Reward-predictive representations generalize across tasks in reinforcement learning

Lucas Lehnert[1,*], Michael L. Littman[1], Michael J. Frank[2,3],

1 Computer Science Department, Brown University, Providence, RI 02912, USA
2 Department of Cognitive, Linguistic & Psychological Sciences, Brown University, Providence, RI 02912, USA
3 Carney Institute for Brain Science, Brown University, Providence, RI 02912, USA

* lucas_lehnert@brown.edu

Abstract

In computer science, reinforcement learning is a powerful framework with which artificial agents can learn to maximize their performance for any given Markov decision process (MDP). Advances over the last decade, in combination with deep neural networks, have enjoyed performance advantages over humans in many difficult task settings. However, such frameworks perform far less favorably when evaluated in their ability to generalize or transfer representations across different tasks. Existing algorithms that facilitate transfer typically are limited to cases in which the transition function or the optimal policy is portable to new contexts, but achieving "deep transfer" characteristic of human behavior has been elusive. Such transfer typically requires discovery of abstractions that permit analogical reuse of previously learned representations to superficially distinct tasks. Here, we demonstrate that abstractions that minimize error in predictions of reward outcomes generalize across tasks with different transition and reward functions. Such reward-predictive representations compress the state space of a task into a lower dimensional representation by combining states that are equivalent in terms of both the transition and reward functions. Because only state equivalences are considered, the resulting state representation is not tied to the transition and reward functions themselves and thus generalizes across tasks with different reward and transition functions. These results contrast with those using abstractions that myopically maximize reward in any given MDP and motivate further experiments in humans and animals to investigate if neural and cognitive systems involved in state representation perform abstractions that facilitate such equivalence relations.

Author summary

Humans are capable of transferring abstract knowledge from one task to another. For example, in a right-hand-drive country, a driver has to use the right arm to operate the shifter. A driver who learned how to drive in a right-hand-drive country can adapt to operating a left-hand-drive car and use the other arm for shifting instead of re-learning how to drive. Despite the fact that both tasks require different coordination of motor skills, both tasks are the same in an abstract sense: In both tasks, a car is operated and there is the same progression from 1st to 2nd gear and so on. We study distinct algorithms by which a reinforcement learning agent can discover state representations that encode knowledge about a particular task, and evaluate how well they can

generalize. Through a sequence of simulation results, we show that state abstractions that minimize errors in prediction about future reward outcomes generalize across tasks, even those that superficially differ in both the goals (rewards) and the transitions from one state to the next. This work motivates biological studies to determine if distinct circuits are adapted to maximize reward vs. to discover useful state representations.

1 Introduction

A central question in reinforcement learning (RL) [1] is which representations facilitate re-use of knowledge across different tasks. Existing deep reinforcement learning algorithms, such as the DQN algorithm [2], construct latent representations to find a reward-maximizing policy in tasks with complex visual inputs. While these representations may be useful for abstracting across states in the service of optimal performance in a specific task, this article considers representations that facilitate re-use across different tasks. Humans are adept at such flexible transfer. As a lay example, consider shifting gears in a manual transmission car. In a right-hand-drive country, the steering wheel is on the left side of the car and the right arm is used for shifting, whereas the opposite is the case in a left-hand-drive country. A person who has learned in one scenario can quickly generalize to the other, despite the fact that both tasks require different coordination of motor skills. Both tasks are the same in an abstract sense: In each case, there is a progression from 1st to 2nd gear and so on, which should be coordinated with the clutch pedal and steering, and this structure can be generalized from a left-hand-drive car to a right-hand-drive car [3, 4] and a driver does not have to learn how to drive from scratch.

By treating two different sensory inputs or states as equivalent, an agent can generalize what it has learned from one state to another and speed up learning [4, 5]. Such equivalences can be modeled using *state abstractions* [3, 6], which map states to a compressed latent representation. The usual RL framework considers Markov Decision Processes (MDPs) [7] in which an agent's sole objective is to maximize reward. In contrast, in transfer or lifelong learning, an agent observes a sequence of MDPs and attempts to learn a state abstraction that can be re-used to speed up learning in a previously unseen task. State abstractions can be constructed in different ways, for example by merging states with the same optimal action or Q-values into the same latent or *abstract state*. This article considers two types of state abstractions:

1. *reward-maximizing state abstractions*, which allow an agent to maximize total reward, and

2. *reward-predictive state abstractions*, which allow an agent to predict future reward outcomes.

