b) a standard model-based interpretation is that the dog models the environment and can use that to predict the future $p(x_{t+k}|x_t)$ and any eventual reward. In contrast to these standard approaches, we posit an alternative model-based interpretation in which the dog forms an internal 'solipsistic'[1] low-dimensional representation s_t as a function of the external environment state x_t and forms a predictive model $p(s_{t+k}|s_t)$ of the representation, without learning a model of the environment itself. This representation is useful if the dog is able to accurately predict eventual reward r_{t+k} using the solipsistic transition $p(s_{t+k}|s_t)$ and reward model $p(r_{t+k}|s_{t+k})$.

Figure 1: Pavlov's dog. After hearing the bell, the dog receives a food reward, with the time between the bell and the reward gradually increased. The dog learns to associate the sound of the bell with the eventual arrival of food, salivating due to this expectation.

In the Reinforcement Learning (RL) setting an agent observes state x_t from the environment at time-step t, takes action a_t, and subsequently observes x_{t+1} and reward r_{t+1}. The goal of the agent is to learn (through interactions with the environment) how to take actions that result in favorable long-term rewards (Sutton and Barto, 2018). A standard RL assumption is that there is an underlying Markov Decision Process with transition $p(x_{t+1}|x_t, a_t)$ (Deisenroth and Rasmussen, 2011; Gal et al., 2016; Amos et al., 2018; Chua et al., 2018); rewards are functions of the observed state and the goal is usually to take

[1] We use 'solipsism' to refer to the philosophy that only an internal representation of the world may exist (Blackburn, 2005). In our context, the agent can plan on the basis of an internal dynamical representation of the external world.



(a) Observed Markov Model (b) Latent Variable Model.

Figure 2: Graphical models for model-based RL. Shaded nodes denote observed quantities. (a) Model dynamics and reward in the original space x and (b) in latent space s.

actions that maximize cumulative reward, see Figure 2a[2]. The action a_t depends on the state x_{t-1}, meaning that the state x_{t-1} is revealed before the action a_t is decided.

In model-based RL, we attempt to learn the model of the transition dynamics. Compared to model-free approaches, model-based RL can be significantly more sample efficient (Deisenroth and Rasmussen, 2011; Gal et al., 2016; Amos et al., 2018; Chua et al., 2018). However, for environments with high-dimensional states (such as an image pixels) the complexity and potential redundancy in the observations can make learning the environment dynamics using a model difficult and potentially unnecessary (Ha and Schmidhuber, 2018; Hafner et al., 2019b).

A recent trend is to learn a lower dimensional representation s_t that is used to model relevant dynamics and reward prediction, such as PlaNet Hafner et al. (2019b), World Model (Ha and Schmidhuber, 2018) and other variants (Chiappa et al., 2017; Ha and Schmidhuber, 2018), see Figure 2b. The usual strategy is to train a latent variable generative model $p(x_t) = \int p(x_t|s_t)p(s_t)ds_t$ with an encoding-decoding structure (Kingma and Welling, 2014), while jointly training a dynamics model $p(s_{t+1}|s_t, a_t)$ in the lower dimensional latent space (Hafner et al., 2019b; Chiappa et al., 2017; Ha and Schmidhuber, 2018; Hafner et al., 2020). The learned model infers a latent representation s_t given a state observation x_t (for example a sample from the posterior $p(s_t|x_t) \propto p(x_t|s_t)p(s_t)$) that can be used by the dynamics model for efficient planning. Arguably, a limitation of these recent approaches is that they spend significant computational effort on learning a generative model of the high-dimensional state x_t – however, this generative model is not used directly during the planning phase. The learned representation s_t in these approaches is therefore likely encoding redundant information about the environment x_t.

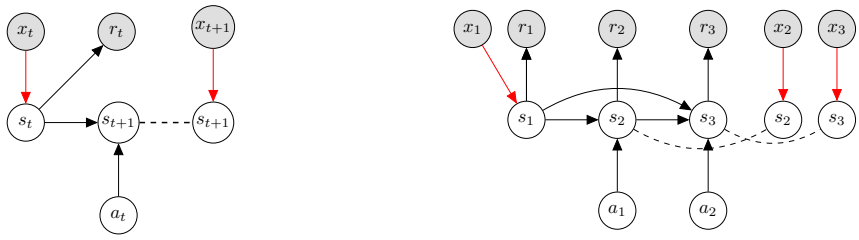
An alternative is to use model-free algorithms such as Deep Q-Networks (DQN), Proximal Policy Optimization (PPO), amongst others (Mnih et al., 2015; Schulman et al., 2017; 2015; Babaeizadeh et al., 2017). The upside is that these approaches avoid creating a generative model of the environment by learning a policy from pixels to state values. A potential downside is that they suffer from poor sample efficiency compared to model based approaches.

The question we therefore study here, similar to other recent research work in this area Hafner et al. (2019a), is whether it is possible to perform model-based RL *without* making a generative model of the environment. If this were possible, we could potentially reap the benefits of the sample-efficiency of model-based RL, without the need to model complex high-dimensional observations.

2 Solipsistic Representations

A solipsistic representation s_t of an observation x_t is one that is consistent (the predicted next solipsistic state s_{t+1} given action a_t matches the observed next solipsistic state s_{t+1}) and informative (one can predict the reward well using s_t). A solipsistic Markov model is depicted in Figure 3a, where we remove the arrow from s to x in the latent variable model and instead introduce a recognition distribution $p(s_t|x_t)$, the purpose of which is to encode only the information in x_t that is needed to effectively learn the dynamics and reward.

[2]Alternatively one may use a higher-order Markov model $p(x_{t+1}|h_t)$ where $h_t = \{s_{1:t}, a_{1:t}\}$ is the history of states and actions up to time point t (Chiappa et al., 2017).



(a) Solipsistic Markov Model

(b) Solipsistic Memory Model

Figure 3: Graphical models of the solipsistic Markov model (a) and memory model (b). We color the edge from the observation to the latent state to highlight that this is not a generative model of the environment. A dashed line indicates the consistent relationship between solipsistic state prediction and future state recognition, as discussed in section 2.

We will use throughout a toy 'MNIST game' to help build intuition, see Figure 4. Each observation x_t is a 28×28 MNIST image representing a digit from 0 to 9. The agent has two possible actions: 'minus 1' or 'plus 1'; the environment shows the resulting digit's image. The digit will stay the same when taking action 'minus 1' from digit 0 and 'plus 1' from digit 9. The game is initialized at digit 4. The reward r_t is 1 if the state is an image of digit 9; otherwise the reward $r_t = 0$. Whilst the observation x_t is a 784 dimensional image, clearly the underlying dynamics is representable by an integer $s_t \in \{0, \ldots, 9\}$.

