LEARNING ABSTRACT MODELS FOR LONG-HORIZON EXPLORATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite recent progress in reinforcement learning (RL), state-of-the-art RL algorithms continue to struggle with high-dimensional, long-horizon, sparse-reward tasks. Even with a perfect model, model-based RL can be intractable because the state space is often high-dimensional (e.g. over 10^{100} states). We address this by automatically constructing an abstract Markov Decision Process (MDP) with an exponentially smaller number of states (e.g. 10^5), where the actions are skills learned by a worker policy. We learn a near-optimal policy on the resulting abstract MDP, which maps to a near-optimal policy on the original MDP. Our approach provably makes monotonic progress and is guaranteed to learn a near-optimal policy. We empirically evaluate our approach on three of the hardest games from the Arcade Learning Environment: MONTEZUMA'S REVENGE, PITFALL!, and PRIVATE EYE, and outperform the previous state-of-the-art by over a factor of 2 in each game. In PITFALL!, our approach is the first to achieve superhuman performance without demonstrations[1].

1 INTRODUCTION

Current reinforcement learning (RL) algorithms struggle in high-dimensional, long-horizon, sparse-reward tasks because they rely on reward signal to learn but almost never receive reward in these tasks. For example, in the infamously difficult game MONTEZUMA'S REVENGE from the Arcade Learning Environment (Bellemare et al., 2013), even state-of-the-art intrinsically-motivated RL agents (Bellemare et al., 2016; Ostrovski et al., 2017; Martin et al., 2017; Tang et al., 2017) achieve only one-tenth the score of an average human (Hester et al., 2018) without demonstrations.

In this work, we explore model-based RL as a potential solution to these difficult tasks. The main advantage of model-based RL is its ability to guide exploration through *planning* based on its model of state dynamics and rewards. However, model-based RL has limited success in tasks with large, high-dimensional state spaces (Talvitie, 2014), partially due to compounding modeling errors during planning. To address this, some prior work has focused on learning more accurate models by using more expressive function approximators (Nagabandi et al., 2018), and learning local models (Levine & Koltun, 2013; Zhang et al., 2018). Others have attempted to robustly use imperfect models by conditioning on, instead of directly following, model-based rollouts (Weber et al., 2017), frequently replanning, and combining model-based with model-free approaches (Abbeel et al., 2006; Sutton, 1990). These works achieve significant improvements in relatively low-dimensional state spaces with relatively dense rewards. However, fundamentally, even if a perfect model were known, in tasks with extremely sparse, delayed rewards and high-dimensional, large state spaces (e.g. over 10^{100} states) recovering a near-optimal policy via planning (e.g. value iteration) may be intractable.

Instead of learning a model over such a large state space, we propose an alternative solution: map the state space to a much smaller abstract state space to learn a model on. Concretely, we assume access to a simple state abstraction function (Li et al., 2006; Singh et al., 1995), which maps a high-dimensional concrete state (e.g. all pixels on the screen) to a low-dimensional abstract state (e.g. the (x, y) coordinates of the agent). To construct an abstract Markov Decision Process (MDP) over the abstract state space, we deploy a hierarchical architecture composed of three parts: manager, worker, and discoverer. The *manager* maintains a *safe set* of abstract states that can be reliably

[1] Videos of our trained agent: `https://sites.google.com/view/abstract-models/home`

reached and estimates an abstract MDP over these abstract states. To expand this abstract MDP to more abstract states, the manager invokes the *discoverer* to discover new abstract states, and invokes the *worker* to learn how to navigate from states within the safe set to these newly discovered states. Once the worker has learned to reliably transition from a safe state to a newly discovered state, the manager adds the new state and the new transition to the abstract MDP. The worker itself operates in the original state space and uses an arbitrary RL algorithm to learn *skills* (i.e. options (Sutton et al., 1999)) to traverse the transition.

When the abstract MDP is fully constructed, our algorithm can plan out to any abstract state at will, and can optimize reward by choosing to navigate to the abstract state with the highest-reward plan. More concretely, our approach is guaranteed to make monotonic progress and learn a near-optimal policy (Section 6).

While abstraction provides us a more manageable state space, we need to address several challenges caused by information loss during abstraction. First, we avoid stochasticity during planning by having the abstract MDP contain only *reliable* transitions: transitions that the worker can carry out with success rate exceeding a threshold close to 1. This allows us to near-deterministically reach any abstract state in the safe set without having to worry about the underlying stochasticity in the original environment. Second, since the worker effectively learns the action set of the abstract MDP, the worker could learn transitions that violate the Markov property of the MDP.[2] As explained in Section 4, we impose conditions on the worker's learned skills to make the resulting process approximately Markov. Finally, the growth of the abstract MDP could stall if the worker forgets old transitions as it learns new ones. This might occur because the worker still operates on high-dimensional concrete states and consequently must represent its skills with function approximation, which can "forget" things learned in the past (Kirkpatrick et al., 2017). We prevent this forgetting by learning independent skills for each transition, enabling the abstract MDP to grow monotonically.

We empirically evaluate our approach on three of the most challenging games from the Arcade Learning Environment (Bellemare et al., 2013): MONTEZUMA'S REVENGE, PITFALL!, and PRIVATE EYE. Our approach significantly outperforms previous non-demonstration state-of-the-art approaches in all three domains. In PITFALL!, we are the first to achieve superhuman performance without demonstrations, surpassing the prior state-of-the-art by over 100 times on average. Additionally, since our approach learns a transition dynamics model, it can also generalize to new unseen reward functions in the same environment with no additional training. When evaluated on a new reward function never seen during training, our approach outperforms previous state-of-the-art methods that explicitly train on the new reward function.

2 APPROACH OVERVIEW

We assume the world is an unknown episodic finite-horizon MDP with (concrete) states $x \in \mathcal{X}$ and actions $a \in \mathcal{A}$. Our goal is to learn a policy $\pi : \mathcal{X} \to \mathcal{A}$ that maximizes the total expected reward. We further assume we have a simple predefined *state abstraction function* mapping concrete states x to abstract states $s = \phi(x)$. In MONTEZUMA'S REVENGE, for instance, a concrete state contains the pixels on the screen, while the corresponding abstract state contains the agent's position and inventory (Figure 1). We assume that the reward only depends on the abstract states; taking action a in concrete state x to x' always leads to reward $R(\phi(x), \phi(x'))$.

Our approach incrementally learns an abstract MDP over the abstract states, which will aid in exploration. Specifically, we maintain a *safe set* consisting of abstract states and skills that allow us to transition reliably between them (each skill is essentially an atomic action in the abstract MDP). We discover new abstract states and invoke a worker to learn new skills. Importantly, we ensure that the abstract MDP improves monotonically over time, so that we can use the abstract MDP to plan during exploration.

Drawing inspiration from the feudal manager-worker architecture (Vezhnevets et al., 2017), our approach consists of three components: The **manager** (Section 3) learns and plans using the abstract

[2]For example, if the agent moves from abstract state A into abstract state B with high velocity, its momentum may cause it to overshoot into some abstract state C. The probability of going from B to C now depends on A, violating the Markov property.