While many different RL transfer algorithms have been proposed (see [8] for a survey), this article demonstrates that, while reward-maximizing state abstractions are useful for compressing states within a given task, they fail to generalize across tasks that differ in reward and transition functions. In contrast, reward-predictive state abstractions can be leveraged to improve generalization even when both transition and reward functions change across tasks. The presented analysis and simulations motivate the design of new RL algorithms that can discover such state abstractions as well as further experiments to investigate whether neural mechanisms in biological agents facilitate learning of such representations.

Previous work [9] shows that reward-predictive state abstractions can be extracted from the *successor representation* (SR) [10], which predicts the discounted expected frequency of visiting future states given the current state. While re-using a previously

learned SR has been shown to speed up learning when reward functions change [11–13], these methods are only suitable if transition functions are shared (e.g., if one is in the same maze but only the location of the goal changes). Further, if the optimal decision-making strategy differs between two tasks, the SR has to be re-learned [14]. In contrast, this article shows that reward-predictive state abstractions afford "zero-shot" transfer across tasks with variations in transition functions, reward functions, and optimal policies and do not have to be adjusted or re-learned for each task. Such "deep transfer" across environments, even in the absence of prior experience with specific transition or reward functions, is predicted by behavioral and neural signatures of human structure learning [4, 15, 16] but not afforded by alternative algorithms that compress the transition function itself directly [17, 18].

To unpack the relative advantages of distinct state abstraction algorithms for generalization, we proceed as follows. In Section 2, we begin with a simple illustration of the state abstraction framework and then present the conceptual utility of reward-predictive state abstractions in Section 3. Next, we examine this advantage quantitatively when a single abstraction is possible for re-use across a range of task settings and assumptions about the number of latent states (Section 3.1). Subsequently, Section 3.2 considers a curriculum learning situation where multiple state abstractions might apply to different MDPs and the agent has to select amongst them when learning a new MDP. Finally, in Section 3.3 we show that this advantage is preserved even when the agent has to simultaneously learn the transitions and rewards of the new MDP and perform inference, and elaborate the conditions under which this is the case.

2 Generalization across states

An MDP [1] is a quintuple $M = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$ with a state space $\mathcal{S}$, an action space $\mathcal{A}$, a transition function $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, and a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. The interaction of an agent in a particular task is modeled by a sequence of transitions between different states. Each transition from state s to state s' is initiated by an action $a \in \mathcal{A}$ and is rewarded with a scalar number $r(s, a, s')$. The probability of reaching state s' from state s using action a is specified by the transition function $p(s, a, s')$. How strongly short-term rewards are favored over long-term rewards is controlled by the discount factor $\gamma \in [0, 1)$.

In *model-free learning*, for example Q-learning [19], the optimal decision-making strategy, called a *policy*, is learned through trial and error interactions in an MDP. Throughout these interactions, a policy is incrementally improved. During learning, only the policy π and some form of cached values of the policy π are stored at any point in time. In other words, the agent only learns and represents the net predicted reward value of an action in a given state, without needing to represent the specific outcomes of each action in terms of the subsequent states that will be encountered. In *model-based learning* [20, 21] an agent attempts to build a model of the task's transition and reward function and uses this model to predict sequences of future reward outcomes $(r_1, r_2, ...)$ given a start state s and a particular sequence of actions $(a_1, a_2, ...)$. While more computationally intensive, this approach is orthogonal to model-free learning, because using this model an RL agent can predict the value of any arbitrary policy, and it can flexibly adjust its policy if the reward changes. In this case, the agent's "knowledge" is sufficient to generalize across the space of all possible policies [20].

For example, in an Atari game [22], the number of states in an MDP is large and it may be inefficient to learn and memorize an optimal action for every possible image pixel configuration. In this case, state abstractions [6, 23] provide a framework for simplifying the input space into a lower-dimensional latent space. A state abstraction, also called state representation, is a function ϕ mapping the state space $\mathcal{S}$ to some other

latent space. Because state representations are many-to-one relations, they can map different states to the same latent state and create a partitioning of the state space. A state partition is a subset of the state space that maps to the same latent or abstract state. An agent using a state abstraction ϕ operates on the space of state partitions and generalizes knowledge learned in one state across the entire state partition. For example, in Q-learning, a value update is applied to the entire state partition even if the update is computed only from one specific state transition, resulting in faster learning if the state abstraction is appropriate [4, 5].