Solipsistic Consistency We wish to ensure that, for a given recognition distribution $p_{\text{rec}}(s_t|x_t)$, the dynamics of the solipsistic model are consistent with the dynamics of the true environment when training the model under sampled trajectories. Thus, in the setting of Pavlov's dog, if the model maps the current environment x_t to the internal state $x_t \to s_t$ that represents hearing the bell $s_t = \mathsf{bell}$, and the dog predicts $s_t \to s_{t+1}$ from this that $s_{t+1} = \mathsf{food}$, then we must have that the next external state $x_{t+1} \to s_{t+1}$ indeed maps to food. This ensures that solipsistic transitions are effective for planning. Similarly, in the MNIST game, we need to force the solipsistic model to predict a digit that is consistent with the image that would appear in the next time step under the true environment transition. More specifically, we assume a Markov transition distribution $p_{\text{tran}}(s_{t+1}|s_t, a_t)$ which takes the current solipsistic state and action as input and gives the distribution for the next solipsistic state s_{t+1}. Given x_t and a_t, the solipsistic state distribution at time $t+1$ can be predicted using

$$p_{\text{pred}}(s_{t+1}|x_t, a_t) = \sum_{s_t} p_{\text{tran}}(s_{t+1}|s_t, a_t) p_{\text{rec}}(s_t|x_t). \tag{1}$$

We want this predicted distribution to be consistent with the recognition distribution $p_{\text{rec}}(s_{t+1}|x_{t+1})$ given the observation x_{t+1} from the next time step. To achieve this we introduce an agreement objective, for example the KL divergence,

$$\text{KL}(p_{\text{pred}}(s_{t+1}|x_t, a_t) || p_{\text{rec}}(s_{t+1}|x_{t+1}))\,. \tag{2}$$

Reward Consistency For effective planning, we want the solipsistic representation to be useful for reward prediction $p_{\text{rew}}(r_t|s_t)$. Given a solipsistic state s_t and observed reward r_t we use maximum likelihood to learn the reward model $p_{\text{rew}}(r_t|s_t)$. This is equivalent to min-

(a) States of the environment

(b) Transition dynamics

Figure 4: MNIST game. (a) The observation x_t is one of the 10 images. (b) Given a 'plus 1' action, the following image x_{t+1} is a higher digit and vice versa for 'minus 1'.

imizing the KL divergence $\mathrm{KL}(\tilde{p}(r_t)||p_{\mathrm{rew}}(r_t|s_t))$ between the empirical reward distribution $\tilde{p}(r_t)$ that places all mass in the observed reward r_t and the model $p_{\mathrm{rew}}(r_t|s_t)$.

Solipsistic Contrast Finally, the intention of the recognition distribution $p_{\mathrm{rec}}(s_t|x_t)$ is to filter out redundant information when producing the solipsistic representation. To achieve this, we need it to learn to distinguish if the state information is useful or not for both reward prediction and transition dynamics. For example in the MNIST game, the recognition distribution may just focus on the image backgrounds which are stationary over time and the solipsistic transition function could just learn the identity mapping. Although this forms a consistent solipsistic representation, it would be useless for planning. Furthermore, having a reward objective is not sufficient to avoid this behavior. Features of the state which inform reward prediction may not be the same as those which inform a useful dynamics model. For example in the MNIST game, the recognition distribution may just keep the features which are relevant to distinguish if an image is 9 or not for predicting the reward, but ignore other useful information about the system dynamics.

One solution is to additionally encourage the agent's solipsistic dynamics to be inconsistent with trajectories which are not observed in reality. In other words, we want an objective that forces the predictive distribution $p_{\mathrm{pred}}(s_{t+1}|x_{,}a_t)$ to be different from $p_{\mathrm{rec}}(s_i|x_i)$ for $i \neq t+1$ [3]. We thus maximize the expected KL divergence (for convenience dropping henceforth the "pred", "rec", "rew" and "tran" subscripts)

$$\mathbb{E}_{i\neq t+1}\left[\mathrm{KL}(p(s_{t+1}|x_t,a_t)||p(s_i|x_i))\right]. \tag{3}$$

This solipsistic contrast term is similar to a contrastive loss used for representation learning or self-supervised learning (Chopra et al., 2005; Hadsell et al., 2006; Chen et al., 2020). Since the KL divergence is unbounded, we use a positive constant cap m.

Overall Objective: For a trajectory of length T the overall objective is to minimize,

$$\frac{1}{T-1}\sum_{t=1}^{T-1}\mathrm{KL}(p(s_{t+1}|x_t,a_t)||p(s_{t+1}|x_{t+1})) + \frac{\lambda_r}{T}\sum_{t=1}^{T}\mathrm{KL}(\tilde{p}(r_t)||p(r_t|s_t))$$
$$+\frac{\lambda_s}{T-1}\sum_{t=1}^{T-1}\mathbb{E}_{i\neq t+1}\left[\max(0, m-\mathrm{KL}(p(s_{t+1}|x_t,a_t)||p(s_i|x_i)))\right] \tag{4}$$

with respect to the parameters of the recognition distribution $p(s_t|x_t)$, transition distribution $p(s_{t+1}|s_t,a_t)$ and reward distribution $p(r_t|s_t)$; λ_s, λ_r and m are user chosen hyper-parameters. In our experience, the results are not particularly sensitive to the choice of these parameters, see section 3. For tasks that require accurate long-term planning, prediction errors using the simple solipsistic Markov model may accumulate during roll-out[4]. In such cases we consider a Solipsistic Memory model(SMM) whose transition depends on all past solipsistic states and actions – see Figure 3b. Within our overall objective, we replace $p(s_{t+1}|x_t,a_t)$ with $p(s_{t+1}|x_t,h_t)$ where $h_t = \{s_{1:t}, a_{1:t}\}$ and use a recurrent neural network (Hochreiter and Schmidhuber, 1997; Chung et al., 2014) to learn the transition dynamics of the solipsistic model.

2.1 Acting in the Environment and Planning

For a given solipsistic model, we take a sequence of actions using a re-planning procedure: we observe x_1 and determine the first solipsistic state distribution using the recognition process $p(s_1|x_1)$. We then determine a_1^* using a planning procedure (e.g. dynamic programming, or sampling from a trained parameterized policy, as we discuss below) and take this action in the environment, observing the resulting x_2. We repeat this recognition and planning procedure until time T, observing x_t at each step and then planning the best next action. This process of re-planning at every time step helps prevent the accumulation of prediction

[3] More generally, in the case of multiple trajectories, we can sample s_i from other trajectories.

[4] Whilst the underlying physical dynamics of a problem might be Markovian, any pixel based representation will result in discretization error. For long sequences, small discretization errors can accumulate, resulting in poor long term prediction unless a longer term history is used.

errors from our model and is efficient since our solipsistic representation is low dimensional. How planning and acting in the environment is folded into the overall RL process of model and policy training is described in detail in algorithm 1, appendix A.

