Planning with Goal-Conditioned Policies

Soroush Nasiriany[*], **Vitchyr H. Pong**[*], **Steven Lin**, **Sergey Levine**
University of California, Berkeley
{snasiriany,vitchyr,stevenlin598,svlevine@berkeley.edu}

Abstract

Planning methods can solve temporally extended sequential decision making problems by composing simple behaviors. However, planning requires suitable abstractions for the states and transitions, which typically need to be designed by hand. In contrast, model-free reinforcement learning (RL) can acquire behaviors from low-level inputs directly, but often struggles with temporally extended tasks. Can we utilize reinforcement learning to automatically form the abstractions needed for planning, thus obtaining the best of both approaches? We show that goal-conditioned policies learned with RL can be incorporated into planning, so that a planner can focus on which states to reach, rather than how those states are reached. However, with complex state observations such as images, not all inputs represent valid states. We therefore also propose using a latent variable model to compactly represent the set of valid states for the planner, so that the policies provide an abstraction of actions, and the latent variable model provides an abstraction of states. We compare our method with planning-based and model-free methods and find that our method significantly outperforms prior work when evaluated on image-based robot navigation and manipulation tasks that require non-greedy, multi-staged behavior.

1 Introduction

Reinforcement learning can acquire complex skills by learning through direct interaction with the environment, sidestepping the need for accurate modeling and manual engineering. However, complex and temporally extended sequential decision making requires more than just well-honed reactions. Agents that generalize effectively to new situations and new tasks must reason about the consequences of their actions and solve new problems via planning. Accomplishing this entirely with model-free RL often proves challenging, as purely model-free learning does not inherently provide for temporal compositionality of skills. Planning and trajectory optimization algorithms encode this temporal compositionality by design, but require accurate models with which to plan. When these models are specified manually, planning can be very powerful, but learning such models presents major obstacles: in complex environments with high-dimensional observations such as images, direct prediction of future observations presents a very difficult modeling problem [4, 43, 36, 6, 27, 3, 31], and model errors accumulate over time [39], making their predictions inaccurate in precisely those long-horizon settings where we most need the compositionality of planning methods. Can we obtain the benefits temporal compositionality inherent in model-based planning, without the need to model the environment at the lowest level, in terms of both time and state representation?

One way to avoid modeling the environment in detail is to plan over *abstractions*: simplified representations of states and transitions on which it is easier to construct predictions and plans. *Temporal* abstractions allow planning at a coarser time scale, skipping over the high-frequency details and instead planning over higher-level subgoals, while *state* abstractions allow planning over a

[*]equal contribution

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

simpler representation of the state. Both make modeling and planning easier. In this paper, we study how model-free RL can be used to provide such abstraction for a model-based planner. At first glance, this might seem like a strange proposition, since model-free RL methods learn value functions and policies, not models. However, this is precisely what makes them ideal for abstracting away the complexity in temporally extended tasks with high-dimensional observations: by avoiding low-level (e.g., pixel-level) prediction, model-free RL can acquire behaviors that manipulate these low-level observations without needing to predict them explicitly. This leaves the planner free to operate at a higher level of abstraction, reasoning about the capabilities of low-level model-free policies.

Building on this idea, we propose a *model-free* planning framework. For *temporal* abstraction, we learn low-level goal-conditioned policies, and use their value functions as implicit models, such that the planner plans over the goals to pass to these policies. Goal-conditioned policies are policies that are trained to reach a goal state that is provided as an additional input [24, 55, 53, 48]. While in principle such policies can solve any goal-reaching problem, in practice their effectiveness is constrained to nearby goals: for long-distance goals that require planning, they tend to be substantially less effective, as we illustrate in our experiments. However, when these policies are trained together with a value function, as in an actor-critic algorithms, the value function can provide an indication of whether a particular goal is reachable or not. The planner can then plan over intermediate subgoals, using the goal-conditioned value function to evaluate reachability. A major challenge with this setup is the need to actually optimize over these subgoals. In domains with high-dimensional observations such as images, this may require explicitly optimizing over image pixels. This optimization is challenging, as realistic images – and, in general, feasible states – typically form a thin, low-dimensional manifold within the larger space of possible state observation values [34]. To address this, we also build abstractions of the state observation by learning a compact latent variable state representation, which makes it feasible to optimize over the goals in domains with high-dimensional observations, such as images, without explicitly optimizing over image pixels. The learned representation allows the planner to determine which subgoals actually represent feasible states, while the learned goal-conditioned value function tells the planner whether these states are reachable.

Our contribution is a method for combining model-free RL for short-horizon goal-reaching with model-based planning over a latent variable representation of subgoals. We evaluate our method on temporally extended tasks that require multistage reasoning and handling image observations. The low-level goal-reaching policies themselves cannot solve these tasks effectively, as they do not plan over subgoals and therefore do not benefit from temporal compositionality. Planning without state representation learning also fails to perform these tasks, as optimizing directly over images results in invalid subgoals. By contrast, our method, which we call Latent Embeddings for Abstracted Planning (LEAP), is able to successfully determine suitable subgoals by searching in the latent representation space, and then reach these subgoals via the model-free policy.

2 Related Work

Goal-conditioned reinforcement learning has been studied in a number of prior works [24, 25, 37, 18, 53, 2, 48, 57, 40, 59]. While goal-conditioned methods excel at training policies to greedily reach goals, they often fail to solve long-horizon problems. Rather than proposing a new goal-conditioned RL method, we propose to use goal-conditioned policies as the abstraction for planning in order to handle tasks with a longer horizon.

Model-based planning in deep reinforcement learning is a well-studied problem in the context of low-dimensional state spaces [50, 32, 39, 7]. When the observations are high-dimensional, such as images, model errors for direct prediction compound quickly, making model-based RL difficult [15, 13, 5, 14, 26]. Rather than planning directly over image observations, we propose to plan at a temporally-abstract level by utilizing goal-conditioned policies. A number of papers have studied embedding high-dimensional observations into a low-dimensional latent space for planning [60, 16, 62, 22, 29]. While our method also plans in a latent space, we additionally use a model-free goal-conditioned policy as the abstraction to plan over, allowing our method to plan over temporal abstractions rather than only state abstractions.

Automatically setting subgoals for a low-level goal-reaching policy bears a resemblance to hierarchical RL, where prior methods have used model-free learning on top of goal-conditioned policies [10, 61,

12, 58, 33, 20, 38]. By instead using a planner at the higher level, our method can flexibly plan to solve new tasks and benefit from the compositional structure of planning.

Our method builds on temporal difference models [48] (TDMs), which are finite-horizon, goal-conditioned value functions. In prior work, TDMs were used together with a single-step planner that optimized over a single goal, represented as a low-dimensional ground truth state (under the assumption that all states are valid) [48]. We also use TDMs as implicit models, but in contrast to prior work, we plan over multiple subgoals and demonstrate that our method can perform temporally extended tasks. More critically, our method also learns abstractions of the state, which makes this planning process much more practical, as it does not require assuming that all state vectors represent feasible states. Planning with goal-conditioned value functions has also been studied when there are a discrete number of predetermined goals [30] or skills [1], in which case graph-search algorithms can be used to plan. In this paper, we not only provide a concrete instantiation of planning with goal-conditioned value functions, but we also present a new method for scaling this planning approach to images, which reside in a lower-dimensional manifold.