While several approaches exist for constructing a useful state abstraction ϕ for complex MDPs (e.g., Atari games), this article investigates how to construct state abstractions that facilitate re-use across different tasks. Specifically, we consider the question of which state abstraction ϕ should be learned from a hypothesis space of all possible state abstractions $\mathcal{H}$, in such a way that would maximize the agent's ability to reuse knowledge in future tasks. A state abstraction is a function mapping states to a smaller latent abstract state space $\mathcal{S}_\phi$. The state-abstraction hypothesis space is then

$$\mathcal{H} = \{\phi : \mathcal{S} \to \mathcal{S}_\phi\} \tag{1}$$

and a representation learning algorithm searches this space to identify a state abstraction ϕ. An agent that uses a state abstraction ϕ operates directly on the latent space $\mathcal{S}_\phi$ rather than the underlying state space $\mathcal{S}$. Depending on how ϕ constructs the latent state space $\mathcal{S}_\phi$, the agent may or may not be able to distinguish between a rewarding and a non-rewarding state.

Fig 1 presents an example of how state representations simplify a task. The left portion of the schematic presents a grid-world navigation task where an agent only receives reward by entering the right column. For this task, the centre schematic illustrates a reward-predictive state representation that generalizes across different columns, as indicated by the colouring. In this case, the task is compressed into a smaller 3×1 grid world where a reward is given for entering the latent state $\boldsymbol{\phi}_3$ (green) and no reward is given for the latent states $\boldsymbol{\phi}_1$ (blue) and $\boldsymbol{\phi}_2$ (orange). While the compressed version of the grid world does not preserve all information about the task, it still possible to predict future reward outcomes. For example, the path indicated by the black boxes is mapped to a latent state sequence of $(\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \boldsymbol{\phi}_2, \boldsymbol{\phi}_3)$. This latent state sequence could then be mapped to a reward sequence of $(0, 0, 0, 1)$. In this example, every possible path in the original task is mapped to a path in the compressed task that produces the same reward sequence. Hence, the smaller compressed task can be used to predict future reward outcomes of the original task and the state representation is thus reward predictive.

A reward-predictive state abstraction $\phi \in \mathcal{H}$ allows an agent to best predict which expected reward sequence $r_1, ..., r_t$ will be observed after executing a decision sequence $a_1, ..., a_t$ starting at a specific state s. If the random variable R_t describes the reward that is observed after following the action sequence $a_1, ..., a_t$ starting at state s, then the expected reward sequence is

$$(r_1, ..., r_t) = \mathbb{E}[(R_1, ..., R_t) \mid s, a_1, ..., a_t]. \tag{2}$$

The expectation in Eq (2) is conditioned on the start state s and is computed over all possible trajectories in an MDP that follow the action sequence $a_1, ..., a_t$. A reward-predictive state abstraction needs to satisfy for any start state s and action sequence $a_1, ..., a_n$ that

$$\mathbb{E}[(R_1, ..., R_t) \mid s, a_1, ..., a_t] = \mathbb{E}[(R_1, ..., R_t) \mid \phi(s), a_1, ..., a_t]. \tag{3}$$

The expectation on the right-hand side is conditioned on the latent state $\phi(s)$. Consequently the expectation on the right-hand side is computed over all possible

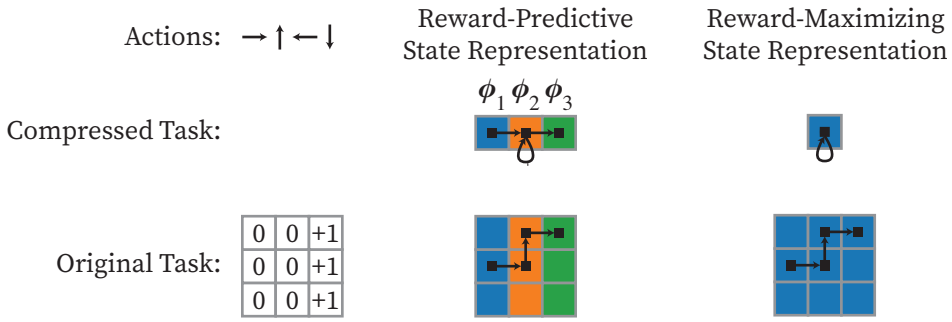


Fig 1. State-Abstraction Examples, adopted from [9]. The left part of the schematic shows a 3×3 grid world where an agent can move up (↑), down (↓), left (←), or right (→). A reward of +1 is given when the right column is entered from the centre column by selecting the action "move right" (→). The centre schematic illustrates a reward-predictive state representation, which generalizes across columns (but not rows) and compresses the 3×3 grid world into a 3×1 grid world with three latent states labelled with $\boldsymbol{\phi}_1$, $\boldsymbol{\phi}_2$, and $\boldsymbol{\phi}_3$. In this compressed task, only the transition moving from the centre orange state $\boldsymbol{\phi}_2$ to the right green state $\boldsymbol{\phi}_3$ is rewarded. The right schematic illustrates a reward-maximizing state representation, which compresses all states into one latent state. In the 3×3 grid, there are three out of nine locations where an agent can receive a reward by selecting the action move right (→). If states are averaged uniformly to construct the one-state compressed task, then the move right action is rewarded with 1/3 and all other actions are not rewarded. In this case, an optimal policy can still be found using the compressed task, but accurate reward predictions are not possible.