2.1.1 Planning in the Solipsistic Markov Model

Given the observation x_1, we would like to predict the expected rewards we would obtain by taking a sequence of subsequent actions $a_{1:T-1}$. The recognition distribution enables us to determine the distribution for the first solipsistic state $p(s_1|x_1)$. Then, given x_1 and $a_{1:T-1}$, the state-action trajectory can be described by the distribution

$$p(s_{1:T}|x_1, a_{1:T-1}) = p(s_1|x_1) \prod_{t=1}^{T-1} p(s_{t+1}|s_t, a_t). \tag{5}$$

For planning, the goal is to maximize the cumulative reward by choosing a sequence of actions. For a discrete solipsistic state space, the objective is

$$\sum_{s_{1:T}} \left(\sum_{t=1}^{T} R(s_t) \right) p(s_1|x_1) \prod_{t=2}^{T} p(s_t|s_{t-1}, a_{t-1}), \tag{6}$$

where we use the reward function defined as $R(s_t) \equiv \mathbb{E}_{p(r_t|s_t)}[r_t]$. If we interpret the solipsistic model as a Markov Decision Process (MDP) [5] then the optimal action sequence has value

$$\max_{a_1} \sum_{s_1} p(s_1|x_1) \cdots \max_{a_{T-2}} \sum_{s_{T-1}} p(s_{T-1}|s_{T-2}, a_{T-1}) \max_{a_{T-1}} \sum_{s_T} p(s_T|s_{T-1}, a_{T-1}) \sum_{t=1}^{T} R(s_t). \tag{7}$$

This is readily solved by dynamic programming, which we describe below.

Value Estimation We let $V(s_T) = R(s_T)$. For $t = T-1, \ldots, 2$ and each state of s_t we calculate

$$V(s_t) = R(s_t) + \max_{a_t} \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) V(s_{t+1}). \tag{8}$$

The first optimal action can be computed using the value function

$$a_1^* = \arg\max_{a_1} \sum_{s_1} p(s_1|x_1) \sum_{s_2} p(s_2|s_1, a_1) V(s_2). \tag{9}$$

We then take the first optimal action a_1^* in the real environment to get observation x_2, and do re-planning based on the new solipsistic distribution $p(s_2|x_2)$. The general procedure is to repeat: (1) take the action a_t^* in the environment and get the new observation x_{t+1}, (2) use the recognition function to compute $p(s_{t+1}|x_{t+1})$, (3) compute the next action a_{t+1}^* using

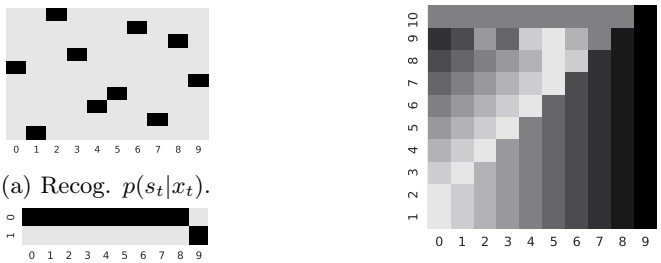
$$a_{t+1}^* = \arg\max_{a_{t+1}} \sum_{s_{t+1}} p(s_{t+1}|x_{t+1}) \sum_{s_{t+2}} p(s_{t+2}|s_{t+1}, a_{t+1}) V(s_{t+2}). \tag{10}$$

In situations where a full action sequence is desired before interacting with the environment, see appendix C. By using this re-planning scheme, we can sequentially decide the optimal action sequence under our model, which we demonstrate in our MNIST game in section 3.1. For continuous solipsistic states and non-linear transition dynamics, exact dynamic programming is usually not available. See (Bertsekas, 1995) for alternative approximate dynamic programming techniques.

2.1.2 Planning in the Solipsistic Memory Model

In the SMM dynamic programming is problematic and we instead learn a policy $p_w(a_t|s_t)$, parameterized by a neural network. Here we assume that the state s_t contains sufficient

[5] See appendix B for an alternative method where we do not assume a MDP.



(a) Recog. $p(s_t|x_t)$.

(b) Reward $p(r_t|x_t)$.

(c) $V(s_t)$ for t from 1 to 10.

Figure 5: The learned solipsistic model for the MNIST game. The pixel images x_t are represented by their corresponding number on the x-axis. (a) The y-axis is the solipsistic state s_t. The model learns to associate an image x_t with a unique solipsistic state s_t, with $p(s_t|x_t)$ being almost deterministic. (b) The learned model predicts the instantaneous reward correctly. (c) The y-axis is the time step from 1 to 10. The value is normalized within each time-step, the darker patches indicate higher values. Given any state at time step 1, the optimal action is always 'plus 1'.

information to determine the best action; however, in order to accurately track long term behavior, we need to use the SMM dynamics to track the state. This is typically the case in pixel-based planning in which the best action is readily determinable from the current state; however, keeping track of the long-term consequences of a sequence of actions requires using a history of states (due to discretization errors in the image rendering). The resulting objective to maximize is

$$E(w) \equiv \int \left(\sum_{t=1}^{T} R(s_t) \right) p(s_1|x_1) \prod_{t=2}^{T} p(s_t|h_{t-1}) p_w(a_{t-1}|s_{t-1}) ds_{1:T} da_{1:T-1}. \tag{11}$$

where, for discrete actions, the integral over $a_{1:T-1}$ is replaced by summation. Previous work has demonstrated that Variational Optimization (VO) style algorithms (Staines and Barber, 2012; Salimans et al., 2017) have an advantage over policy gradients for environments with long time horizons. We therefore use a standard VO algorithm to learn the policy parameters w, see appendix D.

3 Experiments

Rather than showing state-of-the-art across a range of RL challenges, the goal of the experiments is to confirm our hypothesis that model-based RL can be achieved without requiring a generative model of the observations x_t. We discuss the simple MNIST game and RL from pixels using continuous s_t for a version of the Cartpole benchmark from OpenAI gym (Brockman et al., 2016). Full details of the models and training are given in appendix E.

3.1 MNIST Game

For the solipsistic state we assume we know the true number of states $s_t \in \{1, \ldots, 10\}$. The recognition distribution $p(s_t|x_t)$ is parameterized by a convolutional network. We parameterize the transition $p(s_{t+1}|s_t, a_t)$ with two normalized 10×10 matrices $p(s_{t+1}|s_t, a_t = 0)$ and $p(s_{t+1}|s_t, a_t = 1)$. The reward distribution $p(r_t|s_t)$ is parameterized by a normalized 10×2 matrix. At each episode we select random actions $a_{1:14}$ and collect a single trajectory $x_{1:15}$ to add to our memory[6]. Then during model training, we iteratively sample batches of consecutive states from memory with batch size 64 and update the model using equation 4 and ADAM (Kingma and Ba, 2015). This process then repeats at each episode.