Lastly, we note that while a number of papers have studied how to combine model-free and model-based methods [54, 41, 23, 56, 44, 51, 39], our method is substantially different from these approaches: we study how to use model-free policies *as the abstraction for planning*, rather than using models [54, 41, 23, 39] or planning-inspired architectures [56, 44, 51, 21] to accelerate model-free learning.

3 Background

We consider a finite-horizon, goal-conditioned Markov decision process (MDP) defined by a tuple $(\mathcal{S}, \mathcal{G}, \mathcal{A}, p, R, T_{\max}, \rho_0, \rho_g)$, where $\mathcal{S}$ is the set of states, $\mathcal{G}$ is the set of goals, $\mathcal{A}$ is the set of actions, $p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$ is the time-invariant (unknown) dynamics function, R is the reward function, $T_{\max}$ is the maximum horizon, ρ_0 is the initial state distribution, and ρ_g is the goal distribution. The objective in goal-conditioned RL is to obtain a policy $\pi(\mathbf{a}_t \mid \mathbf{s}_t, \mathbf{g}, t)$ to maximize the expected sum of rewards $\mathbb{E}[\sum_{t=0}^{T_{\max}} R(\mathbf{s}_t, \mathbf{g}, t)]$, where the goal is sampled from ρ_g and the states are sampled according to $\mathbf{s}_0 \sim \rho_0$, $\mathbf{a}_t \sim \pi(\mathbf{a}_t \mid \mathbf{s}_t, \mathbf{g}, t)$, and $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$. We consider the case where goals reside in the same space as states, i.e., $\mathcal{G} = \mathcal{S}$.

An important quantity in goal-conditioned MDPs is the goal-conditioned value function V^π, which predicts the expected sum of future rewards, given the current state $\mathbf{s}$, goal $\mathbf{g}$, and time t:

$$V^\pi(\mathbf{s}, \mathbf{g}, t) = \mathbb{E}\left[\sum_{t'=t}^{T_{\max}} R(\mathbf{s}_{t'}, \mathbf{g}, t') \mid \mathbf{s}_t = \mathbf{s}, \pi \text{ is conditioned on } \mathbf{g}\right].$$

To keep the notation uncluttered, we will omit the dependence of V on π. While various time-varying reward functions can be used, temporal difference models (TDMs) [48] use the following form:

$$R_{\text{TDM}}(\mathbf{s}, \mathbf{g}, t) = -\delta(t = T_{\max}) d(\mathbf{s}, \mathbf{g}). \tag{1}$$

where δ is the indicator function, and the distance function d is defined by the task. This particular choice of reward function gives a TDM the following interpretation: given a state $\mathbf{s}$, how close will the goal-conditioned policy π get to $\mathbf{g}$ after t time steps of attempting to reach $\mathbf{g}$? TDMs can thus be used as a measure of reachability by quantifying how close to another state the policy can get in t time steps, thus providing *temporal* abstraction. However, TDMs will only produce reasonable reachability predictions for *valid* goals – goals that resemble the kinds of states on which the TDM was trained. This important limitation requires us to also utilize *state* abstractions, limiting our search to valid states. In the next section, we will discuss how we can use TDMs in a planning framework over high-dimensional state observations such as images.

4 Planning with Goal-Conditioned Policies

We aim to learn a model that can solve arbitrary long-horizon goal reaching tasks with high-dimensional observation and goal spaces, such as images. A model-free goal-conditioned reinforcement learning algorithm could, in principle, solve such a problem. However, as we will show in our experiments, in practice such methods produce overly greedy policies, which can accomplish short-term goals, but struggle with goals that are more temporally extended. We instead combine

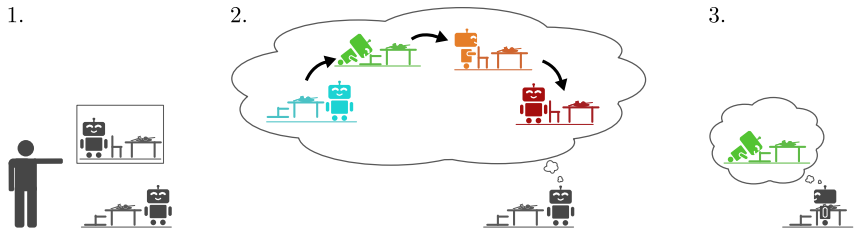

Figure 1: Summary of Latent Embeddings for Abstracted Planning (LEAP). (1) The planner is given a goal state. (2) The planner plans intermediate subgoals in a low-dimensional latent space. By planning in this latent space, the subgoals correspond to valid state observations. (3) The goal-conditioned policy then tries to reach the first subgoal. After t_1 time steps, the policy replans and repeats steps 2 and 3.

goal-conditioned policies trained to achieve subgoals with a planner that decomposes long-horizon goal-reaching tasks into K shorter horizon subgoals. Specifically, our planner chooses the K subgoals, $\mathbf{g}_{t_1}, \ldots, \mathbf{g}_{t_K}$, and a goal-reaching policy then attempts to reach the first subgoal $\mathbf{g}_{t_1}$ in the first t_1 time steps, before moving onto the second goal $\mathbf{g}_{t_2}$, and so forth, as shown in Figure 1. This procedure only requires training a goal-conditioned policy to solve short-horizon tasks. Moreover, by planning appropriate subgoals, the agent can compose previously learned goal-reaching behavior to solve new, temporally extended tasks. The success of this approach will depend heavily on the choice of subgoals. In the sections below, we outline how one can measure the quality of the subgoals. Then, we address issues that arise when optimizing over these subgoals in high-dimensional state spaces such as images. Lastly, we summarize the overall method and provide details on our implementation.

4.1 Planning over Subgoals

Suitable subgoals are ones that are reachable: if the planner can choose subgoals such that each subsequent subgoal is reachable given the previous subgoal, then it can reach any goal by ensuring the last subgoal is the true goal. If we use a goal-conditioned policy to reach these goals, how can we quantify how reachable these subgoals are?

One natural choice is to use a goal-conditioned value function which, as previously discussed, provides a measure of reachability. In particular, given the current state $\mathbf{s}$, a policy will reach a goal $\mathbf{g}$ after t time steps if and only if $V(\mathbf{s}, \mathbf{g}, t) = 0$. More generally, given K intermediate subgoals $\mathbf{g}_{1:K} = \mathbf{g}_1, \ldots, \mathbf{g}_K$ and $K+1$ time intervals $t_1, \ldots, t_{K+1}$ that sum to $T_{\max}$, we define the *feasibility vector* as

$$\overrightarrow{\mathbf{V}}(\mathbf{s}, \mathbf{g}_{1:k}, t_{1:k}, \mathbf{g}) = \begin{bmatrix} V(\mathbf{s}, \mathbf{g}_1, t_1) \\ V(\mathbf{g}_1, \mathbf{g}_2, t_2) \\ \vdots \\ V(\mathbf{g}_{K-1}, \mathbf{g}_K, t_K) \\ V(\mathbf{g}_K, \mathbf{g}, t_{K+1}) \end{bmatrix}.$$

The feasibility vector provides a quantative measure of a plan's feasibility: The first element describes how close the policy will reach the first subgoal, $\mathbf{g}_1$, starting from the initial state, $\mathbf{s}$. The second element describes how close the policy will reach the second subgoal, $\mathbf{g}_2$, starting from the first subgoal, and so on, until the last term measures the reachability to the true goal, $\mathbf{g}$.