trajectories in a latent space, while the expectation on the left-hand side is computed over all possible trajectories in an MDP. A reward-predictive state abstraction constructs a latent space such that for any start state and any arbitrary action sequence, both the compressed task and original task produce the same expected reward sequence [9]. While this model is different to learning an explicit approximation of the transition and reward function, learning a reward-predictive state abstraction is akin to model-based RL, because both systems are used to predict sequences of future reward outcomes.

The right portion of Fig 1 presents a reward-maximizing state representation. Because the optimal action is to move right in this specific task, the task can be compressed into a single state in this example. If rewards are only given when an agent enters the right column by moving right, then only the right action is directly rewarded. In this case, the compressed task can still be used to find an optimal policy, because only the move right action is rewarded; the remaining three actions are not rewarded in any case. Nevertheless, the compressed task cannot be used to make accurate predictions of future reward outcomes, because this state representation simplifies the task into only one latent state (the agent does not know which column it is in: it is as if moving right simply produces stochastic rewards). Because this state abstraction allows an agent to recover the optimal policy, this state abstraction is reward maximizing in this example.

In this article, we use Linear Successor Feature Models (LSFMs) [9] as a means to learn reward-predictive state abstractions. Successor Features (SFs) [47] are a generalization of the SR [10], and predict the expected visitation frequencies in some latent feature space:

$$\boldsymbol{\psi}^{\pi}(s,a) = \mathbb{E}\left[\phi(s_1) + \gamma\phi(s_2) + \gamma^2\phi(s_3) + \cdots \,\middle|\, s = s_1, a = a_1, \pi\right], \tag{4}$$

where the expectation is computed over all infinite length trajectories that start in state s with action a and then follow the policy π. The discount factor $\gamma \in [0, 1)$ is used such that states in the more distant future are weighted to a lesser degree in the summation in Eq (4). Intuitively, SFs incorporate information about which latent state features are observed along a trajectory, including their relative temporal positions, because for each

time step t a different weight γ^t is associated with the latent state feature vector $\phi(s_t)$ (Eq (4)) (allowing the state abstraction to distinguish between a reward sequence "$+1, -1$" vs. "$-1, +1$", for example). LSFMs extract this temporal property from SFs and construct a state abstraction ϕ that is predictive of the order with which particular latent state features are observed. Critically, the LSFM latent space is constructed so as to most efficiently predict reward sequences without being tied to the specific transitions or rewards, and thus permit a "deeper" form of transfer loosely akin to analogical reasoning.

If such a state abstraction also associates each feature vector $\phi(s_t)$ with one-step reward outcomes, then this state abstraction is reward-predictive. Mathematically, this intuition can be generalized to predict reward outcomes for any start state s and action sequence $a_1, ..., a_t$ and indeed, learning LSFMs is equivalent to learning reward-predictive state abstractions [9].

Note that this LSFM approach contrasts with the typical application of SFs in which Q-values are expressed as a dot-product between the SF vectors $\boldsymbol{\psi}^\pi(s, a)$ and a reward-model vector. While that approach allows an agent to re-use SFs when rewards and the associated reward-model vector change, it does not afford analogical transfer when transitions change. In fact, because SFs depend on the transition function and a particular policy, a transferred SF has to be relearned and adjusted to a specific task. Fig 2 presents an intuitive transfer example and plots different SFs for each task. Consequently, an agent would have to adjust a previously learned SF.

3 Generalization across tasks

To generalize knowledge across different tasks, a compressed state abstraction is needed that preserves key aspects of the tasks even if the details of transitions or rewards change. Consider the transfer example in Fig 2, where an agent is first presented Task A, and then transfers a state abstraction to Task B. The key similarity between the tasks is evident in that they both have columnar structure, but transitions, rewards, and the optimal policy can all differ. In this example, the reward-predictive state abstraction can be re-used to plan a different policy in Task B, while the reward-maximizing state abstraction cannot be re-used in Task B. Of course, such a benefit is only possible if the two tasks share an abstract relation: This columnar state abstraction would not be useful in subsequent MDPs that arranged in rows. Below, we consider how multiple state abstractions can be learned and where generalization involves an inference process to select which one of them is most applicable [3].