We can see from Figure 5a that the 10 different images are assigned to 10 different solipsistic states with high probability. In Figure 5b, we see successful reward predictions, with the image state 9 having high probability of reward 1. At test time, we randomly initialize the image state and set the horizon for planing to $T = 10$, since for any given state at $t = 1$, the agent can reach the goal state within 10 steps. In Figure 5c we plot the value of each state from $t = 1, \ldots, 10$. The optimal action at time 1 is 'plus 1' for each initial state; similarly the subsequent optimal action is always 'plus 1', thus correctly solving the problem.

[6]The MNIST game is simple enough that, during model training, random exploration is sufficient.

3.2 Gym Control : Binary Action Cartpole

In Cartpole, a pole is attached by an un-actuated joint to a cart that moves along a frictionless track, which can be controlled by applying a force of $+1$ or -1 to the cart at each time step. A reward of 1 is received at every time step that the pole remains upright and the episode terminates when it falls over or the cart moves too far from the center. A maximum horizon of 200 time steps is generally used.

Instead of using the low-dimensional states provided by the OpenAI Gym, we use the rendered image frames as the state observations. We first gray scale pre-process each video frame and down-sample to produce a 64×64 frame f_t at time t; three consecutive frames are then stacked to represent each state observation, $x_t = \{f_t, f_{t-1}, f_{t-2}\}$ for $t \geq 3$. This provides higher-order dynamics information like speed, which are hidden in a single frame. For simplicity we assume a stationary initial position (initial pole position is upright) and define $x_1 = \{f_1, f_1, f_1\}$ and $x_2 = \{f_2, f_1, f_1\}$. We use the SMM since this gives accurate long-term prediction, despite discretization error from the pixel rendering of the true continuous underlying dynamics.

We follow algorithm 1 (see appendix A) to interact with the environment and learn the policy and model. We set $s_t \in \mathbb{R}^{16}$ and choose $p(s_t|x_t) = \delta(s_t - g_\theta(x_t))$, $p(s_{t+1}|h_t) = \delta(s_{t+1} - f_\theta(h_t))$. The recognition function g_θ and transition function f_θ are a convolutional neural network and recurrent network respectively. The reward distribution $p_\theta(r_t|s_t)$ is a Bernoulli distribution with the probability parameterized using a small neural network with a sigmoid output. All model parameters are trained jointly using the ADAM optimizer. Since the KL divergence between two delta distributions is not formally defined for equation 4, we use the spread KL divergence (Zhang et al., 2020) with fixed Gaussian spread noise that has variance 0.5, resulting in a square loss objective.

The policy, section 2.1.2, $p_\theta(a_t|s_t)$ is a Bernoulli distribution with the probability parameterized using a small feed-forward network with a sigmoid output and trained using VO - see appendix E. When evaluating the trained policy in the real environment we take the most likely action at each re-planning step.

We compare to DQN and PPO (Mnih et al., 2015; Schulman et al., 2017) following their standard open source implementations adapted for acting directly in pixel space – see appendix E.3 for details. In Figure 6 we report the average reward over 5 different runs for all methods, with each run carried out using different random parameter seeds, but the same fixed initial position of the environment.

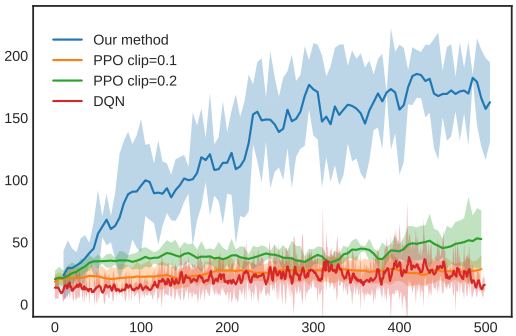


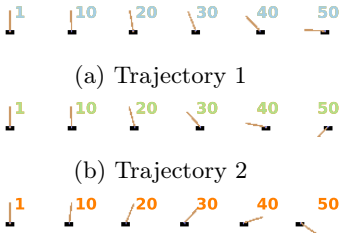
Figure 6: Evaluating learned policies. x-axis is number of trajectories sampled from the environment. y-axis is cumulative reward received, averaged over 5 training runs. The data is smoothed using a moving average with window size 3.

Compared to PPO and DQN our solipsistic model-based approach is significantly more sample efficient, learning to balance the pole for over 150 time steps, after seeing only 300 trajectories. This demonstrates that our solipsistic approach is accurately modeling both the relevant environment dynamics and the reward. For this setting, we found PPO and DQN struggled to balance the pole for over 50 time steps on average after 2500 sampled trajectories from the environment - see Figure 9 in appendix E.3.

3.2.1 Learned Representations

Of interest for a qualitative analysis on the recognition function and learned solipsistic representations are: (1) Are the solipsistic trajectories consistent with their corresponding observation trajectories? (2) has the recognition function learned to filter out redundant

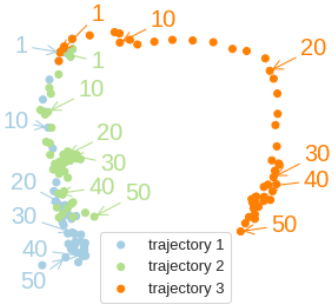
information? In support of (2), in Figure 7d we illustrate that the solipsistic trajectories have smooth transitions and are disentangled in accordance with their corresponding trajectories in pixel space. We address (2) in appendix E.4.



(a) Trajectory 1

(b) Trajectory 2

(c) Trajectory 3



(d) Solipsistic state trajectories

Figure 7: (a,b,c) show three test trajectories, where we plot the frames $\{f_1, f_{10}, \ldots, f_{50}\}$. (d) We construct $x_1, \ldots, x_{50}$ by stacking the successive frames within each trajectory, and then produce $s_1, \ldots, s_{50}$. We use PCA to project the solipsistic states to 2D for visualization. Trajectories 1 and 2 are similar (the pole falls to the left) and have similar solipsistic trajectories. Trajectory 3 shows the object fall to the right and the solipsistic representation is far away from that of trajectory 1 and 2.

4 Related Work

Contrastive learning is widely used in the field of representation learning (Bengio et al., 2013). The aim is to encourage similar datapoints to have similar representations (Chopra et al., 2005; Hadsell et al., 2006; Chen et al., 2020). In RL, this idea has been used to improve the data efficiency of model-free algorithms by treating contrastive learning as an auxiliary task (Oord et al., 2018; Srinivas et al., 2020; Jaderberg et al., 2017). The contrastive objective we use (section 2) instead aims to prevent the recognition function from learning a trivial solution to ensure the solipsistic representation is useful for planning.

Most closely related to our work is Hafner et al. (2019a) which uses a Recurrent State Space Model without a decoder component, akin to our Solipistic Markov Model setup. They use the variational information bottleneck (VIB) principle (Tishby et al., 2000; Alemi et al., 2016) to derive the following regularizing term for their learning objective:

$$\log p(s_t|x_t) - \log \sum_i p(s_t|x_i), \tag{12}$$

where the summation is over the observations in the current sequence batch. The paper shows that this term keeps s_t predictable from the current image, whilst also keeping the latent representations diverse. Although this work also achieves model based RL in the representation space without reconstructing the original image state, their regularized VIB objective is different to our Solipsistic consistent-contrast objective (equation 4). We leave detailed comparisons in both theory and practice to future work.