To create a feasible plan, we would like each element of this vector to be zero, and so we minimize the norm of the feasibility vector:

$$\mathcal{L}(\mathbf{g}_{1:K+1}) = ||\overrightarrow{\mathbf{V}}(\mathbf{s}, \mathbf{g}_{1:k}, t_{1:k}, \mathbf{g})||. \tag{2}$$

In other words, minimizing Equation 2 searches for subgoals such that the overall path is feasible and terminates at the true goal. In the next section, we turn to optimizing Equation 2 and address issues that arise in high-dimensional state spaces.

4.2 Optimizing over Images

We consider image-based environments, where the set of states $\mathcal{S}$ is the set of valid image observations in our domain. In image-based environments, solving the optimization in Equation 2 presents two

problems. First, the optimization variables $\mathbf{g}_{1:K}$ are very high-dimensional – even with 64x64 images and just 3 subgoals, there are over 10,000 dimensions. Second, and perhaps more subtle, the optimization iterates must be constrained to the set of valid image observations $\mathcal{S}$ for the subgoals to correspond to meaningful states. While a plethora of constrained optimization methods exist, they typically require knowing the set of valid states [42] or being able to project onto that set [46]. In image-based domains, the set of states $\mathcal{S}$ is an unknown r-dimensional manifold embedded in a higher-dimensional space $\mathbb{R}^N$, for some $N \gg r$ [34] – i.e., the set of valid image observations.

Optimizing Equation 2 would be much easier if we could directly optimize over the r dimensions of the underlying representation, since $r \ll N$, and crucially, since we would not have to worry about constraining the planner to an unknown manifold. While we may not know the set $\mathcal{S}$ a priori, we can learn a latent-variable model with a compact latent space to capture it, and then optimize in the latent space of this model. To this end, we use a variational-autoencoder (VAE) [28, 52], which we train with images randomly sampled from our environment.

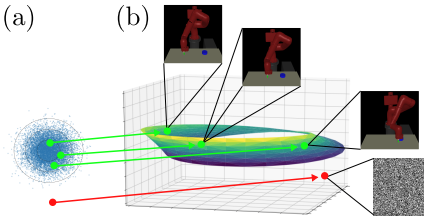

Figure 2: Optimizing directly over the image manifold (b) is challenging, as it is generally unknown and resides in a high-dimensional space. We optimize over a latent state (a) and use our decoder to generate images. So long as the latent states have high likelihood under the prior (green), they will correspond to realistic images, while latent states with low likelihood (red) will not.

A VAE consists of an encoder $q_\phi(\mathbf{z} \mid \mathbf{s})$ and decoder $p_\theta(\mathbf{s} \mid \mathbf{z})$. The inference network maps high-dimensional states $\mathbf{s} \in \mathcal{S}$ to a distribution over lower-dimensional latent variables $\mathbf{z}$ for some lower dimensional space $\mathcal{Z}$, while the generative model reverses this mapping. Moreover, the VAE is trained so that the marginal distribution of $\mathcal{Z}$ matches our prior distribution p_0, the standard Gaussian.

This last property of VAEs is crucial, as it allows us to tractably optimize over the manifold of valid states $\mathcal{S}$. So long as the latent variables have high likelihood under the prior, the corresponding images will remain inside the manifold of valid states, as shown in Figure 2. In fact, Dai and Wipf [9] showed that a VAE with a Gaussian prior can always recover the true manifold, making this choice for latent-variable model particularly appealing.

In summary, rather than minimizing Equation 2, which requires optimizing over the high-dimensional, unknown space $\mathcal{S}$ we minimize

$$\mathcal{L}_{\text{LEAP}}(\mathbf{z}_{1:K}) = ||\overrightarrow{\mathbf{V}}(\mathbf{s}, \mathbf{z}_{1:k}, t_{1:k}, \mathbf{g})||_p - \lambda \sum_{k=1}^{K} \log p(\mathbf{z}_k) \tag{3}$$

where

$$\overrightarrow{\mathbf{V}}(\mathbf{s}, \mathbf{z}_{1:k}, t_{1:k}, \mathbf{g}) = \begin{bmatrix} V(\mathbf{s}, \psi(\mathbf{z}_1), t_1) \\ V(\psi(\mathbf{z}_1), \psi(\mathbf{z}_2), t_2) \\ \vdots \\ V(\psi(\mathbf{z}_{K-1}), \psi(\mathbf{z}_K), t_K) \\ V(\psi(\mathbf{z}_K), \mathbf{g}, t_{K+1}) \end{bmatrix} \quad \text{and} \quad \psi(\mathbf{z}) = \arg\max_{\mathbf{g}'} p_\theta(\mathbf{g}' \mid \mathbf{z}).$$

This procedure optimizes over latent variables $\mathbf{z}_{t_k}$, which are then mapped onto high-dimensional goal states $\mathbf{g}_{t_k}$ using the maximum likelihood estimate (MLE) of the decoder $\arg\max_{\mathbf{g}}(\mathbf{g} \mid \mathbf{z})$. In our case, the MLE can be computed in closed form by taking the mean of the decoder. The term summing over $\log p(\mathbf{z}_{t_k})$ penalizes latent variables that have low likelihood under the prior p, and λ is a hyperparameter that controls the importance of this second term.

While any norm could be used, we used the ℓ_∞-norm which forces each element of the feasibility vector to be near zero. We found that the ℓ_∞-norm outperformed the ℓ_2-norm, which only forces the sum of squared elements near zero. [2]

4.3 Goal-Conditioned Reinforcement Learning

For our goal-conditioned reinforcement learning algorithm, we use temporal difference models (TDMs) [48]. TDMs learn Q functions rather that V functions, and so we compute V by evaluating

[2] See Subsection A.1 comparison.

Q with the action from the deterministic policy: $V(\mathbf{s}, \mathbf{g}, t) = Q(\mathbf{s}, \mathbf{a}, \mathbf{g}, t)|_{\mathbf{a}=\pi(\mathbf{s},\mathbf{g},t)}$. To further improve the efficiency of our method, we can also utilize the same VAE that we use to recover the latent space for planning as a state representation for TDMs. While we could train the reinforcement learning agents from scratch, this can be expensive in terms of sample efficiency as much of the learning will focus on simply learning good convolution filters. We therefore use the pretrained mean-encoder of the VAE as the state encoder for our policy and value function networks, and only train additional fully-connected layers with RL on top of these representations. Details of the architecture are provided in Appendix C. We show in Section 5 that our method works without reusing the VAE mean-encoder, and that this parameter reuse primarily helps with increasing the speed of learning.

4.4 Summary of Latent Embeddings for Abstracted Planning

Our overall method is called Latent Embeddings for Abstracted Planning (LEAP) and is summarized in Algorithm 1. We first train a goal-conditioned policy and a variational-autoencoder on randomly collected states. Then, given a new goal, we choose subgoals by minimizing Equation 3. Once the plan is chosen, the first goal $\psi(z_1)$ is given to the policy. After t_1 steps, we repeat this procedure: we produce a plan with $K-1$ (rather than K) subgoals, and give the first goal to the policy. In this work, we fix the time intervals to be evenly spaced out (i.e., $t_1 = t_2 \dots t_{K+1} = \lfloor T_{\max}/(K+1) \rfloor$), but additionally optimizing over the time intervals would be a promising future extension.