In principle, there always exists one state abstraction that is both reward-maximizing and reward-predictive in a model-based agent: Trivially, if the identity map is used to map nine distinct states into a latent space of nine distinct states, then such a state representation is always reward maximizing and reward predictive. However, such a state representation is not really "abstract" in that it does not inform an agent across which states information can be generalized. But, for the same reason, this representation preserves information that might be needed in other tasks. We will further discuss this trade-off in Section 3.3.

3.1 Transfer with single state abstractions

The above example was illustrative for a single MDP designed to show the potential utility of reward-predictive state abstractions. We next systematically assess the generalization potential of reward-maximizing or reward-predictive state abstractions across a range of different tasks. The goal of this experiment is to be algorithm agnostic: Rather than focusing on how a particular algorithm performs at transfer with a single

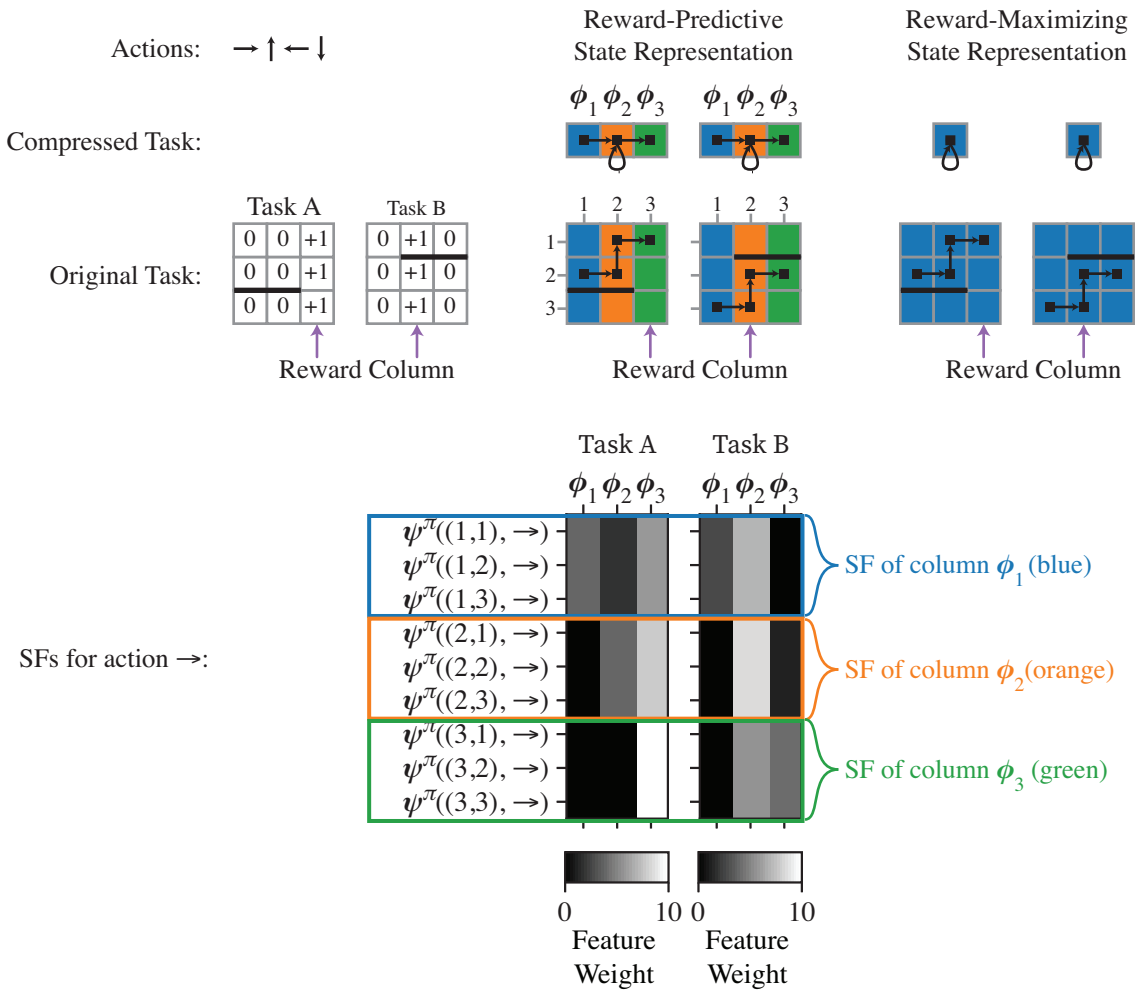


Fig 2. Transferring State Abstractions between MDPs. In both grid-world tasks, the agent can move up (↑), down (↓), left (←), or right (→) and is rewarded when a reward column is entered. The black line indicates a barrier the agent cannot pass. Both Task A and Task B differ in their rewards and transitions, because a different column is rewarded and the barrier is placed at different locations. In the centre of the schematic a reward-predictive state representation is illustrated for both tasks and the corresponding SFs are plotted below. Each row in these matrix plots (bottom two panels) visualizes the entries of a three dimensional SF vector. Similar to the example in Fig 1, a reward-predictive state abstraction merges each column into one latent state, as indicated by the colouring. In both tasks, reward sequences can be predicted using the compressed representation for any arbitrary start state and action