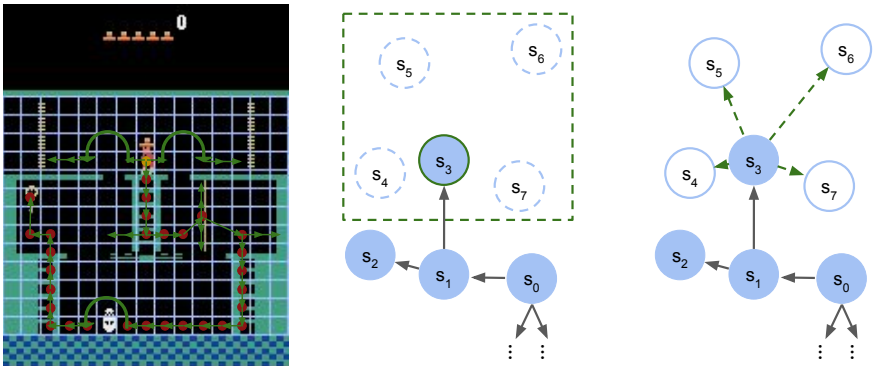


Figure 1: (a) Illustration of our approach on MONTEZUMA'S REVENGE. We have superimposed a white grid on top of the original game. At any given time, the agent is in one of the grid cells – each grid cell is an abstract state. In this example, the agent starts at the top of a ladder (yellow dot). The worker then navigates transitions between abstract states (green arrows) to follow a plan made by the master (red dots). (b) Circles represent abstract states. Shaded circles represent states within the safe set. The manager navigates the agent to the fringe of the safe set (s_3), then calls the discoverer to discover new abstract states near s_3 (dotted box). via random exploration. (c) The worker extends the abstract MDP by learning to navigate to the newly discovered abstract states (dotted arrows).

MDP. To improve the model, the manager calls the discoverer to find new abstract states and calls the worker to learn reliable transitions between known abstract states.

The **worker** (Section 4) learns a policy $\pi^{\mathrm{w}}(a \mid x, (s, s'))$ for performing the subtask of going from abstract state s to s', given the agent is in a concrete state x. This allows the manager to navigate the agent to any state s' by creating a plan $(s_0, s_1), (s_1, s_2), \cdots, (s_{k-1}, s_k = s')$ with its model and sequentially calling the worker on each transition (s_i, s_{i+1}). Finally, the **discoverer** (Section 5) discovers unseen abstract states and transitions via a random exploration policy $\pi^{\mathrm{d}}(a_t|x_{0:t}, a_{0:t-1})$. After navigating to an under-explored abstract state, the manager invokes the discoverer to find new nearby abstract states and to update its reward model from the observed rewards.

3 MANAGER

The manager estimates the abstract MDP consisting of:

- The *safe set*: the set of abstract states that the abstract MDP is currently defined over, which can be reliably reached through planning and calling the worker. The safe set initially contains only the starting abstract state and gradually grows as the worker learns to reliably traverse more transitions. The manager's goal is to cover the entire abstract state space with the safe set, enabling it to reach the abstract states corresponding to the highest reward.
- The reward dynamics $R(s, s')$, encoding the reward received when transitioning from abstract state s to abstract state s'. When the discoverer observes a tuple (x, a, r, x'), the manager sets $R(\phi(x), \phi(x')) = r$.
- The transition dynamics $P(s, s')$, encoding the probability of ending in the abstract state s' when calling the worker on the transition (s, s') starting in the abstract state s.[3] The manager sets $P(s, s')$ as a sliding window estimate of the success rate of the worker on (s, s')
- The abstract action set, formed by the transitions that the worker can to traverse.

[3] The transition (s, s') can be viewed as an abstract action in the abstract MDP. We choose this simplified notation of the transition dynamics model because the probability that the worker fails is vanishingly small since the worker learns reliable skills: $1 - P(s'|(s, s'), s) < \epsilon$, and all failures are considered equal.

Algorithm 1 MANAGER

while abstract MDP not fully constructed **do**
  Compute a set of candidate exploration goals, C
  ($c \in C$ is either a transition (s, s') or an abstract state s)
  Score all candidates and select highest priority candidate c
  Compute a plan $(s_0, s_1), (s_1, s_2), \cdots, (s_{T-1}, s_T = s)$ with model
  for $t = 1$ to T **do**
    Call worker to nagivate transition (s_{t-1}, s_t)
  if c is a transition (s, s') to learn **then**
    TRAIN_TRANSITION(s, s')
  else c is an abstract state s to explore
    EXPLORE()

The manager updates the abstract MDP and grows the safe set using Algorithm 1. On each episode, the manager either calls the worker to learn a new transition, or calls the discoverer to explore new abstract states. This is done by constructing a prioritized list of *exploration goals* where each goal is either a transition to learn or an under-explored abstract state to explore. Upon choosing an exploration goal with the highest priority (e.g. the transition (s, s')), the manager navigates to the relevant abstract state (e.g. s) by planning with the abstract MDP (producing a plan such as $(s_0, s_1), (s_1, s_2), \cdots, (s_{T-1}, s_T = s)$), tells the worker to execute the plan, and then calls the worker or the discoverer depending on the chosen candidate.

Transitions to learn. The manager generates candidate transitions for the worker to learn as follows. First, it starts with the set of all transitions (s, s') ever observed by the discoverer. Then, it removes all transitions (s, s') where s lies outside the safe set, since the agent would not know how to navigate to s before the transition can be learned. Finally, it adds "long-distance" transitions: (s, s') pairs for which the discoverer did not directly transition from s to s', but indirectly did so through a sequence of intermediate states: $(s_0 = s, s_1, \ldots, s_T = s')$. Let $d(s, s')$ be the length of the shortest such path. The manager adds all pairs (s, s') for which $d(s, s') \leq d_{\text{max}}$.

We found that long-distance transitions were crucial for successfully extending the safe set while preserving the Markov property of the abstract MDP. To illustrate why, consider how an agent might jump over a dangerous hole in MONTEZUMA'S REVENGE. This might involve traversing three abstract states: s_1 (the cliff in front of the hole), s_2 (the air above the hole), s_3 (solid ground on the other side of the hole). In this situation, the worker can easily go from s_1 to s_2 by simply walking off the cliff. But after falling into the hole, it can no longer reach s_3. By exploring a longer-distance transition (s_1, s_3), we avoid the hole.

Abstract states to explore. For each abstract state s in the safe set, the manager maintains a count $n(s)$ of the number of times the discoverer has explored the neighborhood around s. Once $n(s)$ exceeds some threshold N_{visit}, we assume that the discoverer has discovered all abstract states near s. The candidate abstract states s to explore are thus all s from the safe set with $n(s) \leq N_{visit}$.

Priority scores of the candidates. In theory, any priority function can be used as long as all candidates are eventually chosen. We uniformly switch between two priority functions, each aiming to prioritize easy candidates. The first priority function f_1 only seeks to expand the safe set without considering reward. On a candidate transition (s, s'), the priority score $f_1(s, s')$ is the sum of:

- How easy the transition is to learn: $\lambda_1 n_{\text{succ}}(s, s') - n_{\text{fail}}(s, s') - d(s, s')^2$, where $n_{\text{succ}}(s, s')$ is the number of times the worker has successfully traversed (s, s'), $n_{\text{fail}}(s, s')$ is the number of times the worker has failed to traverse (s, s').
- How useful learning the transition would be: positive score λ_2 if learning the transition would introduce more candidate targets, and 0 otherwise.

On a candidate abstract state s to explore, we simply define $f_1(s) = \lambda_3 - n(s)$. The second priority function f_2 is equal to the first, except it additionally considers the reward of the path to reach the candidate, according to the model. Concretely, $f_2(s, s') = f_1(s, s') + \sum_{t=0}^{T} R(s_t, s_{t+1})$, where

Algorithm 2 TRAIN_TRANSITION(s, s', x_0)

Input: a transition (s, s') to learn, called at concrete state x_0 with $s(x_0) = s$
Set worker horizon $H = d(s, s') \times H_{\text{worker}}$
Choose $a_0 \sim \pi^{\text{w}}(x_0, (s, s')) = \pi_{k(s_{T-1}, s')}(x_0, s')$
for $t = 1$ to H **do**
    Observe x_t
    Compute worker intrinsic reward $r_t = R_{(s,s')}(x_t|s')$
    Update worker on $(x_{t-1}, a_{t-1}, r_t, x_t)$
    Choose $a_t \sim \pi^{\text{w}}(x_t, (s, s')) = \pi_{k(s,s')}(x_t, s')$
Compute success = $\mathbf{1}[r_1 + \cdots + r_H \geq R_{\text{min}}]$
Update transition model $P(s, s') \leftarrow$ success rate of past $N_{transition}$ attempts
if $P(s, s') \geq 1 - \epsilon$ **then**
    Freeze worker's skill $\pi_{k(s,s')}$

$(s_0, s_1), \cdots, (s_{T-1}, s_T = s)$ is the plan to navigate to s. We define f_2 on an abstract state s by adding the path reward to $f_1(s)$ in the same way.