Algorithm 1 Latent Embeddings for Abstracted Planning (LEAP)

Train VAE encoder q_ϕ and decoder p_θ.
Train TDM policy π and value function V.
Initialize state, goal, and time: $\mathbf{s}_1 \sim \rho_0$, goal $\mathbf{g} \sim \rho_g$, and $t = 1$.
Assign the last subgoal to the true goal, $\mathbf{g}_{K+1} = \mathbf{g}$
for k in $1, \dots, K+1$ **do**
    Optimize Equation 3 to choose latent subgoals $\mathbf{z}_k, \dots, \mathbf{z}_K$ using V and p_θ if $k \leq K$.
    Decode $\mathbf{z}_k$ to obtain goal $\mathbf{g}_k = \psi(\mathbf{z}_k)$.
    for t' in $1, \dots, t_k$ **do**
        Sample next state $\mathbf{s}_{t+1}$ using goal-conditioned policy $\pi(\cdot \mid \mathbf{s}_t, \mathbf{g}_k, t_k - t')$.
        Increment the global timer $t \leftarrow t + 1$.
    end for
end for

5 Experiments

Our experiments study the following two questions: **(1)** How does LEAP compare to model-based methods, which directly predict each time step, and model-free RL, which directly optimizes for the final goal? **(2)** How does the use of a latent state representation and other design decisions impact the performance of LEAP?

5.1 Vision-based Comparison and Results

We study the first question on two distinct vision-based tasks, each of which requires temporally-extended planning and handling high-dimensional image observations.

The first task, *2D Navigation* requires navigating around a U-shaped wall to reach a goal, as shown in Figure 3. The state observation is a top-down image of the environment. We use this task to conduct ablation studies that test how each component of LEAP contributes to final performance. We also use this environment to generate visualizations that help us better understand how our method uses the goal-conditioned value function to evaluate reachability over images. While visually simple, this task is far from trivial for goal-conditioned and planning methods: a greedy goal-reaching policy that moves directly towards the goal will never reach the goal. The agent must plan a temporally-extended path that moves around the walls, sometimes moving away from the goal. We also use this environment to compare our method with prior work on goal-conditioned and model-based RL.

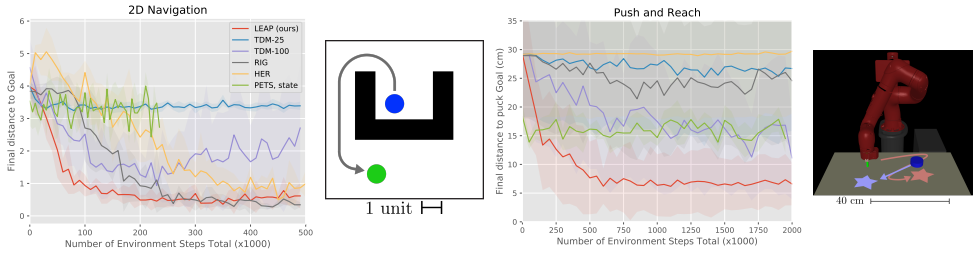

Figure 3: Comparisons on two vision-based domains that evaluate temporally extended control, with illustrations of the tasks. In 2D Navigation (left), the goal is to navigate around a U-shaped wall to reach the goal. In the Push and Reach manipulation task (right), a robot must first push a puck to a target location (blue star), which may require moving the hand away from the goal hand location, and then move the hand to another location (red star). Curves are averaged over multiple seeds and shaded regions represent one standard deviation. Our method, shown in red, outperforms prior methods on both tasks. On the Push and Reach task, prior methods typically get the hand close to the right location, but perform much worse at moving the puck, indicating an overly greedy strategy, while our approach succeeds at both.

To evaluate LEAP on a more complex task, we utilize a robotic manipulation simulation of a *Push and Reach* task. This task requires controlling a simulated Sawyer robot to both (1) move a puck to a target location and (2) move its end effector to a target location. This task is more visually complex, and requires more temporally extended reasoning. The initial arm and and puck locations are randomized so that the agent must decide how to reposition the arm to reach around the object, push the object in the desired direction, and then move the arm to the correct location, as shown in Figure 3. A common failure case for model-free policies in this setting is to adopt an overly greedy strategy, only moving the arm to the goal while ignoring the puck.

We train all methods on randomly initialized goals and initial states. However, for evaluation, we intentionally select difficult start and goal states to evaluate long-horizon reasoning. For 2D Navigation, we initialize the policy randomly inside the center square and sample a goal from the region directly below the U-shaped wall. This requires initially moving away from the goal to navigate around the wall. For Push and Reach, we evaluate on 5 distinct challenging configurations, each requiring the agent to first plan to move the puck, and then move the arm only once the puck is in its desired location. In one configuration for example, we initialize the hand and puck on opposite sides of the workspace and set goals so that the hand and puck must switch sides.

We compare our method to both model-free methods and model-based methods that plan over learned models. All of our tasks use $T_{\max} = 100$, and LEAP uses CEM to optimize over $K = 3$ subgoals, each of which are 25 time steps apart. We compare directly with model-free TDMs, which we label **TDM-25**. Since the task is evaluated on a horizon of length $T_{\max} = 100$ we also compare to a model-free TDM policy trained for $T_{\max} = 100$, which we label **TDM-100**. We compare to reinforcement learning with imagined goals (**RIG**) [40], a state-of-the-art method for solving image-based goal-conditioned tasks. RIG learns a reward function from images rather than using a pre-determined reward function. We found that providing RIG with the same distance function as our method improves its performance, so we use this stronger variant of RIG to ensure a fair comparison. In addition, we compare to hindsight experiment replay (**HER**) [2] which uses sparse, indicator rewards. Lastly, we compare to probabilistic ensembles with trajectory sampling (PETS) [7], a state-of-the-art model-based RL method. We favorably implemented PETS on the ground-truth low-dimensional state representation and label it **PETS, state**.

The results are shown in Figure 3. LEAP significantly outperforms prior work on both tasks, particularly on the harder Push and Reach task. While the TDM used by LEAP (TDM-25) performs poorly by itself, composing it with 3 different subgoals using LEAP results in much better performance. By 400k environment steps, LEAP already achieves a final puck distance of under 10 cm, while the next best method, TDM-100, requires 5 times as many samples. Details on each task are in Appendix B, and algorithm implementation details are given in Appendix C.