sequence, similar to the centre schematic in Fig 1. In this case the agent simply needs to learn a different policy for Task B using the same compressed representation. In contrast, the matrix plots in the bottom panels illustrate that SFs are different in each task and cannot be immediately reused in this example (because SFs are computed for the optimal policy which is different in each task [14]). Note that states that belong to the same column have equal SF weights (as indicated by the coloured boxes). LSFMs construct a reward-predictive state representation by merging states with equal SFs into the same state partition. This algorithm is described in Section S.3.4 and prior work [9]. The schematics on the right show one possible reward-maximizing state abstraction. While it is possible to learn or compute the optimal policy using this state abstraction in Task A (i.e., always go right), this state abstraction cannot be used to learn the optimal policy in Task B in which the column position is needed to know whether to go left or right. This example illustrates that reward-predictive state representations are suitable for re-use across tasks that vary in rewards and transitions. While reward-maximizing state abstractions may compress a task further than reward-predictive state abstractions, reward-maximizing state abstractions may also simplify a task to an extend that renders them proprietary to a single specific task.

learned state abstraction, we enumerated the entire hypothesis space $\mathcal{H}$ for all possible partitions of the state space and evaluated them in all transfer tasks. For each state abstraction in $\mathcal{H}$, we computed a compressed abstract MDP [23] for every tested MDP and solved it using value iteration [1, Chapter 4.4] (please also refer to Section S.1). A reward-maximizing state abstraction is then identified by testing the computed policy in a single randomly selected task for N trials over T time steps and computing the total reward

$$R_{\text{total}} = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} r_{n,t}, \tag{5}$$