4 WORKER

The worker learns to execute concrete actions that can navigate a given transition (s, s'). This is done via a policy $\pi^{\text{w}}(a|x, (s, s'))$, which picks the next concrete action conditioning on the current concrete state and the transition. During training, the worker tries to satisfy three criteria:

1) The worker seeks to *highly reliably* navigate each transition, absorbing stochasticity from the environment, so that the manager's model does not suffer from compounding errors.
2) The worker seeks to make *monotonic progress* so that when it learns to navigate a new transition, its ability to traverse old transitions remains unchanged.
3) The worker seeks to make the abstract MDP *Markov* so that the manager's model can condition on just the current abstract state instead of the entire history.

While it is possible to learn a single model $\pi^{\text{w}}(a|x, (s, s'))$ for handling all transitions, it is tricky to satisfy 2) as newly learned transitions can have deleterious effects on previously learned transitions. Instead, we opt for a non-parametric approach by learning a separate *skill* for each transition. When a skill learns to navigate the transition with high probability, the worker freezes the skill's parameters so that it never changes again.

Concretely, the worker maintains a *skill repository* K, which maps each transition (s, s') to the skill that can reliably navigate that transition. Each skill is a goal-conditioned sub-policy $\pi_{K(s,s')}(a|x, s')$, which produces concrete actions a conditioned on the current concrete state x and the goal abstract state s'. When the worker navigates a transition (s, s'), it calls on the corresponding skill until the transition is traversed, i.e. $\pi^{\text{w}}(a|x, (s, s')) = \pi_{K(s,s')}(a|x, s')$.

To learn a new transition, the worker first tries to reuse the skills it already knows from the skill repository. The worker measures the skill's success rate over $N_{transition}$ attempts, and reuses the skill by updating the skill repository mapping if the skill can reliably traverse the transition (i.e. the success rate is at least $1 - \epsilon$). If no previously learned skill can reliably traverse the new transition, the worker creates a new skill and trains it to navigate the transition by optimizing intrinsic reward during skill episodes (Algorithm 2). For a transition (s, s'), a skill episode is a fixed number of timesteps, proportional to the distance between s and s', set as $d(s, s') \times H_{\text{worker}}$. During the skill episode, the skill receives intrinsic reward $R_{(s,s')}(x) = 1$ if $s(x) = s'$ and 0 otherwise. The skill episode additionally terminates if the main episode terminates or if the manager receives negative environment reward.

To learn skills that make the abstract MDP Markov (criteria 3), the worker imposes the *holding heuristic*: a skill episode is counted as successful only if the skill accumulates at least R_{min} reward during the skill episode. Intuitively, the holding heuristic enforces that the skill can stay at an abstract state for many timesteps, meaning that the worker is in control (e.g. not falling or about to die).

Algorithm 3 EXPLORE(x_0)

Input: called at concrete state x_0 to explore neighborhood of $s(x_0)$
Choose $a_0 \sim \pi^{\mathrm{d}}(x_0)$
while $n(s(x_0)) \leq N_{visit}$ **do**
  for $t = 1$ to T_{d} **do**
    Observe x_t, r_t
    if $(s(x_{t-1}), s(x_t))$ has never been observed before **then**
      Update reward model $R(s(x_{t-1}), s(x_t)) \leftarrow r_t$
      Add transition to transitions model $P(s(x_{t-1}), s(x_t)) \leftarrow 0$.
    Choose $a_t \sim \pi^{\mathrm{d}}(x_{1:t}, a_{1:t-1})$
  Continue exploring: reset $x_0 \leftarrow x_{T_{\mathrm{d}}}$ and choose $a_0 \sim \pi^{\mathrm{d}}(x_0)$

Any RL algorithm can be used to represent and train the skills. We choose to represent each skill as a Double Deep Q-Learning Network (DDQN) (van Hasselt et al., 2016; Wang et al., 2016). Different skills can require vastly different training times. For example, learning to jump over the monster requires more episodes and more exploration than learning to move a few steps to the left. To address this, the skills use a saw-tooth epsilon schedule for their epsilon-greedy exploration, motivated by the doubling trick from online learning (Auer et al., 1995). In addition, for faster training, the skills use a form of self-imitation (Oh et al., 2018) to more quickly learn from previous successful episodes, and a form of count-based intrinsic motivation similar to (Bellemare et al., 2016) to more quickly initially discover skill reward. Appendix **??**ully describes our skill training and architecture.

5 DISCOVERER

The discoverer explores new abstract states and new transitions for the worker to learns. It does this by exploring the neighborhood of an abstract state chosen by the manager and following a simple exploration policy for a fixed number of timesteps T_{d} (Algorithm 3). The discover records the transitions and rewards it observes: $(s(x_0), r_0, s(x_1)), (s(x_1), r_1, s(x_2)), \cdots, (s(x_{T_{\mathrm{d}}-1}), r_{T_{\mathrm{d}}-1}, s(x_{T_{\mathrm{d}}}))$. It then reports this information back to the manager, which uses the rewards to update its rewards model and the transitions as candidates for the worker. In addition, if the discoverer ends in an under-explored abstract state (i.e. $n(s(x_{T_{\mathrm{d}}})) \leq N_{visit}$) after exploring for T_{d} timesteps, it simply explores for another T_{d} timesteps.

In theory, the discoverer can use any policy $\pi^{\mathrm{d}}(a_t | x_{0:t}, a_{0:t-1})$ that places non-zero probability on every action to randomly explore (e.g. uniformly random). Our discoverer uses a slightly more complex policy. At each timestep, the discoverer uniformly samples an action, as well as a number between 1 and T_{repeat}, and repeats the action the sampled number of times.

6 FORMAL ANALYSIS

Under certain, relatively strong assumptions, our approach provably recovers a near-optimal policy on the original MDP in time polynomial in the size of the abstract MDP (see Appendix for details). In contrast, tabular RL algorithms such as R-MAX or MBIE (Brafman & Tennenholtz, 2002; Strehl & Littman, 2005) also give guarantees for learning a near-optimal policy, but require time and space polynomial in the size of the original MDP, which is intractable (e.g. 10^{100} or more states). Earlier work combining probably efficient R-MAX with hierarchy had sample complexity bounds that scaled exponentially with the horizon (K & Peter, 2008).

In the context of general function approximation and RL, there has been growing interest and encouraging results in bounding the performance of RL algorithms with function approximation. Recent work on contextual Markov decision processes (Jiang et al., 2016; Dann et al., 2018) provides sample complexity bounds as a function of the Bellman rank of the domain but in domains like Atari how small the Bellman rank of the ground domain is, and these algorithms do not specifically address how to construct an alternate representation which may facilitate learning, as we do here. For the setting of deterministic episodic systems, recent work bounds the regret in terms of the

Eluder dimension, which for state aggregated domains will be a polynomial function of the state features (Wen & Roy, 2013; Russo & Roy, 2013); however the proposed approach is only demonstrated on small domains and it is unclear it will be computationally feasible in very large feature spaces. In contrast to this recent work where the primary contribution is on the theoretical analysis, our aim is to create algorithms that can practically solve long horizon, sparse, high dimensional domains.

7 EXPERIMENTS

7.1 TASK SETUP

We empirically evaluate our approach on three of the most challenging games from the Arcade Learning Environment (Bellemare et al., 2013): MONTEZUMA'S REVENGE, PITFALL!, and PRIVATE EYE. We use the standard Arcade Learning Environment setup (Appendix A) and end the episode when the agent loses a life. We perform periodic evaluations every 4000 episodes by having the manager plan for optimal reward in the currently constructed abstract MDP. We average our approach over 4 seeds and report 1 standard deviation error bars in the training curves. Our experiments use the same set of hyperparameters (Appendix A.1) across all three games, where the hyperparameters were exclusively and minimally tuned on MONTEZUMA'S REVENGE.