We visualize the subgoals chosen by LEAP in Figure 4 by decoding the latent subgoals $\mathbf{z}_{t_{1:K}}$ into images with the VAE decoder p_θ. In Push and Reach, these images correspond to natural subgoals for the task. Figure 4 also shows a visualization of the value function, which is used by the planner to determine reachability. Note that the value function generally recognizes that the wall is impassable,

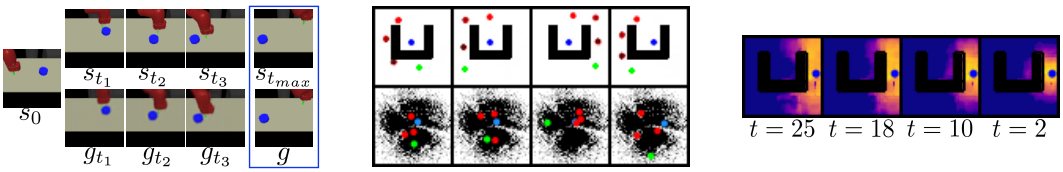

Figure 4: (Left) Visualization of subgoals reconstructed from the VAE (bottom row), and the actual images seen when reaching those subgoals (top row). Given an initial state s_0 and a goal image $\mathbf{g}$, the planner chooses meaningful subgoals: at $\mathbf{g}_{t_1}$, it moves towards the puck, at $\mathbf{g}_{t_2}$ it begins pushing the puck, and at $\mathbf{g}_{t_3}$ it completes the pushing motion before moving to the goal hand position at $\mathbf{g}$. (Middle) The top row shows the image subgoals superimposed on one another. The blue circle is the starting position, the green circle is the target position, and the intermediate circles show the progression of subgoals (bright red is $\mathbf{g}_{t_1}$, brown is $\mathbf{g}_{t_3}$). The colored circles show the subgoals in the latent space (bottom row) for the two most active VAE latent dimensions, as well as samples from the VAE aggregate posterior [35]. (Right) Heatmap of the value function $V(\mathbf{s}, \mathbf{g}, t)$, with each column showing a different time horizon t for a fixed state $\mathbf{s}$. Warmer colors show higher value. Each image indicates the value function for all possible goals g. As the time horizon decreases, the value function recognizes that it can only reach nearby goals.

and makes reasonable predictions for different time horizons. Videos of the final policies and generated subgoals and code for our implementation of LEAP are available on the paper website[3].

5.2 Planning in Non-Vision-based Environments with Unknown State Spaces

While LEAP was presented in the context of optimizing over images, we also study its utility in non-vision based domains. Specifically, we compare LEAP to prior works on an *Ant Navigation* task, shown in Figure 5, where the state-space consists of the quadruped robot's joint angles, joint velocity, and center of mass. While this state space is more compact than images, only certain combinations of state values are actually valid, and the obstacle in the environment is unknown to the agent, meaning that a naïve optimization over the state space can easily result in invalid states (e.g., putting the robot inside an obstacle).

This task has a significantly longer horizon of $T_{\max} = 600$, and LEAP uses CEM to optimize over $K = 11$ subgoals, each of which are 50 time steps apart. As in the vision-based comparisons, we compare with model-free TDMs, for the short-horizon setting (**TDM-50**) which LEAP is built on top of, and the long-horizon setting (**TDM-600**). In addition to **HER**, we compare to a variant of HER that uses the same rewards and relabeling strategy as RIG, which we label **HER+**. We exclude the PETS baseline, as it has been unable to solve long-horizon tasks such as ours. In this section, we add a comparison to hierarchical reinforcement learning with off-policy correction (**HIRO**) [38], a hierarchical method for state-based goals. We evaluate all baselines on a challenging configuration of the task in which the ant must navigate from the top left corner to the top right corner of the maze, by going around a long wall. The desired behavior will result in large negative rewards during the trajectory, but will result in an optimal final state. We see that in Figure 5, LEAP is the only method that successfully navigates the ant to the goal. HIRO, HER, HER+ don't attempt to go around the wall at all, as doing so will result in a large sum of negative rewards. TDM-50 has a horizon that is too short and results in greedy behavior, while TDM-600 fails to learn meaningful behavior due to the temporal sparsity of the reward.

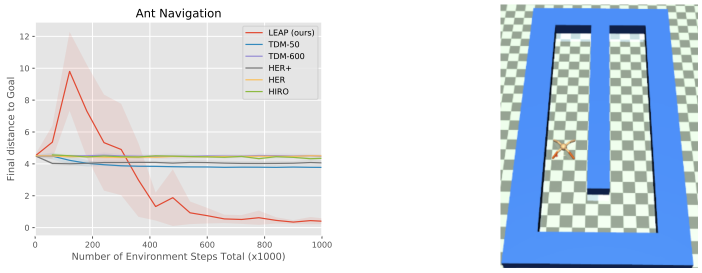

Figure 5: In the Ant Navigation task, the ant must move around the long obstacle, which will result in large negative rewards during the trajectory, but will result in an optimal final state. Our method, shown in red, is the only method that successfully navigates the ant to the goal.

[3] `https://sites.google.com/view/goal-planning`

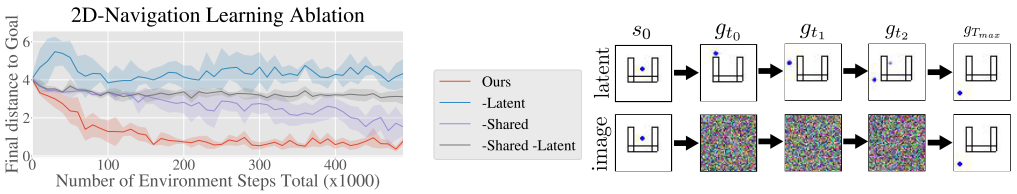

Figure 6: (Left) Ablative studies on 2D Navigation. We keep all components of LEAP the same but replace optimizing over the latent space with optimizing over the image space (-latent). We separately train the RL methods from scratch rather than reusing the VAE mean encoder (-shared), and also test both ablations together (-latent, shared). We see that sharing the encoder weights with the RL policy results in faster learning, and that optimizing over the latent space is critical for success of the method. (Right) Visualization of the subgoals generated when optimizing over the latent space and decoding the image (top) and when optimizing over the images directly (bottom). The goals generated when planning in image space are not meaningful, which explains the poor performance of "-latent" shown in (Left).

5.3 Ablation Study

We analyze the importance of planning in the latent space, as opposed to image space, on the navigation task. For comparison, we implement a planner that directly optimizes over image subgoals (i.e., in pixel space). We also study the importance of reusing the pretrained VAE encoder by replicating the experiments with the RL networks trained from scratch. We see in Figure 6 that a model that does not reuse the VAE encoder does succeed, but takes much longer. More importantly, planning over latent states achieves dramatically better performance than planning over raw images. Figure 6 also shows the intermediate subgoals outputted by our optimizer when optimizing over images. While these subgoals may have high value according to Equation 2, they clearly do not correspond to valid state observations, indicating that the planner is exploiting the value function by choosing images far outside the manifold of valid states.

We include further ablations in Appendix A, in which we study the sensitivity of λ in Equation 3 (Subsection A.3), the choice of norm (Subsection A.1), and the choice of optimizer (Subsection A.2). The results show that LEAP works well for a wide range of λ, that ℓ_∞-norm performs better, and that CEM consistently outperforms gradient-based optimizers, both in terms of optimizer loss and policy performance.

6 Discussion

We presented Latent Embeddings for Abstracted Planning (LEAP), an approach for solving temporally extended tasks with high-dimensional state observations, such as images. The key idea in LEAP is to form *temporal* abstractions by using goal-reaching policies to evaluate reachability, and *state* abstractions by using representation learning to provide a convenient state representation for planning. By planning over states in a learned latent space and using these planned states as subgoals for goal-conditioned policies, LEAP can solve tasks that are difficult to solve with conventional model-free goal-reaching policies, while avoiding the challenges of modeling low-level observations associated with fully model-based methods. More generally, the combination of model-free RL with planning is an exciting research direction that holds the potential to make RL methods more flexible, capable, and broadly applicable. Our method represents a step in this direction, though many crucial questions remain to be answered. Our work largely neglects the question of exploration for goal-conditioned policies, and though this question has been studied in some recent works [17, 45, 59, 49], examining how exploration interacts with planning is an exciting future direction. Another exciting direction for future work is to study how lossy state abstractions might further improve the performance of the planner, by explicitly discarding state information that is irrelevant for higher-level planning.