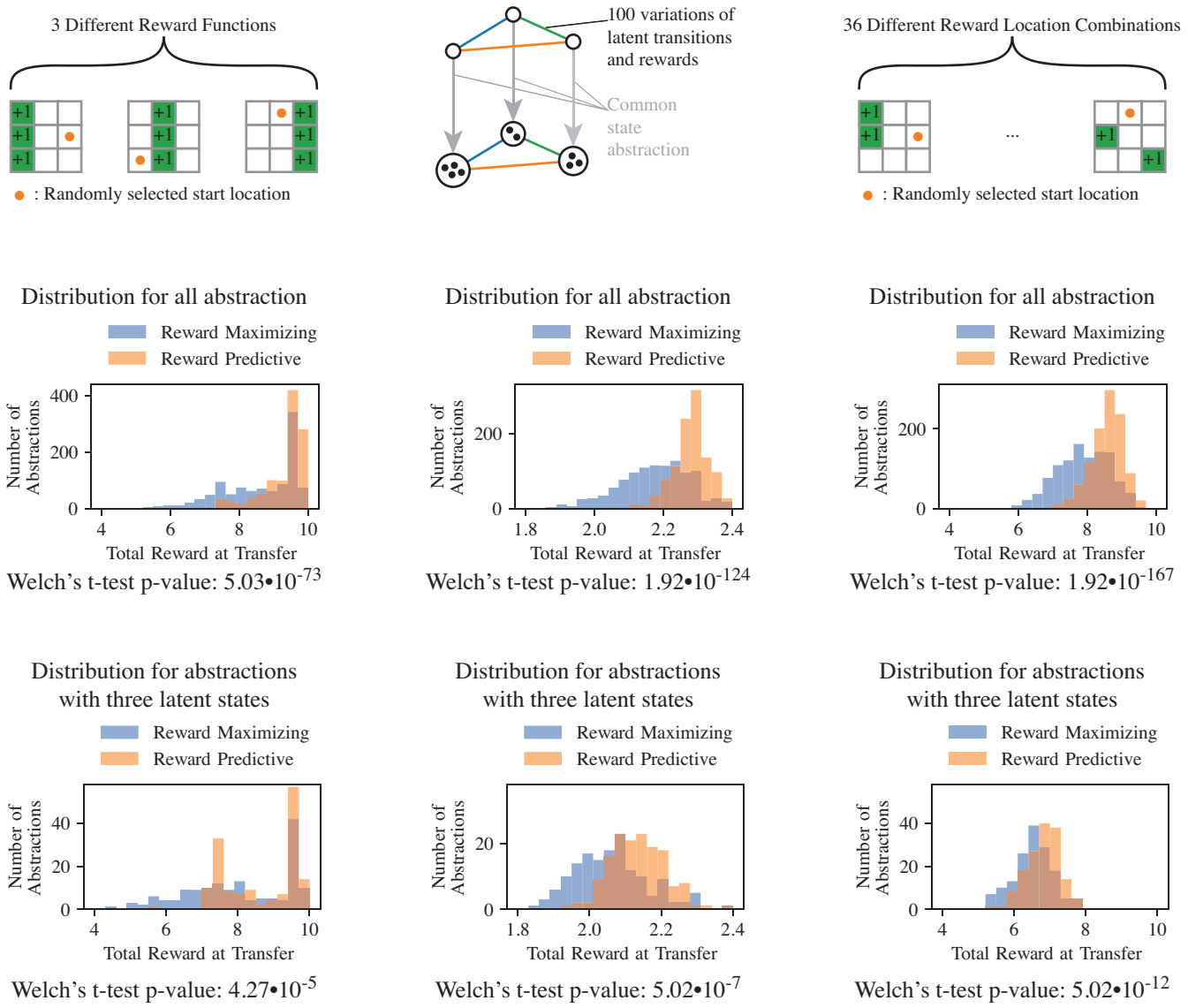
where $r_{n,t}$ is the reward incurred in trial n at time step t. A reward-predictive state abstraction is identified by sampling N random state-and-action sequence pairs $(s_n, a_{n,1}, ..., a_{n,T})$ and predicting the reward sequence $\hat{r}_{n,1}, ..., \hat{r}_{n,T}$ using the abstract MDP. The reward-sequence prediction error is

$$RS_{\text{error}} = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} |r_{n,t} - \hat{r}_{n,t}|. \tag{6}$$

We considered three types of tasks: column-worlds (like those in the motivating example), 100 randomly generated MDPs, and grid worlds (Fig 3). For each transfer experiment, the top 5% scoring state abstractions were re-evaluated on the remaining transfer MDPs and the total rewards generated by these state abstractions are plotted as histograms in Fig 3. In all cases, state abstractions with low reward-sequence prediction errors RS_{error} generate a higher total reward at transfer than state abstractions that were selected based on their ability to construct a well performing policy (produce a high R_{total} score) on the original MDP. Note that restricting the hypothesis space $\mathcal{H}$ to abstractions that construct three latent states (second row of histograms in Fig 3) does not change the overall result. This result indicates that reward-predictive state abstractions encode information about an MDP that can be generalized across different MDPs that share the same abstract structure. In Sections 3.2 and 3.3, we will present an extension to environments in which multiple structures are possible and have to be inferred.

Fig 3(a) shows the results for the transfer experiment discussed in Fig 2. Both histograms in Fig 3(a) indicate that state abstractions with low reward-sequence prediction errors outperform on average state representations that only maximize total reward in one of the tasks. Because all three MDPs can be compressed into three latent states, constraining the hypothesis space $\mathcal{H}$ to only contain state abstractions that create three latent states does not impact the total reward generated at transfer time significantly. In this case, both histograms have equal support.