In all three games, the state abstraction function uses the RAM state, available through the Atari simulator, to extract the bucketed location of the agent and the agent's current inventory. Roughly, this distinguishes states where the agent is in different locations or has picked up different items, but doesn't distinguish states where other details differ (e.g. whether the monster is in a different location, or the obstacles are in a different configuration). Notably, the abstract state function does not specify what each part of the abstract state means, and the agent does not know the entire abstract state space beforehand. We describe the exact abstract states in Appendix A.2.

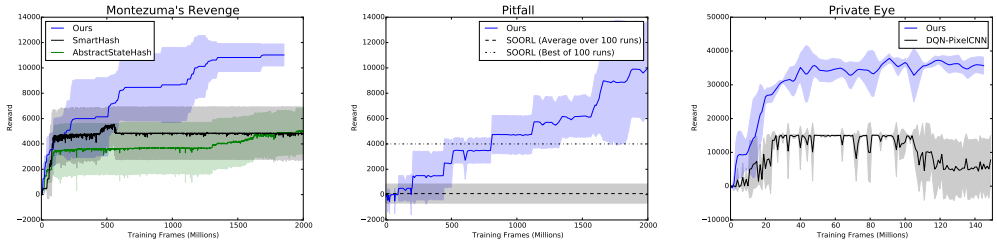
7.2 MAIN RESULTS

We compare our approach to the previous state-of-the-art approach that does not use demonstrations on each game:

- In MONTEZUMA'S REVENGE, we compare against *SmartHash* (Tang et al., 2017), a count-based exploration approach which estimates state visit counts with a hash-based density model and provides intrinsic reward to revisit states with low visit counts. Like our approach, *SmartHash* also uses RAM state information. It hashes each state to a hand-selected subset of the RAM state and maintains visit counts on the hashes. We additionally compare with *AbstractStateHash*, which runs the same code as *SmartHash*, but instead hashes based on the subset of the RAM used by our state abstraction function.[4]
- In PITFALL!, we compare with *SOORL* (Keramati et al., 2018), a planning approach which requires prior knowledge to extract the objects on the screen. *SOORL* is the only non-demonstration approach to achieve positive reward in PITFALL!, but requires extensive engineering to identify and extract all objects. All other non-demonstration approaches achieve 0 reward or less. Once *SOORL* has access to the objects, it can learn in very few frames since data from all similar objects can be pooled in learning. Consequently, in our training curves, we report its average performance over 100 runs, as well as its best performance over 100 runs, instead of training curves.
- In PRIVATE EYE, we compare with another count-based exploration method, *DQN-PixelCNN* (Ostrovski et al., 2017), which uses a pixel-based density model to estimate state visitation counts. We compare with the results reported in Ostrovski et al. (2017).

Figure 2 shows the main results. In MONTEZUMA'S REVENGE, while *SmartHash* is competitive with our approach for the first 150M frames, it quickly plateaus, whereas our approach continues to learn. Our approach achieves a final average reward of 11020, almost doubling the previous average reward of 5001 from *SmartHash*. Additionally, the strong but sub-optimal performance

[4]The results from *SmartHash* and *AbstractStateHash* are obtained by running code generously provided by the original authors (Tang et al., 2017).



(a) (b) (c)

Figure 2: Comparison of our approach with the prior non-demonstration state-of-the-art approach in MONTEZUMA'S REVENGE, PITFALL!, and PRIVATE EYE.

Task	Ours (Best Run)	Ours (Worst Run)	Prior SOTA (Best Run)
MONTEZUMA'S REVENGE	**12500**	*10500*	6600 (*SmartHash*)
PITFALL!	**15000**	*6000*	4000 (*SOORL*)
PRIVATE EYE	**40100**	36200	*39000* (*DQN-PixelCNN*)

Table 1: We compare our worst performing seed against the best performing seed of the prior state-of-the-art. We additionally report our best-performing seed. Note: Bellemare et al. (2016) matches the peak performance of *SmartHash* to get 6600 on MONTEZUMA'S REVENGE.

of *AbstractStateHash* suggests that simply having access to the state abstraction function is not sufficient to achieve high reward.

Our approach is the first non-demonstration approach to achieve superhuman performance on PITFALL!, significantly outperforming the prior non-demonstration state-of-the-art, *SOORL*. After 2B frames of training, our approach achieves a final average reward of 9959.6, in contrast to human performance, which averages to 6464 (Pohlen et al., 2018), and *SOORL*, which achieves a maximum reward of 4000 over 100 runs, but only achieves a mean reward of 80.52 over those runs. In addition, our approach even significantly outperforms *Ape-X DQfD*, which uses high-scoring expert demonstrations during training to achieve an average score of 3997.5.

In PRIVATE EYE, our approach achieves a mean reward of 35636.1, more than double the reward of *DQN-PixelCNN*, which achieves 15806.5.

Stability. Recent work (Islam et al., 2017) has drawn attention to the instability of deep RL results. To highlight the stability of our results, in each game, we compare our worst performing seed against the prior state-of-the-art in Table 1. Even our worst performing seed outperforms the mean performance of the prior state-of-the-art approaches. In addition, our worst performing seed is competitive with the highest previously reported scores in each of the games, significantly outperforming the previous high in MONTEZUMA'S REVENGE and PITFALL!, and narrowly performing worse than the previous high in PRIVATE EYE. Furthermore, our best performing seeds achieve new peak performances in each of the games.

Skill sharing. Our worker successfully learns skills that generalize to many similar transitions. This reduces the number of required parameters and saves many training frames. Due to space constraints, we defer to Appendix B.1 for greater detail.

7.3 GENERALIZATION TO NEW TASKS

Since the manager maintains the abstract MDP, our approach can generalize to new tasks in the same environment that were not seen during training. It can do this by revisiting all the transitions in the abstract MDP and updating the rewards model with the newly observed reward. We study the ability of our approach to do this. After our approach completes training on the original reward function on MONTEZUMA'S REVENGE, we evaluate its performance on three new navigation-based

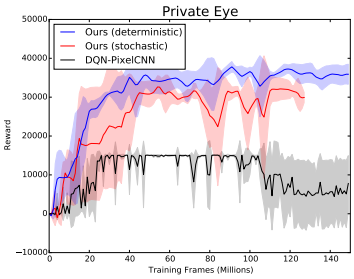


Figure 3: Our method continues to outperform the prior state-of-the-art on the stochastic version of PRIVATE EYE.

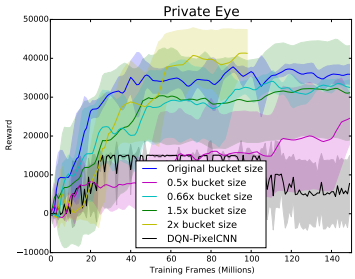


Figure 4: Our method outperforms the prior state-of-the-art on a wide range of granularities of the state abstraction function.

reward functions, allowing our approach to first interact with the environment for an additional 1M frames to observe each new reward function. We compare with *DQN-CTS* (Bellemare et al., 2016), which performs similarly to the state-of-the-art *SmartHash* on MONTEZUMA'S REVENGE, trained from scratch directly on the new reward functions for near 50M frames. Our approach successfully generalizes and optimizes the new reward function, achieving an average reward of 716.7 out of an optimal reward of 1000. In contrast, even when trained from scratch directly on the new reward function, *DQN-CTS* achieves an average of -0.42 at the end of evaluation. We describe the reward functions and present more detailed results in Appendix B.2.

7.4 ROBUSTNESS TO STOCHASTICITY

We additionally evaluate the performance of our approach on the recommended (Machado et al., 2017) form of Atari stochasticity (sticky actions 25% of the time) on PRIVATE EYE (selected because it requires the fewest frames for training). Figure 3 compares the performance of our method on the stochastic version of PRIVATE EYE with the performance of our method on the deterministic version of PRIVATE EYE. Performance degrades slightly on the stochastic version, because the worker's skills become harder to learn. However, both versions outperform the prior state-of-the-art *DQN-PixelCNN*, and the worker is able to successfully abstract away stochasticity from the manager in the stochastic version of the game, so that the abstract MDP is still near-deterministic.