5 Summary

We introduced Solipsistic Reinforcement Learning, a new model-based reinforcement learning framework that learns useful latent representations of the environment for planning and reward prediction, without constructing a generative model of the environment. Our work is consistent with the recent general trend away from modeling the dynamics of high dimensional spaces and towards learning models that more directly solve the task at hand. Whilst model-based reinforcement learning is arguably preferable to model-free alternatives, particularly because of data efficiency, most previous approaches do not learn in an end-to-end fashion and also require an explicit model of the environment. We hope therefore that we have shown that there is scope to solve reinforcement learning problems in a model-based way, but without the downsides of requiring complex models of the environment.

References

A. A. Alemi, I. Fischer, J. V. Dillon, and K. Murphy. Deep variational information bottleneck. *arXiv preprint arXiv:1612.00410*, 2016.

B. Amos, L. Dinh, S. Cabi, T. Rothörl, S. G. Colmenarejo, A. Muldal, T. Erez, Y. Tassa, N. de Freitas, and M. Denil. Learning Awareness Models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz. Reinforcement Learning through Asynchronous Advantage Actor-Critic on a GPU. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

Y. Bengio, A. Courville, and P. Vincent. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8): 1798–1828, 2013.

D. P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena scientific Belmont, MA, 1995.

S. Blackburn. *The Oxford Dictionary of Philosophy*. OUP Oxford, 2005.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym, 2016.

T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A Simple Framework for Contrastive Learning of Visual Representations. *arXiv:2002.05709*, 2020.

S. Chiappa, S. Racanière, D. Wierstra, and S. Mohamed. Recurrent Environment Simulators. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

S. Chopra, R. Hadsell, and Y. LeCun. Learning a Similarity Metric Discriminatively, with Application to Face Verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, volume 1, pages 539–546. IEEE, 2005.

K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep Reinforcement Learning in a Handful of Trials Using Probabilistic Dynamics Models. In *Advances in Neural Information Processing Systems*, pages 4754–4765, 2018.

J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv:1412.3555*, 2014.

M. Deisenroth and C. E. Rasmussen. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, pages 465–472, 2011.

C. Dennis and J. Mitterer. Introduction to Psychology: Gateways to Mind and Behavior, 2004.

P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov. OpenAI Baselines. `https://github.com/openai/baselines`, 2017.

T. Furmston and D. Barber. Efficient Inference in Markov Control Problems. In F. G. Cozman and A. Pfeffer, editors, *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, UAI 2011, Barcelona, Spain, July 14-17, 2011*, pages 221–229, 2011.

Y. Gal, R. McAllister, and C. E. Rasmussen. Improving PILCO with Bayesian Neural Network Dynamics Models. In *Data-Efficient Machine Learning workshop, ICML 2016*, volume 4, page 34, 2016.

D. Ha and J. Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, pages 2450–2462, 2018.

R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality Reduction by Learning an Invariant Mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2006*, volume 2, pages 1735–1742. IEEE, 2006.

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

D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning Latent Dynamics for Planning from Pixels. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*, pages 2555–2565, 2019b.

D. Hafner, T. P. Lillicrap, J. Ba, and M. Norouzi. Dream to Control: Learning Behaviors by Latent Imagination. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural computation*, 9(8): 1735–1780, 1997.

S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*, pages 448–456, 2015.

M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement Learning with Unsupervised Auxiliary Tasks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. In Y. Bengio and Y. LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

A. v. d. Oord, Y. Li, and O. Vinyals. Representation Learning with Contrastive Predictive Coding. *arXiv:1807.03748*, 2018.

A. Paszke. Reinforcement Learning (DQN) Tutorial. `https://github.com/pytorch/tutorials`, 2020.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *arXiv:1703.03864*, 2017.

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust Region Policy Optimization. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*, pages 1889–1897, 2015.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347*, 2017.

A. Srinivas, M. Laskin, and P. Abbeel. CURL: Contrastive Unsupervised Representations for Reinforcement Learning. *arXiv:2004.04136*, 2020.

J. Staines and D. Barber. Variational Optimization. *arXiv:1212.4507*, 2012.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018.

N. Tishby, F. C. Pereira, and W. Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

M. Zhang, P. Hayes, T. Bird, R. Habib, and D. Barber. Spread Divergences. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020*, 2020.

A Overall Learning Procedure

Here we provide the general procedure we follow for tackling RL problems with solipsistic reinforcement learning, where we assume the use of the memory model and a parameterized policy.

Algorithm 1: Overall learning procedure - with parameterized policy and memory model

Set constants I, N_{model}, N_{policy}, $T_{collect}$, T_{model}, T_{policy}, B (model batch size), ϵ (exploration noise), J (number Monte Carlo samples in VO)

Initialize $\mathcal{M} \leftarrow \emptyset$ (trajectory memory)

Initialize Parameters of $p_{\text{tran}}, p_{\text{rec}}$, p_{rew}, p_{w_0} (to random policy)

for *iteration* $i = 1, \ldots, I$ **do**

  `/* Trajectory collection */`

  for *collection step* $c = 1, \ldots, C$ **do**

    Reset environment

    for *time step* $t = 1, \ldots, T_{collect}$ **do**

      Observe x_t, r_t and sample $s_t \sim p_{\text{rec}}(s_t|x_t)$

      Sample $\hat{a}_t$ based on exploration strategy using $p_{w_i}(a_t|s_t)$, ϵ ; `// See section E`

      Take action $\hat{a}_t$ in environment

    $\mathcal{M} \leftarrow \mathcal{M} \cup \{(x_t, s_t, \hat{a}_t, r_t)\}_{t=1}^{T_{collect}}$

  `/* Model training */`

  for *training step* $n = 1, \ldots, N_{model}$ **do**

    From memory $\mathcal{M}$ sample batch of trajectories $\{\{(x_t, s_t, \hat{a}_t, r_t)_b\}_{t=1}^{T_{model}}\}_{b=1}^{B}$

    Jointly update parameters of $p_{\text{tran}}, p_{\text{rec}}$, p_{rew} using equation 4 and ADAM