7 Acknowledgments

This work was supported by the Office of Naval Research, the National Science Foundation, Google, NVIDIA, Amazon, and ARL DCIST CRA W911NF-17-2-0181.

References

[1] Arpit Agarwal, Katharina Muelling, and Katerina Fragkiadaki. Model learning for look-ahead exploration in continuous control. *AAAI*, 2019.

[2] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob Mcgrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, 2017.

[3] Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H Campbell, and Sergey Levine. Stochastic variational video prediction. In *International Conference on Learning Representations*, 2018.

[4] Byron Boots, Arunkumar Byravan, and Dieter Fox. Learning predictive models of a depth camera & manipulator from raw execution traces. In *IEEE International Conference on Robotics and Automation*, 2014.

[5] Arunkumar Byravan, Felix Leeb, Franziska Meier, and Dieter Fox. Se3-pose-nets: structured deep dynamics models for visuomotor planning and control. In *IEEE International Conference on Robotics and Automation*.

[6] Silvia Chiappa, Sébastien Racaniere, Daan Wierstra, and Shakir Mohamed. Recurrent environment simulators. In *International Conference on Learning Representations*, 2017.

[7] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, 2018.

[8] Cédric Colas, Pierre Fournier, Olivier Sigaud, and Pierre-Yves Oudeyer. CURIOUS: intrinsically motivated multi-task, multi-goal reinforcement learning. *International Conference on Machine Learning*, 2019.

[9] Bin Dai and David Wipf. Diagnosing and enhancing vae models. In *International Conference on Learning Representations*, 2019.

[10] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems*, 1993.

[11] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1), 2005.

[12] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13, 2000.

[13] Frederik Ebert, Chelsea Finn, Alex X Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. In *Conference on Robot Learning*, 2017.

[14] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

[15] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *Advances in Neural Information Processing Systems*, 2016.

[16] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *IEEE International Conference on Robotics and Automation*, 2016.

[17] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International Conference on Machine Learning*, 2018.

[18] David Foster and Peter Dayan. Structure in the space of value functions. *Machine Learning*, 49 (2-3), 2002.

[19] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, 2018.

[20] Dibya Ghosh, Abhishek Gupta, and Sergey Levine. Learning actionable representations with goal-conditioned policies. In *International Conference on Learning Representations*, 2019.

[21] Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sébastien Racanière, Théophane Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, et al. An investigation of model-free planning. In *International Conference on Machine Learning*, 2019.

[22] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, 2019.

[23] Nicolas Heess, Greg Wayne, David Silver, Timothy Lillicrap, Yuval Tassa, and Tom Erez. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, 2015.

[24] Leslie Pack Kaelbling. Learning to achieve goals. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume vol.2, 1993.

[25] Leslie Pack Kaelbling. Hierarchical learning in stochastic domains: preliminary results. In *International Conference on Machine Learning*, 1993.

[26] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[27] Nal Kalchbrenner, Aäron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Video pixel networks. In *International Conference on Machine Learning*, 2017.

[28] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

[29] Thanard Kurutach, Aviv Tamar, Ge Yang, Stuart J Russell, and Pieter Abbeel. Learning plannable representations with causal infogan. In *Advances in Neural Information Processing Systems*, 2018.

[30] Terran Lane and Leslie Pack Kaelbling. Toward hierarchical decomposition for planning in uncertain environments. In *Proceedings of the 2001 IJCAI workshop on planning under uncertainty and incomplete information*, 2001.

[31] Alex X Lee, Richard Zhang, Frederik Ebert, Pieter Abbeel, Chelsea Finn, and Sergey Levine. Stochastic adversarial video prediction. *arXiv preprint arXiv:1804.01523*, 2018.

[32] Ian Lenz, Ross Knepper, and Ashutosh Saxena. DeepMPC: learning deep latent features for model predictive control. In *Robotics: Science and Systems (RSS)*, 2015.

[33] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. In *International Conference on Learning Representations*, 2019.

[34] Haw-Minn Lu, Yeshaiahu Fainman, and Robert Hecht-Nielsen. Image manifolds. In *Applications of Artificial Neural Networks in Image Processing III*, volume 3307. International Society for Optics and Photonics, 1998.

[35] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. In *International Conference on Learning Representations*, 2016.

[36] Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. In *International Conference on Learning Representations*, 2016.

[37] Andrew W Moore, Leemon Baird, and Leslie P Kaelbling. Multi-value-functions: Effcient automatic action hierarchies for multiple goal mdps. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1999.

[38] Ofir Nachum, Google Brain, Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, 2018.

[39] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *IEEE International Conference on Robotics and Automation*, 2018.

[40] Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems*, 2018.

[41] Derrick H Nguyen and Bernard Widrow. Neural networks for self-learning control systems. *IEEE Control systems magazine*, 1990.

[42] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[43] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems*, 2015.

[44] Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In *Advances in Neural Information Processing Systems*, 2017.

[45] Fabio Pardo, Vitaly Levdik, and Petar Kormushev. Q-map: a convolutional approach for goal-oriented reinforcement learning. *CoRR*, abs/1810.02927, 2018.

[46] Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3), 2014.

[47] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob Mcgrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-goal reinforcement learning: challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

[48] Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: model-free deep RL For model-based control. In *International Conference on Learning Representations*, 2018.

[49] Vitchyr H. Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: state-covering self-supervised reinforcement learning. *CoRR*, abs/1903.03698, 2019.

[50] Ali Punjani and Pieter Abbeel. Deep learning helicopter dynamics models. In *IEEE International Conference on Robotics and Automation*, 2015.

[51] Sébastien Racanière, Théophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomenech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 2017.

[52] Danilo J Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, 2014.

[53] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, 2015.

[54] Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning Proceedings 1990*. Elsevier, 1990.

[55] Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *International Conference on Autonomous Agents and Multiagent Systems*, 2011.

[56] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, 2016.

[57] Vivek Veeriah, Junhyuk Oh, and Satinder Singh. Many-goals reinforcement learning. *arXiv preprint arXiv:1806.09605*, 2018.

[58] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. FeUdal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, 2017.

[59] David Warde-Farley, Tom Van de Wiele, Tejas Kulkarni, Catalin Ionescu, Steven Hansen, and Volodymyr Mnih. Unsupervised control through non-parametric discriminative rewards. *CoRR*, abs/1811.11359, 2018.

[60] Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: a locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems*, 2015.

[61] Marco Wiering and Jürgen Schmidhuber. Hq-learning. *Adaptive Behavior*, 6(2), 1997.

[62] Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew J Johnson, and Sergey Levine. Solar: deep structured latent representations for model-based reinforcement learning. In *International Conference on Machine Learning*, 2019.