7.5 VARYING THE GRANULARITY OF THE ABSTRACT STATE

We further study the robustness of our approach to state abstraction functions of varying degrees of coarseness on PRIVATE EYE. Our state abstraction function buckets the agent's (x, y) coordinates. We vary the coarseness of the abstraction function by varying the bucketing size: increasing the bucketing size results in fewer, coarser abstract states.

We report results in Figure 4 on five different bucket sizes obtained by scaling the original bucket size by $\frac{1}{2}, \frac{2}{3}, 1, \frac{3}{2}$, and 2. To adjust for the updated bucket sizes, we also scale the worker's skill episode horizon H_{worker} by the same value. Notably, our method outperforms the prior state-of-the-art approach *DQN-PixelCNN* across the entire range of bucket sizes, although performance degrades when the bucket size becomes too small.

In general, changing the granularity of the abstraction makes a trade-off between the size of the abstract MDP and the difficulty of learning each worker skill. As the abstraction becomes more fine-grained, abstract states are fewer actions apart, and learning each skill becomes easier. However, the abstract MDP also grows in size, which means that the worker must learn a greater number of total skills. At one extreme, when the abstract state becomes too fine-grained, the worker can no longer abstract stochasticity away from the manager. Intuitively, the worker lacks the leeway to deal with stochasticity because the manager's plan specifies the goals in too great detail. At the other extreme, when the abstract state become too coarse, the worker's reward risks becoming sparse, since nearby abstract states are far away. This can prevent the worker from effectively learning skills. However, empirically, performance stays strong through a wide range of granularities.

8 RELATED WORK

Exploration in tabular settings is well-understood via optimism in the face of uncertainty (OFU) (Brafman & Tennenholtz, 2002; Strehl et al., 2009) and posterior sampling (Osband et al., 2013; Osband & Roy, 2016). OFU methods such as R-MAX (Brafman & Tennenholtz, 2002), MBIE (Strehl & Littman, 2005), and UCRL (Jaksch et al., 2010) efficiently explore by providing bonuses for exploring regions of uncertainty and provably yield near-optimal policies in polynomial time. Nevertheless, despite recent progress that has yielded much stronger optimality bounds (Azar et al., 2017; Dann et al., 2017), these methods do not scale well to the deep RL setting, where the state space dimensionality is prohibitively large. Bellemare et al. (2016); Tang et al. (2017); Ostrovski et al. (2017) generalize optimism to high-dimensional state spaces. However, these methods no longer guarantee optimality and can overlook states unseen states, whereas our discoverer proactively seeks new abstract states.

Model-based RL approaches achieve great success in the tabular setting (Strehl et al., 2009), relatively low-dimensional state spaces (Nagabandi et al., 2018), and in MDPs with known dynamics and relatively dense reward (Silver et al., 2016; 2017). However, in high-dimensional state spaces, function approximation can cause compounding model errors (Talvitie, 2017; 2014). Many works (Weber et al., 2017) address this issue, but with sparse, delayed rewards in high-dimensional state spaces, planning can be intractable even with a perfect model,

Our approach learns a model in the smaller abstract state space, drawing motivation from prior work on hierarchical reinforcement learning (HRL): state abstraction (Li et al., 2006; Singh et al., 1995), action abstraction (Sutton, 1995; Sutton et al., 1999), and learning subgoals (Dietterich, 2000). However, prior work on HRL has drawbacks. Many works rely on human-engineered goal specification (Kulkarni et al., 2016) or options (Sutton et al., 1999). Others learn high-level options from scratch (Bacon et al., 2017; Vezhnevets et al., 2017), but risk learning unhelpful problem decompositions. Our work requires some prior knowledge to provide the state abstraction function, but given the state abstraction function, the problem naturally and usefully decomposes into learning many transitions. (Roderick et al., 2017) uses a similar framework to ours, but our approach empirically performs much better, due to our design decisions for incremental and monotonic progress.

We note that imitation learning methods (Aytar et al., 2018; Hester et al., 2018; Pohlen et al., 2018) achieve high scores on the Arcade Learning Environment. However, these require extensive prior knowledge in the form of high-scoring human demonstrations, which circumvents the exploration problem, because high reward regions are known.

9 DISCUSSION

This work presents a framework for tackling long-horizon, sparse-reward, high-dimensional tasks by using abstraction to address the compounding errors problem in model-based RL. Empirically, this framework performs well in hard exploration tasks, and additionally provides theoretical guarantees of near-optimality. However, this work is not without limitations. First, our approach relies on some prior knowledge in the form of the state abstraction function. While the abstraction function is usually simple and natural, and especially easy to specify in the Arcade Learning Environment, where the RAM is exposed, extracting this information in general can be more difficult. Still, the abstraction function provides less prior knowledge than expert demonstrations, which effectively solve the exploration problem, and prior state-of-the-art approaches (Bellemare et al., 2016; Ostrovski et al., 2017) also implicitly encode prior knowledge into their agents in the form of a density model, although these approaches can be easier to apply to new domains. Future work could attempt to automatically learn the state abstraction or extract the abstraction directly from the visible pixels. One potential method for automatically learning the state abstraction function would be to start with an extremely coarse represention, and then iteratively refine the representation by splitting abstract states whenever reward is discovered. We leave this to future work. Another limitation of our work is that our simple theoretical guarantees only hold under relatively strong assumptions. Fortunately, even when these assumptions are not satisfied, our approach can still perform well, but no longer has guarantees for optimality

Reproducibility Our code is available at `https://github.com/anonymous`

REFERENCES

P. Abbeel, M. Quigley, and A. Y. Ng. Using inaccurate models in reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 1–8, 2006.

P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *International Conference on Robotics and Automation (ICRA)*, pp. 322–322, 1995.

Y. Aytar, T. Pfaff, D. Budden, T. L. Paine, Z. Wang, and N. de Freitas. Playing hard exploration games by watching youtube. *arXiv preprint arXiv:1805.11592*, 2018.

M. G. Azar, I. Osband, and R. Munos. Minimax regret bounds for reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2017.

P. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pp. 1726–1734, 2017.

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

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research (JAIR)*, 47: 253–279, 2013.

R. Brafman and M. Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, pp. 213–231, 2002.

C. Dann, T. Lattimore, and E. Brunskill. Unifying PAC and regret: Uniform PAC bounds for episodic reinforcement learning. *Advances in Neural Information Processing Systems*, pp. 5713–5723, 2017.

C. Dann, N. Jiang, A. Krishnamurthy, A. Agarwal, J. Langford, and R. E. Schapire. On polynomial time PAC reinforcement learning with rich observations. *arXiv preprint arXiv:1803.00606*, 2018.

T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, pp. 227–303, 2000.

T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. Agapiou, J. Z. Leibo, and A. Gruslys. Deep Q-learning from demonstrations. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2018.

R. Islam, P. Henderson, M. Gomrokchi, and D. Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017.

T. Jaksch, R. Ortner, and P. Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, pp. 1563–1600, 2010.

N. Jiang, A. Krishnamurthy, A. Agarwal, J. Langford, and R. E. Schapire. Contextual decision processes with low bellman rank are PAC-learnable. *arXiv preprint arXiv:1610.09512*, 2016.

J. N. K and S. Peter. Hierarchical model-based reinforcement learning: R-max+ maxQ. In *International Conference on Machine Learning (ICML)*, pp. 432–439, 2008.

R. Keramati, J. Whang, P. Cho, and E. Brunskill. Strategic object oriented reinforcement learning. *arXiv preprint arXiv:1806.00175*, 2018.

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

J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 2017.

T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, pp. 3675–3683, 2016.

S. Levine and V. Koltun. Guided policy search. In *International Conference on Machine Learning (ICML)*, 2013.

L. Li, T. J. Walsh, and M. L. Littman. Towards a unified theory of state abstraction for mdps. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2006.