  `/* Policy training */`

  Re-initialize w_i to a random policy

  Initialise VO parameter $\mu_1 \leftarrow w_i$

  for *training step* $n = 1, \ldots, N_{policy}$ **do**

    for *sample* $j = 1, \ldots, J$ **do**

      initialize $h_1 \leftarrow \emptyset$

      Sample x_1 from memory $\mathcal{M}$

      Sample $s_1 \sim p_{\text{rec}}(s_1|x_1)$

      Sample $w^j \sim N(\mu_n, 0.2)$

      for *time step* $t = 1, \ldots, T_{policy}$ **do**

        Sample $a_t \sim p_{w_i^j}(a_t|s_t)$

        $h_t \leftarrow h_t \cup (s_t, a_t)$

        Predict $R(s_t)$

        Sample $s_{t+1} \sim p_{\text{tran}}(s_{t+1}|h_t)$

      Compute $E(w^j) = \sum_{t=1}^{T} R(s_t)$; `// See section D`

    Compute μ_{n+1} with $\{E(w^j)\}_j^J$ using VO update equation 18 and ADAM

  Update policy parameters $w_{i+1} \leftarrow \mu_{n+1}$

B Best Trajectory Planning

In section 2.1, our goal is to maximize the cumulative reward objective

$$\sum_{s_{1:T}} \left(\sum_{t=1}^{T} R(s_t) \right) p(s_1|x_1) \prod_{t=2}^{T} p(s_t|s_{t-1}, a_{t-1}). \tag{13}$$

We then assume the state s_t is only revealed after we take action a_{t-1} and form the problem as a Markov Decision Process in equation 7. An alternative objective is to directly optimize equation 13 with respect to the action sequence $a_1, \ldots, a_{t-1}$ – we refer to this as the 'best trajectory planning' objective. Although the optimization problem is no longer tractable, an approximate but efficient Expectation Maximization (EM) style algorithm can be used following the lines of (Furmston and Barber, 2011). When applied to the setting of only a final reward, we didn't find any significant improvement over MDP planning; however, we leave a more detailed discussion of this approach for future work.

C Full Trajectory Planning Using the Model

In section 2.1 we discussed a planning approach using our model where we do re-planning each time we sample an action. For the situation that the environment needs an 'immediate' response from the agent (with no time for re-planning), we may decide a sequence of actions purely based on the model predictions given observation x_1. Subsequently, we deploy that action sequence in the environment. We describe the method below.

We first compute the value function using equation 8 and decide the first optimal action based on

$$a_1^* = \arg\max_{a_1} \sum_{s_1} p(s_1|x_1) \sum_{s_2} p(s_2|s_1, a_1) V(s_2). \tag{14}$$

For $t = 2, \ldots, T-1$, we sequentially compute

$$a_t^* = \arg\max_{a_t} \sum_{s_t} p(s_t|x_1, a_1^*, \ldots, a_{t-1}^*) \sum_{s_{t+1}} p(s_{t+1}|s_t, a_t) V(s_{t+1}), \tag{15}$$

where

$$p(s_t|x_1, a_1^*, \ldots, a_{t-1}^*) = \sum_{s_{t-1}} p(s_t|s_{t-1}, a_{t-1}^*) p(s_{t-1}|x_1, a_1^*, \ldots, a_{t-2}^*) \tag{16}$$

for $t \geq 3$. The potential drawback of this approach is that the error will accumulate during the long term predictions using the model. We implemented this method for the MNIST game, but found little difference in the results over re-planning. In that setting the model provides very accurate long term planning. We leave it to future work to assess the performance of this approach in more complex environments.

D Variational Optimization

With VO we model the parameters of the policy with a Gaussian distribution $w \sim \mathcal{N}(w \,|\, \mu, \sigma)$ to form a differentiable upper bound that we can minimise w.r.t μ and σ.

$$U(\mu, \sigma) = \mathbb{E}_{p(w|\mu,\sigma)} \left[-E(w) \right], \tag{17}$$

where $E(w)$ is our memory model planning objective from equation 11. After optimising this bound with respect to μ, σ, we take the final μ as our value for w. We compute gradients with the usual log-derivative trick, using J Monte Carlo samples to approximate the resulting expectation as follows. In practice, we find that learning σ does not improve performance, hence we fix it to a value of 0.2 throughout and minimize w.r.t μ,

$$\frac{\partial U}{\partial \mu} \approx \frac{1}{J} \sum_{j=1}^{J} \frac{\partial}{\partial \mu} \log p(w^j|\mu, \sigma)(-E(w^j)), \tag{18}$$

where w^j refers to the jth Monte Carlo sample of the policy parameters - see section *Policy training* in algorithm 1 for further details. In practice, we also apply *fitness shaping* Salimans et al. (2017) to $E(w^j)$ in equation 18 to make the VO update invariant w.r.t. order-preserving transformations of the reward value.

E Experiment Details

E.1 MNIST Game

In this section we discuss the details of the architectures and hyper-parameter settings used for our toy MNIST game experiment from section 3.1.

Model Architecture The recognition network is composed of a two-layer convolution neural network, followed by a two-layer feed-forward network. The two convolution layers have 10 and 20 filters respectively with stride 2 and kernel size 5. We use max pooling after convolutional layer. We use ReLU activation functions for both the convolutional and feed-forward layers. The forward network has 100 hidden units in each layer and the output size is equal to the number of solipsistic states (in this case 10). We use a softmax function to create the probabilities of the categorical distribution for determining the solipsistic state assignments.

Model Training For our model objective equation 4 we set the hyper parameters as $\lambda_s = 1$, $\lambda_r = 2$ and $m = 5$. We use ADAM (Kingma and Ba, 2015) as the optimizer with learning rate 10^{-4} to train the model for $N_{model} = 3000$ iterations with batch size $B = 64$.

Solipsistic Contrast For computational efficiency, we approximate the contrast term in equation 3 using the following Monte Carlo approximation

$$\mathbb{E}_{i \neq t+1} \left[\mathrm{KL}(p(s_{t+1}|x_t, a_t)||p(s_i|x_i)) \right] \approx \frac{1}{B-1} \sum_{s_j} \mathrm{KL}(p(s_{t+1}|x_t, a_t)||p(s_j|x_j)) \,, \quad (19)$$

where $s_j \in \mathcal{M}(s_{t+1}) \backslash s_{t+1}$ and $\mathcal{M}(s_{t+1})$ is the mini-batch set that s_{t+1} belongs to. So the set $\mathcal{M}(s_{t+1}) \backslash s_{t+1}$ has size $B - 1$.

E.2 Gym Control

In this section we discuss the details of the architectures and hyper-parameter settings used for our Cartpole experiment from section 3.2. For model and policy training we follow algorithm 1.

Model Architecture The recognition network is a four-layer convolution neural network with Batch Norm (Ioffe and Szegedy, 2015) and ReLU activation functions followed by a feed-forward layer with output size equal to the size of the solipsistic representation $dim(S) = 16$. We set the kernel size to 3 for the first three convolutional layers and 5 for the last convolutional layer. We set the channel size to 8 and stride to 1 for the first convolutional layer and 16 with stride 2 for the other three convolutional layers. The policy network is a two layers feed forward neural network with 50 hidden units in each layer, which maps from the solipsistic state to a sigmoid function that parameterizes the probability of a Bernoulli distribution. The RNN we used is a single layer Gated Recurrent Unit (GRU) (Chung et al., 2014) with memory in the first time-step initialized as the first solipsistic state s_1. In each recurrent step, the GRU cell takes one action as input and outputs the prediction of the solipsistic state for the next time step. Therefore, the size of the hidden memory of GRU is equal to the size of the solipsistic state.