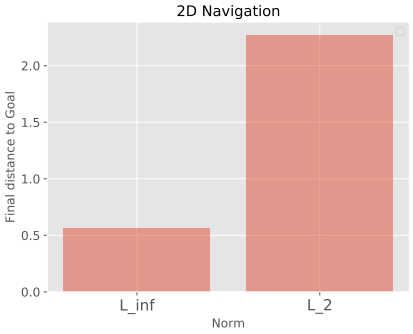

Figure 7: We compare using the ℓ_∞-norm to the ℓ_2-norm. We see that the ℓ_∞-norm outperforms ℓ_2-norm

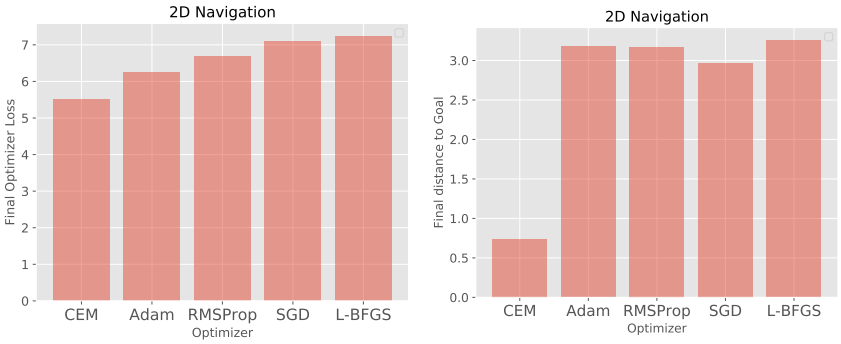

Figure 8: We compare CEM to different optimizers L-BFGS, Adam, RMSProp, and gradient descent (SGD) that have had their learning rates tuned. (Left) The optimizer loss, where CEM outperforms the other methods. (Right) The performance of the policy after using the plan chosen by each optimizer. We see that the lower optimizer loss of CEM corresponds to a better performance.

A Additional Experiments

A.1 Norm Ablation

We compare using the ℓ_∞-norm to minimize the feasibility vector with using the ℓ_2-norm. As shown in Figure 7, ℓ_∞-norm performs better, which matches the intuition it will more consistently push all terms in the feasibility vector towards zero.

A.2 Optimizer Ablation

We compare the performance of different optimizers on the 2D Navigation tasks. As shown in Figure 8, CEM consistently outperforms other optimizers both in terms of the optimizer loss, and the corresponding final performance on the task.

A.3 Likelihood Penalty Ablation

We examine the effect of the additional log-likelihood term (under the VAE prior) in Equation 3. In particular, we vary the weighting hyperparameter λ for the 2D Navigation and Push and Reach environments. For each environment, we note the final performance of the RL algorithm, in addition to the log-likelihood values and V values that compose equation 3. See Figure 9 for detailed results. We see that there is a trade-off between achieving a high likelihood under the prior and high V values. As we increase the weighting term λ the likelihood values increase while the V values decrease. There is an optimal threshold at which RL performance is maximized. For 2D Navigation, we note this value to be $\lambda = 0.01$ and for Push and Reach any range of values between 0.0001 and 0.01. For Ant Navigation, we independently verified an optimal choice of $\lambda = 0.1$.

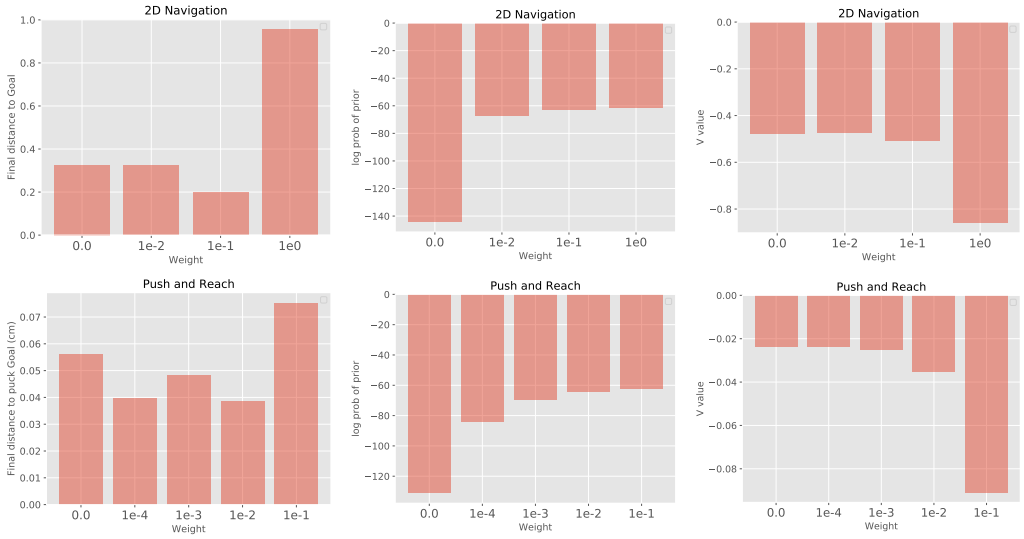

Figure 9: Examining the effect of the weight λ in Equation 3. We note the final RL performance (left), log-likelihood under the VAE prior (middle), and V values (right). As we increase λ, the log-likelihood values increase while the V values decrease. For 2D navigation (top), we note the optimal value to be $\lambda = 0.01$ and for Push and Reach (bottom) any range of values between 0.0001 and 0.01.

B Environment Details

B.1 2D Navigation

The agent must learn to navigate around a square room with a U-shaped wall in the center. See Figure 3 for a visualization of the environment. The dimensions of the space are 8×8 units, the walls are 1 unit thick, and the agent is a circle with radius 0.5 units. The observation is a 48×48 RGB image and the agent specifies a 2D velocity as the action. At each timestep, the agent can attempt to move up to 0.15 units in either dimension. The distance for Equation 1 is the distance between the current 2D position and the target position. We note that a greedy policy can easily lower the final distance by moving directly towards the goal. To measure whether or not the final policy performs more non-greedy behavior, we define success as whether or not the policy ends below the horizontal wall and within a diameter of the intended goal. Complete results are provided in Figure 10. Plots are averaged across 5 seeds, with the exception of PETS, which uses 3 seeds due to computational constraints. For image based baselines (all except PETS), we first train VAEs and select the top 5 seeds based on VAE loss. We proceed to training our RL algorithm with one seed per selected VAE. Note that for the ablation study in Figure 6, we select the top VAE seed based on VAE loss, and train our RL algorithm with 5 seeds.

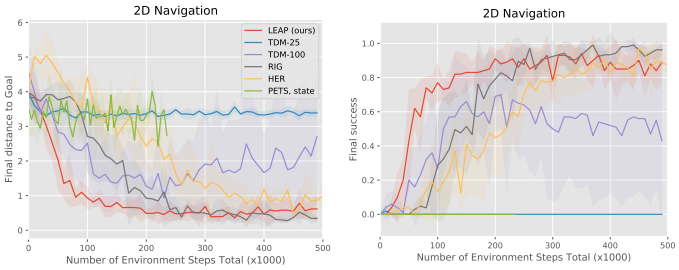

Figure 10: Complete 2D Navigation Results

B.2 Push and Reach

This task is based on the environment released by Nair et al. [40]. An additional invisible wall around the goal space of the puck has been added to prevent the puck from moving to unreachable hand

locations. In contrast to prior work evaluated on goal-conditioned pushing tasks [2, 47, 8], this task is solved using images as the observations and cannot be solved with a simple, unidirectional pushing behavior [40, 49]. Specifically, the observation is an 84×84 RGB image showing a top-down view of the scene. The robot is operated via 2D position control, where each action is limited to moving the robot end effector 2 cm in either dimension. The distance for Equation 1 is the Euclidean distance between (1) the goal and (2) the XY-position of the puck concatenated with the XY-position of the hand. We modify the task so as to require the agent to perform temporally extended planning. First, we increase the workspace of the environment to 40 cm $\times$ 20 cm. Second, we evaluate the final policy on 5 hard scenarios which require temporally extended behavior: rather than simply executing a simple, unidirectional pushing behavior, the robot must reach across the table to a corner where the puck is located, move its arm around the puck, and then pull the puck to a different corner of the table, as shown in Figure 3. A trajectory is successful if the final puck position is within 6 cm of the target position. For context, the puck has a radius of 4 cm. Complete results are provided in Figure 11. Plots are averaged across 8 seeds, with the exception of PETS, which uses 5 seeds due to computational constraints. For image based baselines (all except MPC), we first train VAEs and select the top 8 seeds based on VAE loss. We proceed to training our RL algorithm with one seed per selected VAE.