M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *arXiv preprint arXiv:1709.06009*, 2017.

J. Martin, S. N. Sasikumar, T. Everitt, and M. Hutter. Count-based exploration in feature space for reinforcement learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

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. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *International Conference on Robotics and Automation (ICRA)*, pp. 7559–7566, 2018.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning (ICML)*, pp. 807–814, 2010.

J. Oh, Y. Guo, S. Singh, and H. Lee. Self-imitation learning. *arXiv preprint arXiv::1806.05635*, 2018.

I. Osband and B. V. Roy. Why is posterior sampling better than optimism for reinforcement learning. *arXiv preprint arXiv:1607.00215*, 2016.

I. Osband, D. Russo, and B. V. Roy. (more) efficient reinforcement learning via posterior sampling. *Advances in Neural Information Processing Systems*, pp. 3003–3011, 2013.

G. Ostrovski, M. G. Bellemare, A. Oord, and R. Munos. Count-based exploration with neural density models. In *International Conference on Machine Learning (ICML)*, pp. 2721–2730, 2017.

T. Pohlen, B. Piot, T. Hester, M. G. Azar, D. Horgan, D. Budden, G. Barth-Maron, H. van Hasselt, J. Quan, M. Večer'ık, et al. Observe and look further: Achieving consistent performance on ATARI. *arXiv preprint arXiv:1805.11593*, 2018.

M. Roderick, C. Grimm, and S. Tellex. Deep abstract Q-networks. *arXiv preprint arXiv:1710.00459*, 2017.

D. Russo and B. V. Roy. Eluder dimension and the sample complexity of optimistic exploration. *Advances in Neural Information Processing Systems*, pp. 2256–2264, 2013.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. V. D. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L., M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

S. P. Singh, T. Jaakkola, and M. Jordan. Reinforcement learning with soft state aggregation. *Advances in neural information processing systems*, pp. 361–368, 1995.

A. L. Strehl and M. L. Littman. A theoretical analysis of model-based interval estimation. *Proceedings of the 22nd international conference on Machine learning*, pp. 856–863, 2005.

A. L. Strehl, L. Li, and M. L. Littman. Reinforcement learning in finite mdps: PAC analysis. *Journal of Machine Learning Research*, pp. 2413–2444, 2009.

R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Machine Learning Proceedings*, pp. 216–224, 1990.

R. S. Sutton. TD models: Modeling the world at a mixture of time scales. *Machine Learning Proceedings*, pp. 531–539, 1995.

R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Articial intelligence*, 112:181–211, 1999.

E. Talvitie. Model regularization for stable sample rollouts. In *Uncertainty in Artificial Intelligence (UAI)*, pp. 780–789, 2014.

E. Talvitie. Self-correcting models for model-based reinforcement learning. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pp. 2597–2603, 2017.

H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel. #exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2753–2762, 2017.

H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double Q-learning. In *Association for the Advancement of Artificial Intelligence (AAAI)*, volume 16, pp. 2094–2100, 2016.

A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

Z. Wang, T. Schaul, M. Hessel, H. V. Hasselt, M. Lanctot, and N. D. Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2016.

C. Watkins. Learning from delayed rewards. *King's College, Cambridge*, 1989.

T. Weber, S. Racanière, D. P. Reichert, L. Buesing, A. Guez, D. J. Rezende, A. P. Badia, O. Vinyals, N. Heess, Y. Li, et al. Imagination-augmented agents for deep reinforcement learning. *arXiv preprint arXiv:1707.06203*, 2017.

Z. Wen and B. V. Roy. Efficient exploration and value function generalization in deterministic systems. *Advances in Neural Information Processing Systems*, pp. 3021–3029, 2013.

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

A Experiment Details

Following Mnih et al. (2015), the pixel concrete states are downsampled and cropped to 84 by 84 and then are converted to grayscale. To capture velocity information, the worker receives as input the past four frames stacked together. Every action is repeated 4 times.

In addition, Montezuma's Revenge and Pitfall! are deterministic by default. As a result, the manager deterministically navigates to the fringes of the safe set by calling on the worker's deterministic, saved skills. To minimize wallclock training time, we save the states at the fringes of the safe set and enable the worker to teleport to those states, instead of repeatedly re-simulating the entire trajectory. When the worker teleports, we count all the frames it would have had to simulate as part of the training frames. Importantly, this only affects wallclock time, and does not benefit or change the agent in any way. Notably, this does not apply to Private Eye, where the initial state is stochastically chosen from two similar possible states.

Hyperparameter	Value
Success weight λ_1	(**1** (stochastic), 10, **100** (deterministic))
New transition exploration goal weight λ_2	**5000**
Abstract state exploration goal weight λ_3	**-2000**
Discoverer exploration horizon T_d	**50**
Discoverer visit threshold N_{visit}	**500**
Discoverer repeat action range	**1 to 20**
Worker horizon H_{worker}	(10, 15, 20, **30**, 45)
Skill failure tolerance ϵ	(**0.05**, 0.1)
Skill minimum reward R_{min}	**4**
Maximum transition distance d_{max}	**15**
Dynamics sliding window size $N_{transition}$	**100**
Adam learning rate	**0.001**
Max buffer size of each skill	**5000**
Skill DQN target sync frequency	**75**
Skill batch size	**32**
Skill minimum buffer size	**50**
Gradient norm clipping	**3.0**
Count-based weight β	**0.63**
Margin weight λ	**0.5**

Table 2: Table of all hyperparameters and the values used in the experiments.

A.1 Hyperparameters

All of our hyperparameters are only tuned on Montezuma's Revenge. Our skills are trained with the Adam optimizer (Kingma & Ba, 2014) with the default. Table 2 describes all hyperparameters and the values used during experiments (bolded), as well as other values that we tuned over (non-bolded). Many of our hyperparameters were selected once and never tuned.

A.2 State Abstraction Function

In Montezuma's Revenge, each abstract state is a (bucketed agent x-coordinate, bucketed agent y-coordinate, agent room number, agent inventory, current room objects, agent inventory history) tuple. These are given by the RAM state at indices 42 (bucketed by 20), 43 (bucketed by 20), 3, 65, and 66 respectively. The agent inventory history is a counter of the number of times the current room objects change (the room objects change when the agent picks up an object).

In Pitfall!, each abstract state is a (bucketed agent x-coordinate, bucketed agent y-coordinate, agent room number, items that the agent has picked up) tuple. These are given by the RAM state at indices 97 (bucketed by 20), 105 (bucketed by 20), 1, and 113 respectively.

In Private Eye, each abstract state is a (bucketed agent x-coordinate, bucketed agent y-coordinate, agent room number, agent inventory, agent inventory history, tasks completed by the agent) tuple. These are given by the RAM state at indices 63 (bucketed by 40), 86 (bucketed by 20), 92, 60, 72, and 93 respectively.

A.3 Skill Training and Architecture

Architecture. Our skills are represented as Dueling DDQNs (van Hasselt et al., 2016; Wang et al., 2016), which produce the state-action value $Q_{(s,s')}(x, a) = A_{(s,s')}(x, a) + V_{(s,s')}(x)$, where $A_{(s,s')}(x, a)$ is the advantage and $V_{(s,s')}(x)$ is the state-value function. The skills recover a policy $\pi_{k(s,s')}(a|x, (s, s'))$ by greedily selecting the action with the highest q-value at each concrete state x.

The skill uses the standard architecture (Mnih et al., 2015) to represent $A_{(s,s')}(x, a)$ and $V_{(s,s')}(x)$ with a small modification to also condition on the transition (s, s'). First, after applying the standard Atari pre-processing, the skill computes the pixel embedding e_x of the pixel state x by applying three

square convolutional layers with (filters, size, stride) equal to $(32, 8, 4)$, $(64, 4, 2)$, and $(64, 4, 2)$ respectively with rectifier non-linearities (Nair & Hinton, 2010), and applying a final rectified linear layer with output size 512. Next, the skill computes the transition embedding $e_{(s,s')}$ by concatenating $[e_r; e_{diff}]$ and applying a final rectified linear layer with output size 64, where:

- e_r is computed as the cumulative reward received by the skill during the skill episode, represented as one-hot, and passed through a single rectified linear layer of output size 32.
- e_{diff} is computed as $s' - s$ passed through a single rectified linear layer of output size 96.