Model Training We use ADAM with learning rate 10^{-4}, on batches of size $B = 10$ of sampled environment trajectories of length $T_{model} = 50$, with $\lambda_s = 1$, $\lambda_r = 2$ and $m = 5$.

Policy Training We use VO (section D), with $J = 50$ parameter perturbations, under model prediction roll-outs of length $T_{policy} = 200$ and we train for $N_{policy} = 50$ iterations, using ADAM with learning rate 10^{-2}. We found that re-initializing the policy parameters w after each model update helped overall performance.

Exploration Strategy At each iteration i of algorithm 1 during the *Trajectory collection* phase we collect $C = 5$ trajectories and add them to our memory for subsequent model training. We use an exploration strategy (see line 9 of algorithm 1) during trajectory collection as follows: for $c = 1$ we follow the latest policy $p_{w_i}(a_t|s_t)$ with re-planning (section B) to collect a full trajectory. Then, for $c = 2, \ldots, 5$, we instead follow an ϵ-policy, where we take the action sampled from $p_{w_i}(a_t|s_t)$ with probability ϵ and we take a random action with probability $1 - \epsilon$, where $\epsilon = 0.5$.

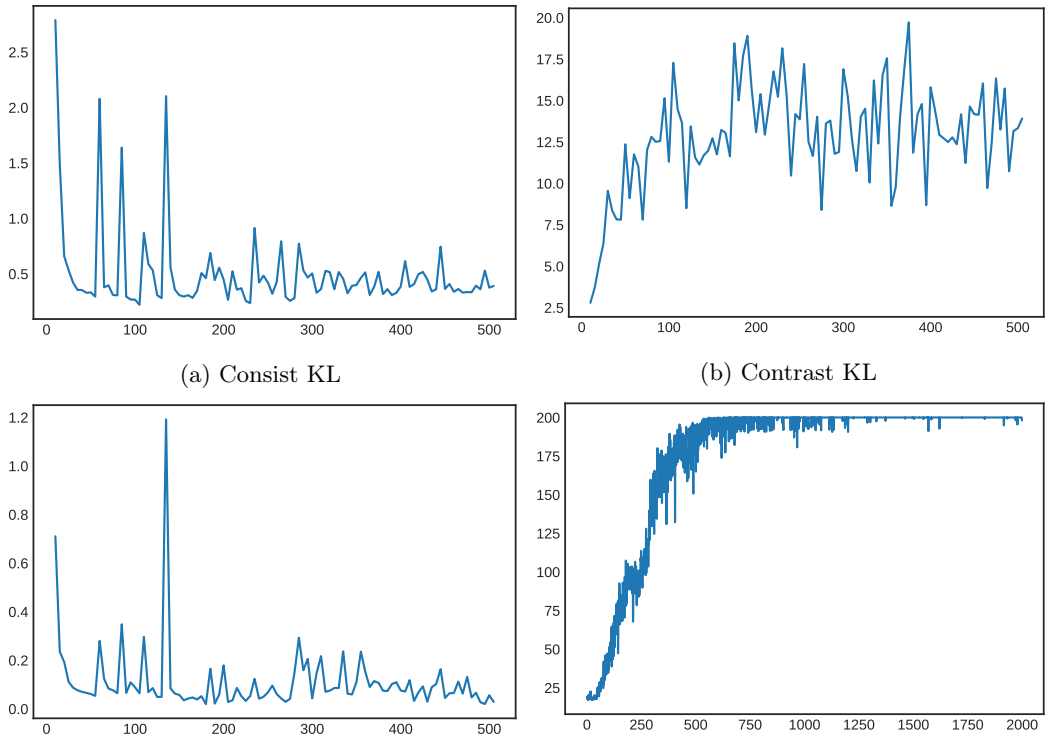
Solipsistic Contrast We use x_t^m to denote the state at time t of the mth trajectory. We approximate the contrast term in equation 3 using the following Monte Carlo approximation

$$\mathbb{E}_{i \neq t+1}\left[\mathrm{KL}\big(p(s_{t+1}^m|x_t^m, a_t^m)||p(s_i|x_i)\big)\right] \approx \mathrm{KL}\big(p(s_{t+1}^m|x_t^m, a_t^m)||p(s_{t+1}^n|x_{t+1}^n)\big)\,, \tag{20}$$

where the nth ($n \neq m$) trajectory is sampled from memory. We found this one-sample Monte Carlo approximation works well in practice.

E.2.1 Training Statistics

In Figure 8 we illustrate the training statistics from a single run of algorithm 1. This is representative of the typical behavior that appears for each run of this procedure from different parameter initializations. We plot the solipsistic consistency (equation 2), solipsistic contrast (equation 3) and reward consistency (section 2) in Figure 8a, Figure 8b and Figure 8c respectively. In Figure 8d, we plot the average of the predicted rewards received during each iteration of policy training, within our overall learning procedure. More specifically, we plot $\frac{1}{J}\sum_{j=1}^{J} E(w^j)$, where the j samples are used during our VO update step (section D).



(a) Consist KL

(b) Contrast KL

(c) Reward Consistent KL

(d) $\frac{1}{J}\sum_{j=1}^{J} E(w^j)$ recorded during policy training

Figure 8: Typical behavior of the training statistics corresponding to the reward evaluations reported in Figure 6.

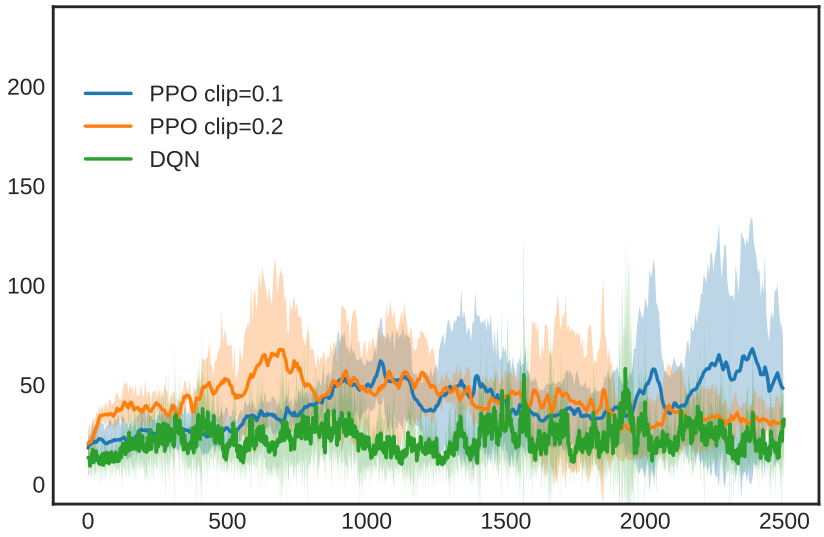


Figure 9: Evaluating the baseline policies trained using 2500 trajectories. We plot the average over 5 runs using different random seeds for our parameter initialization. We smooth the curves using a moving average with window size 3. We find the curves for PPO have high variance, meaning that PPO can occasionally get reasonable performance for this number of trajectories (e.g. balancing the pole for over 100 time steps successfully) but is not very unstable during training and across across different random seeds.