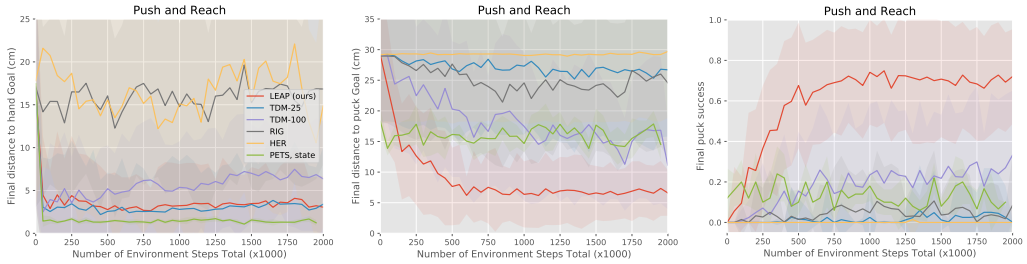

Figure 11: Complete Push and Reach Results

B.3 Ant Navigation

The ant must learn to navigate around a narrow rectangular room with a long wall in the center. See Figure 5 for a visualization of the environment. The dimensions of the space are 7.5×18 units, the wall is 1.5 units thick, and the ant has a radius of roughly 0.75 units. The state includes the position, orientation (in Euler angles rather than quaternions), joint angles, and velocities of the aforementioned components. The gear ratio for the ant is reduced to 10 units, to prevent the ant from flipping over. The distance for Equation 1 is the distance between the current 2D position and the target position, in addition to the differences in orientation of the ant with respect to the target orientation. We define success as whether or not the ant is within 1.5 units of the goal position. Complete results are provided in Figure 12. Plots are averaged across 15 seeds, with the exception of HIRO, which uses 5 seeds due to computational constraints. For LEAP, we first train VAEs and select the top 5 seeds based on VAE loss. We proceed to training our RL algorithm with three seed per selected VAE. Unlike the image-based experiments, the VAE is not used for training the RL algorithm. It is only used during test time for planning subgoals. The VAE is trained on a dataset in which the ant is in various valid positions of the maze, with a fixed orientation and fixed joint angles.

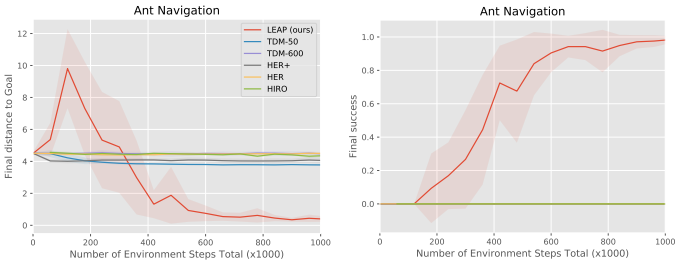

Figure 12: Complete Ant Navigation Results

Hyper-parameter	Value
Q network hidden sizes	$400, 300$
Policy network hidden sizes	$400, 300$
Q network and policy activation	ReLU
Q network output activation	None
Policy network output activation	tanh
Exploration noise	ϵ-greedy, $\epsilon = .1$ (2D Navigation) OU-process $\theta = .3$, $\sigma = .3$ (Push and Reach and Ant Navigation)
# training batches per time step	1
Batch size	128 (2D Navigation) 2048 (Push and Reach and Ant Navigation)
Optimizer	Adam
Learning rate (all networks)	0.001
Target update rate τ	0.005
Replay buffer size	1000000

Table 1: TD3 [19] hyperparameters.

C Implementation Details

This section contains descriptions and hyperparameters of the experiment implementations.

C.1 Goal-conditioned reinforcement learning

Both the Q network and policy concatenate all inputs and pass them through a feed-forward network. For RIG, the Q network outputs a scalar corresponding to the infinite discounted sum of rewards. For TDMs, the Q network outputs a vector corresponding to the negative distance between the final state and goal along each of the state dimensions. We train our networks using the twin delayed deep deterministic policy gradient algorithm [19] (TD3). Hyperparameter details are provided in Table 1. When sampling minibatches from the replay buffer, we sample transitions, goals, and times (for TDMs only). Inspired by RIG, we relabel the goals in our minibatches in the following manner:

- 20%: original goals from collected trajectories
- 40%: randomly sampled states from the replay buffer
- 40%: future states along the same collected trajectory, as dictated by hindsight experience replay [2] (HER).

We note that in the Ant Navigation task, we split sampling from the replay buffer to 20% from the replay buffer and 20% oracle goals from the environment.

C.2 Latent space optimization

In this subsection, we describe how we use the cross entropy method (CEM) [11] to optimize equation 3. Given an optimization problem over K subgoals, with each subgoal represented as an r-dimensional latent vector, the CEM optimizer is initialized with a standard multivariate Gaussian distribution $\mathcal{N}(0_{rK}, I_{rK})$, where 0_{rK} is a rK-dimensional vector of zeros, and I_{rK} is the $rK \times rK$ identity matrix. We sample different subgoal sequences from our distribution and evaluate the value of each sample using Equation 3. We then fit a diagonal multivariate Gaussian distribution to the top 5% of samples. We repeat this process for 15 iterations, and at each iteration we sample 1000 subgoal sequences from the fitted Gaussian. For the Ant Navigation task which involves optimizing over significantly higher number of subgoals, we sample 10000 subgoal sequences and run for 50 iterations instead. In addition, we found it beneficial to filter the top 25% of samples for the first half of iterations, and then filter the top 1% in the latter half. For the weight on the log-likelihood of the latents, we use $\lambda = 0.1$ for 2D Navigation and Ant Navigation tasks, and $\lambda = 0.001$ for Push and Reach.

C.3 Variational auto-encoder

We use separate VAE architectures for 2D Navigation (48×48 image) and Push and Reach (84×84 image). For 2D Navigation, encoder kernel sizes of $[5, 3, 3]$, encoder strides of $[3, 2, 2]$, $[16, 32, 64]$ encoder channels, decoder kernel sizes of $[3, 3, 6]$, decoder strides of $[2, 2, 3]$, and $[64, 32, 16]$ decoder channels are used. For Push and Reach, we use encoder kernel sizes of $[5, 5, 5]$, encoder strides of $[3, 3, 3]$, $[16, 16, 32]$ encoder channels, decoder kernel sizes of $[5, 6, 6]$, decoder strides of $[3, 3, 3]$, and $[32, 32, 16]$ decoder channels. Both architectures have a representation size of 16 and ReLU activation. We trained the 2D Navigation VAEs with binary cross-entropy loss, and the Push and Reach VAEs with mean squared error loss.