Finally, e_x and $e_{(s,s')}$ are concatenated and passed through a final linear layer to obtain $A_{(s,s')}(x, a)$ and $V_{(s,s')}(x)$.

To prevent the skill from changing rapidly as it begins to converge on the optimal policy, we keep a sliding window estimate of its success rate $p_{success}$. At each timestep, with probability $1 - p_{success}$, we sample a batch of (x, a, r, x') tuples for transition (s, s') from the replay buffer and update the policy according the DDQN loss function: $\mathcal{L} = ||Q_{(s,s')}(x, a) - \text{target}||_2^2$, where target $= (r + Q^{\text{target}}(x', \arg\max_{a' \in \mathcal{A}} Q_{(s,s')}(x', a')))$. Additionally, since the rewards are intrinsically given, the optimal q-value is known to be between 0 and R_{min}. We increase stability by clipping target between these values.

Pixel blindness. In addition, some skills are easy to learn (e.g. move a few steps to the left) and don't require pixel inputs to learn at all. To prevent the skills from unnecessarily using millions of parameters for these easy skills, as an optimization, the worker first attempts to learn *pixel-blind* skills for simple transitions (s, s') with $d(s, s') = 1$ (i.e. (s, s') was directly observed by the discoverer). The pixel-blind skills only compute $e_{(s,s')}$ and pass this through a final layer to compute the advantage and value functions. If the worker fails to learn a pixel-blind skill, (e.g. if the skill actually requires pixel inputs, such as jumping over a monster) it will later try to learn a pixel-aware skill instead.

Epsilon schedule. The skills use epsilon-greedy exploration, where at each timestep, with probability ϵ, a random action is selected instead of the one produced by the skill's policy (Watkins, 1989). Once a skill becomes frozen, ϵ is permanently set to 0.

The number of episodes required to learn each skill is not known in advance, since some skills require many episodes to learn, while other skills learn in few episodes. Because of this, using an epsilon schedule that decays over a fixed number of episodes, which is typical for many RL algorithms, is insufficient. If epsilon is decayed over too many episodes, the simple skills waste valuable training time making exploratory actions, even though they've already learned near-optimal behavior. In contrast, if epsilon is decayed over too few episodes, the most difficult skills may never observe reward, and may consequently fail to learn. To address this, we draw motivation from the doubling trick in online learning Auer et al. (1995) to create an epsilon schedule, which accomodates skills requiring varying number of episodes to learn. Instead of choosing a fixed horizon, we decay epsilon over horizons of exponentially increasing length, summarized in Figure 5. This enables skills that learn quickly to achieve low values of epsilon early on in training, while skills that learn slowly will later explore with high values of epsilon over many episodes.

Count-based exploration. Our skill additionally use count-based exploration similar to Tang et al. (2017); Bellemare et al. (2016) to learn more quickly. Each skill maintains a count of the number of times visit(s) it has visited each abstract state s. Then, the skill provides itself with additional intrinsic reward to motivate itself to visit novel states, equal to $\frac{\beta}{\sqrt{\text{visit}(s)}}$ each time it visits abstract state s. We choose $\beta = 0.63$.

Self-imitation. When learning to traverse difficult obstacles (e.g. jumping over a disappearing floor), the skill may observe a few successes long before successfully learning a policy to reliably traverse the difficult obstacle. We use a variant of the self-imitation described in (Oh et al., 2018) to decrease this time. Whenever a skill successfully traverses a transition, it adds the entire successful trajectory to a separate replay buffer and performs imitation learning on the successful trajectories. These successful trajectories are actually optimal skill trajectories because the skill episode uses

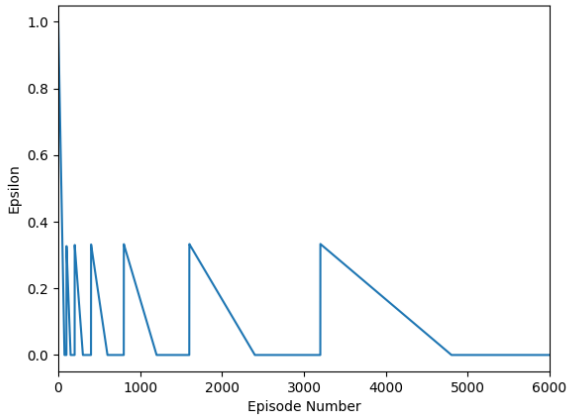
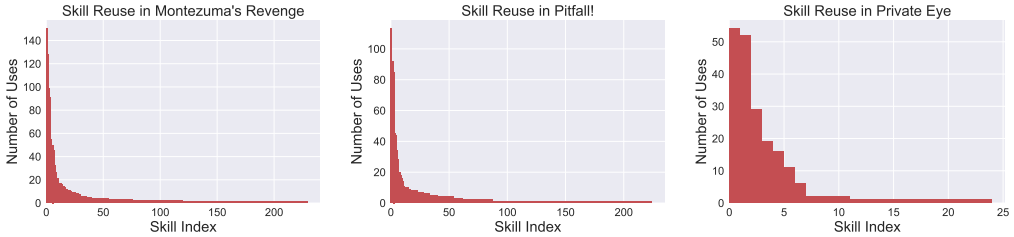


Figure 5: The saw-tooth epsilon schedule used by our skills



(a) (b) (c)

Figure 6: Number of transitions in the abstract MDP that correspond to each skill in MONTEZUMA'S REVENGE, PITFALL!, and PRIVATE EYE. Skills are sorted by decreasing usage.

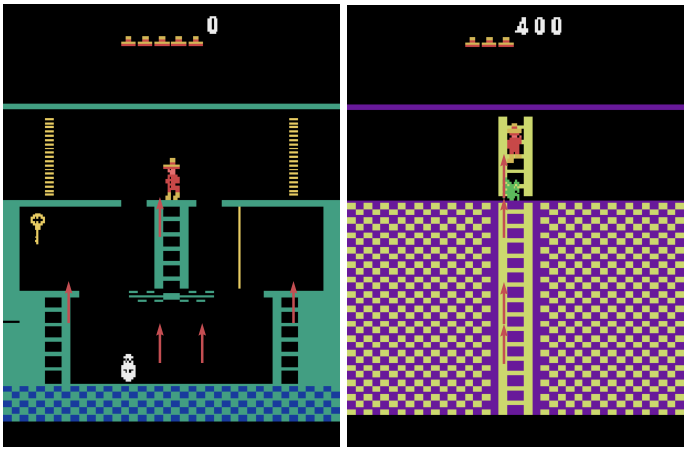
undiscounted reward, so all successful trajectories are equally optimal. To update on these skills, the skill periodically samples from this replay buffer and updates on an imitation loss function $\mathcal{L}_{imitation}(\theta) = \mathcal{L}_1(\theta) + \mathcal{L}_2(\theta)$, where θ is the skill's parameters, and $\mathcal{L}_1$ and $\mathcal{L}_2$ are defined as below:

- Let $G_t = \sum_{i=t}^{T} r_t$ be the returns to-go for a successful trajectory $(x_0, a_0, r_0), \cdots, (x_T, a_T, r_T)$. $\mathcal{L}_1$ directly regresses $Q_{(s,s')}(x, a)$ toward the reward to go of the successful trajectory, because G_t is actually the optimal q-value on successful trajectories. That is $\mathcal{L}_1 = ||G_t - Q_{(s,s')}(x_t, a_t)||_2$.
- We use the margin-loss from Hester et al. (2018) for $\mathcal{L}_2$. When sampling a transition (x, a_E, r, x'), $\mathcal{L}_2 = \max_{a \in \mathcal{A}}[Q_{(s,s')}(x, a) + \lambda \mathbf{1}[a = a_E]] - Q_{(s,s')}(s, a_E)$. Intuitively, $\mathcal{L}_2$ encourages the skill to replay the actions that lead to successful trajectories over alternate actions. We use $\lambda = 0.5$, which was chosen with no hyperparameter tuning.