E.3 Baseline Model-Free Methods

E.3.1 PPO

We implemented the PPO method (Schulman et al., 2017) as a baseline to evaluate our methods. We run PPO using two different clip ratios 0.1 and 0.2. The policy and the value networks share similar architecture to our recognition convolutional neural network described in section E.2 with the only difference being the dimensionality of the output layers. For the Cartpole experiment, the output size of the policy network is 2 and we use a softmax activation function to parameterize the probabilities of choosing from two actions. The output of the value network is a linear layer with output dimension 1. For each training episode, we sample 5 trajectories from the true environment with a maximum length of 200. We then train the model for 10 epochs using ADAM where we take the hyper-parameters provided by the OpenAI Baselines implementation Dhariwal et al. (2017), where learning is 3^{-4}, $\gamma = 0.99$, $\lambda = 0.95$, the weight of the value term is 0.5 and the weight of the entropy term is 0.01. We report results for two different clip ratios of 0.1 and 0.2.

E.3.2 DQN

The other model free baseline we compared to is DQN (Mnih et al., 2015). Like for the case of PPO, we keep the architecture of the Q-network and the target network similar to our recognition network, except for the final output layer. In our Cartpole experiment, the Q-network outputs the Q-value for the two possible actions given the states. The action with the larger Q-value is chosen during control. We train DQN for 2500 episodes (as shown in Figure 9). For each episode, we sample one trajectory with the maximum length of 200. After a limited grid search of hyper-parmeters, we find that the hyper-parameters from the PyTorch DQN tutorial (Paszke, 2020; Paszke et al., 2019).

E.4 Recognition function characteristics

Of interest for a qualitative analysis on the recognition function is whether it has learned to filter out redundant information. To answer this we extract filters from the first layer of the CNN recognition function, which act as attention maps over the pixels. In Figure 10 we see that different filters are attending to different physical attributes, for example pole position, pole speed and cart position. In contrast, no clear physical interpretation was apparent for samples from PPO's convolutional policy and value networks, which we discuss further in appendix E.5.

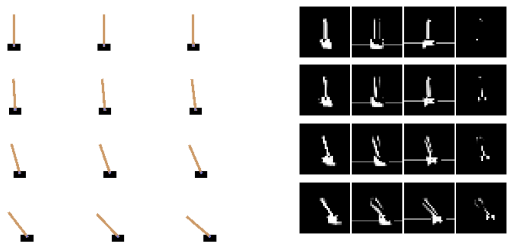


Figure 10: Visualization of the CNN filters' activations. We take action 'push cart to the right' for 15 steps, so that the velocity of the cart and pole are monotonically increasing. The left half of the figure shows four states (i.e. the 4 rows) $\{x_1, x_5, x_{10}, x_{15}\}$ where each state is the stack of 3 successive frames (i.e. the 3 columns) $x_t = \{f_t, f_{t-1}, f_{t-2}\}$. The right half of the figure shows the activations of the CNN's first layer in the recognition network. We select 4 filters' activations (there are 8 filters in the first layer of recognition network) and use a sigmoid function to create these grey-scale images. The first filter appears to encode spatial information of the cart and pole (the activated pixels are consistent with the position of both throughout). The second and third filters we believe represent velocity information for the pole (given the activated pixels can be interpreted as providing a finite difference type estimate of velocity by encoding the position of the pole in the first and last frames), while the fourth can be interpreted as encoding information about the velocity of the cart (given more pixels become activated in the region of the cart the faster it goes).

E.5 Activation Maps of PPO

In this section, we visualize the activations of the first convolution layer of the policy network and the value network for PPO. We plot all 8 filters' activations and use a sigmoid function to create the following grey-scale images. Each column represents a filter. Each row represents the state of a trajectory as the cart is pushed to the right with monotonically increasing velocity (as in Figure 10). In both Figure 11 and Figure 12, none of the activation maps have a change in activation that can clearly be interpreted as relating to the increase in velocity (unlike the case of the solipsistic recognition model illustrated Figure 10); only the positional information of the cart and pole is obviously captured.

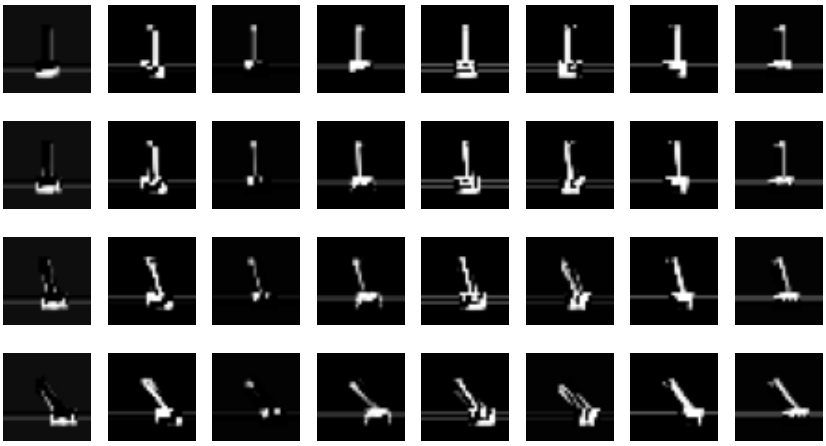


Figure 11: Visualization of the activations in the PPO's policy network.

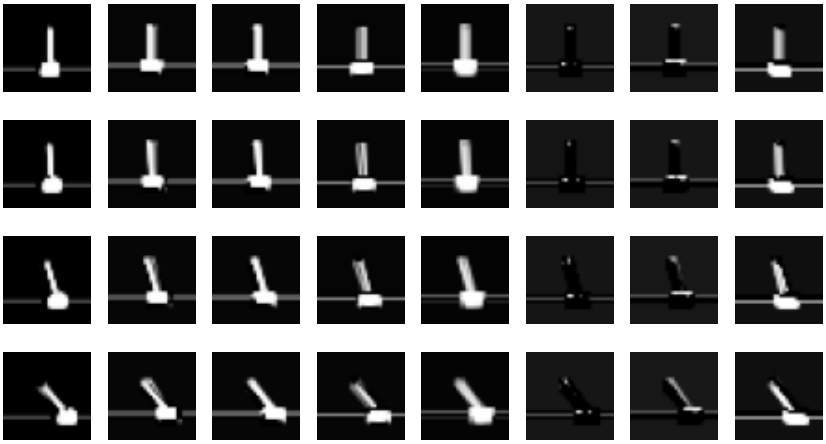


Figure 12: Visualization of the activations in the PPO's value network.