B ADDITIONAL RESULTS

B.1 SKILL SHARING

The worker learns skills that successfully apply in to many similar transitions. Figure 6 depicts the number of different transitions each skill is used on in MONTEZUMA'S REVENGE, PITFALL!, and PRIVATE EYE. The simplest skills (e.g. move to the left) enjoy the highest number of reuses, while more esoteric skills (e.g. jump over a particular monster) are only useful in few scenarios.



(a) (b)

Figure 7: Skill reuse in MONTEZUMA'S REVENGE. The same skill can climb up multiple ladders and jump over a monster.

(a) (b) (c)

Figure 8: Comparison of our approach with the prior non-demonstration state-of-the-art approach in MONTEZUMA'S REVENGE, PITFALL!, and PRIVATE EYE.

Figure 7 additionally visualizes an example of a reused skill in MONTEZUMA'S REVENGE. The arrows denote the movement of the agent when it executes the skill. The same skill that jumps over a monster in the first room is reused in multiple rooms to climb up ladders as well. In Figure 7(b), the skill appears to know how to climb up all parts of the ladder except for this middle. This occurs because the spider occasionally blocks the middle of the ladder, and a different special skill must be used to avoid the spider.

B.2 GENERALIZATION TO NEW TASKS

We evaluate on three new reward functions in MONTEZUMA'S REVENGE:

- *Only key:* the agent receives 1100 reward for picking up the key in room 14 (6 rooms away from the start). In addition, the agent receives -100 reward for picking up any other objects or opening any other doors.
- *Kill spider:* the agent receives 1000 reward for killing the spider in room 13 (5 rooms away from the start). To kill the spider, the agent must first pick up the sword in room 6 (3 rooms away from the start) and save the sword for the spider. The agent receives no other reward.
- *Enter room 8:* the agent receives 1000 reward for entering room 8 (6 rooms away from the start). The agent receives no other reward.

The results are summarized in Figure 8. Even across 4 seeds, *DQN-CTS* fails to learn at all in near 50M frames. By contrast, our approach succeeds in generalizing to a new reward function not seen

during training, with only 1M additional frames to observe the new rewards. Our approach achieves near-optimal reward in roughly 75% of the seeds on each task. It should be noted that the results are not direct comparable, since our approach interacted with the environment more during the initial training phase on a different reward function. However, being able to quickly generalize to new tasks is attractive, even if the initial learning may be slow.

C GUARANTEES FOR NEAR OPTIMALITY

We require certain strong assumptions to provably achieve a near-optimal policy on the original MDP in time polynomial in the size of the abstract MDP. Notably, our approach doesn't break down when these assumptions are violated, but it no longer guarantees near optimality. Given these assumptions, proof of near optimality is elementary. We prove this result in three main steps:

1. We show that our approach achieves a near-optimal policy on the current abstract MDP at all times.
2. We show that if the abstract MDP contains all abstract states, then a near-optimal policy on the abstract MDP maps to a near-optimal policy on the original MDP.
3. We show that with high probability, the abstract MDP grows to cover all abstract states of the original MDP in time polynomial in the size of the abstract MDP.

Step 1. Our approach achieves near optimality over the current abstract MDP because the manager holds an accurate model of the abstract MDP. If the manager's model is sufficiently accurate, then the manager can recover a near-optimal policy via planning (e.g. value iteration). The manager can achieve arbitrarily high model accuracy by setting $N_{transition}$ to be large.

Step 2. In general, near optimality on the original MDP is not guaranteed, because the learned skills in the abstract MDP may be suboptimal. Because of this, hierarchical reinforcement learning literature typically focuses on hierarchical optimality (Dietterich, 2000) optimality given the abstractions (i.e. optimality in the abstract MDP). However, if we have the strong assumption of *path independence*, where all trajectories leading to an abstract state s achieve the same reward, we can trivially map near optimality on the abstract MDP to near optimality on the original MDP, if the abstract MDP contains all abstract states. We note that the optimal policy on the original MDP must terminate at some state x. Since the abstract MDP contains all abstract states (i.e. all abstract states are contained in the safe set), in particular, the manager can navigate to abstract state $\phi(x)$. By path independence, the manager's path to $\phi(x)$ achieves the same reward as the optimal policy's trajectory.

Importantly, path independence holds under many natural abstraction functions. In the Atari games we evaluate on, the state abstraction function captures the agent's inventory and a history of the agent's inventory. Since all reward in these games is given when the agent picks up new items, or uses an item in its inventory, the agent's inventory and history encodes path independent reward. This also holds for many robotic arm manipulation tasks. For example, in a block stacking task with sparse rewards, a natural state abstraction might be the location of all the blocks. Then, the reward of a trajectory is encoded by the last abstract state of the trajectory: all trajectories that lead to a stacked configuration of blocks achieve the same reward for a success, while all non-stacking trajectories achieve the same failure reward.

Step 3. Finally, we detail the number of samples required for the abstract MDP to cover all abstract states. This relies on the discoverer to discover all neighboring transitions at each abstract state and for the worker to learn the discovered transitions.

If we appropriately size N_{visit}, the number of times the discoverer explores around each abstract state, we can guarantee that the discoverer all outgoing transitions from each abstract state. In particular, let p be the probability that the discoverer fails to discover a particular abstract state on a single discoverer episode and let K be the maximum number of outgoing transitions from an abstract state (maximum degree). Both p and K are polynomial in the size of the abstract state space. Given this, it is elementary to show that if N_{visit} is set to $\frac{\log K|S| + \log \frac{1}{\delta}}{\log \frac{1}{p}}$, then the discoverer discovers

all transitions with probability at least $1 - \delta$. This follows because there are at most $K|S|$ total transitions to discover and the discoverer fails to discover each transition with probability $p^{N_{visit}}$. Applying the union bound, the probability the discoverer fails to discover at least one transition is at most $K|S|p^{N_{visit}}$. The desired result follows by plugging in for N_{visit}. Consequently, the discoverer explores for $O(\frac{\log K|S| + \log \frac{1}{\delta}}{\log \frac{1}{p}})$ timesteps.

Next, we bound the time required for the worker to learn all the transitions. We assume that function approximation can learn each transition at least as quickly as brute-force search over deterministic policies. Consequently, we bound the time required to learn each transition by $|\mathcal{A}|^H$, where H is the maximum worker horizon equal to $d_{max} \times H_{worker}$.

Finally, we compute the time used by the worker to traverse to the fringes of the safe set when learning new transitions or when the discoverer discovers new abstract states. The total time to construct the abstract MDP is $O(\sum_{s \in \mathcal{S}} N(s))$ where $\mathcal{S}$ is the abstract state space and $N(s)$ is the number of times the worker visits state s at the endpoint of a transition, counted when the abstract MDP is complete. We claim that $N(s) \leq N_{visit} + |E(G_s)|(|A|^H + N_{visit})$, where $E(G_s)$ is the set of edges in the subgraph of the abstract MDP consisting of directed transitions from s. This holds by strong induction on $|E(G_s)|$. In the base case, when $|E(G_s)|$ is 0, s is only visited N_{visit} times for the discoverer to explore. In the inductive case, supppose $|E(G_s)| = c$ and that the inductive hypothesis holds for all values less than c. $N(s)$ is at most $N_{visit} + \sum_{(s,s') \in E(G_s)} N(s')$. Since $E(G_s) = \bigcup_{(s,s') \in E(G_s)} E(G_{s'}) \cup \{(s, s')\}$, the inductive hypothesis holds for each $N(s')$. Plugging in gives the desired result.

Altogether, under relatively strong assumptions, our approach provably learns a near-optimal policy in time polynomial in the size of the abstract MDP.