To further control for a potential dependency between the constructed MDPs and a particular state-representation type, the experiment in Fig 3(b) randomly generates transition and rewards. This experiment is similar to the previous test case in that all 100 randomly generated MDPs can be compressed with the same state representation. These MDPs are constructed by generating random three-state MDPs and then "inflating" the state space size based on this common but randomly generated state abstraction. Aside from this common "hidden" state representation, these 100 MDPs differ in both transition and reward functions. The histograms confirm the claim that abstractions yielding low reward-sequence prediction errors perform best in generalization across different MDPs. In contrast, state representations that result in high total reward in any of the original MDPs generate on average less reward in any of the remaining MDPs. Again, constraining the hypothesis space $\mathcal{H}$ to only include abstractions that construct three latent states does not change the support of the histogram, but the shape changes and the median shifts. This shift in the bottom



(a) Column World Transfer (b) Random MDP Transfer (c) Grid-World Navigation Transfer

Fig 3. Minimizing reward-sequence prediction errors identifies state abstractions amenable for "deep transfer". For each experiment, all possible state abstractions in $\mathcal{H}$ were enumerated using Algorithm U [24] to obtain a ground truth distribution over the hypothesis space $\mathcal{H}$. State abstractions were scored by compressing an MDP using the state abstraction of interest [6]. The total reward score was computed by running the computed policy 20 times for 10 time steps in the MDP from a randomly selected start state. The reward-sequence error was computed by selecting 20 random start states and then performing a random walk for 10 time steps. The histograms report averages over all repeats and transfer MDPs for either all state abstractions that are possible in a nine state MDP (top histogram) or for all state abstractions that compress a nine state MDP into a three state MDP (bottom histogram). The Welch's t-test was performed to compute the p-values of the difference in mean total reward being insignificant for each histogram.

histogram in Fig 3(b) can be explained by the fact that incorrectly compressing a task and incurring approximation errors can quickly degrade an agent's ability to perform optimally. If a state abstraction does not maximally compress a task, for example from nine to eight states, then performance may not degrade as quickly.

The above simulation assumed lossless compression (given that the abstraction was selected and then inflated to generate larger state spaces). To test which state abstractions generalize across tasks when no "hidden" state abstraction is embedded in the tasks, we next considered situations in which state spaces could not be compressed without some information loss. Fig 3(c) presents a transfer experiment where two reward locations are permuted in a grid world. In this experiment, the MDPs cannot be compressed without incurring some loss, because the grid location is important for predicting where the goal locations are and what action is optimal at each location. However, both histograms in Fig 3(c) indicate that state abstractions that minimize the reward-sequence prediction error criterion still perform better than those that maximize total reward. By nature grid worlds have a specific topology of the state space and state representations that cluster only neighbouring states approximately preserve the grid location information and would be expected to perform relatively well across all MDPs. If the hypothesis space $\mathcal{H}$ is constrained to abstractions that compress nine states into three latent states, then the advantage shrinks. This difference can be explained by the fact that for arbitrary navigation tasks grid worlds should ideally not be compressed (for efficient navigation an agent needs to be aware of its position), and hence neither abstraction yields an optimal policy. However, the top histogram in Fig 3(c) suggests that there exist several state abstractions that compress nine states into six or seven latent states that can still lead to (close to) optimal performance.

Note that the identity map, which does not compress the state space and can always be used to construct an optimal policy, is also included in this histogram and occurs in the bin with highest total reward. The identity map is included exactly once into each histogram that plots the distribution for all abstractions, because this experiment tests each possible state abstraction once. This experiment highlights a trade-off between the ability to obtain an optimal policy in a task and re-use of a particular state abstraction that compresses a task.

3.2 Transfer with Multiple State Abstractions

The previous experiment assumes that all tasks share a common "hidden" state abstraction that can be learned and re-used by an agent. In this section, we consider the situation in which different MDPs might correspond to different abstractions. A non-parametric Bayesian model maintains a belief space of possible state abstractions [3, 4], which it can use for inference. Fig 4 illustrates how the curriculum of tasks is randomly generated. Each task can be compressed in one of two different ways (this approach can be expanded to larger numbers without loss of generality; two is used here for clarity of exposition). Critically, this state abstraction is hidden from the learning agent. After observing an MDP sequence $M_1, ..., M_t$, the agent updates its belief space $\mathcal{B}_t$ using a posterior over which state abstraction is most suitable to solve a given task M_t:

$$\Pr(\phi|M_t, \mathcal{B}_t, c_t) \propto \Pr(\phi|M_t)\Pr(\phi|\mathcal{B}_t, c_t), \tag{7}$$

where $c_t(\phi)$ is the count of how often an abstraction ϕ was used in the previous $t-1$ tasks. These counts are used to construct a Chinese Restaurant Process (CRP) [25, 26] prior for an intensity $\alpha \geq 0$:

$$\Pr(\phi|\mathcal{B}_t, c_t) = \begin{cases} \frac{c_t(\phi)}{t-1+\alpha} & \text{if } \phi \in \mathcal{B}_t \\ \frac{\alpha}{t-1+\alpha} & \text{otherwise.} \end{cases} \tag{8}$$

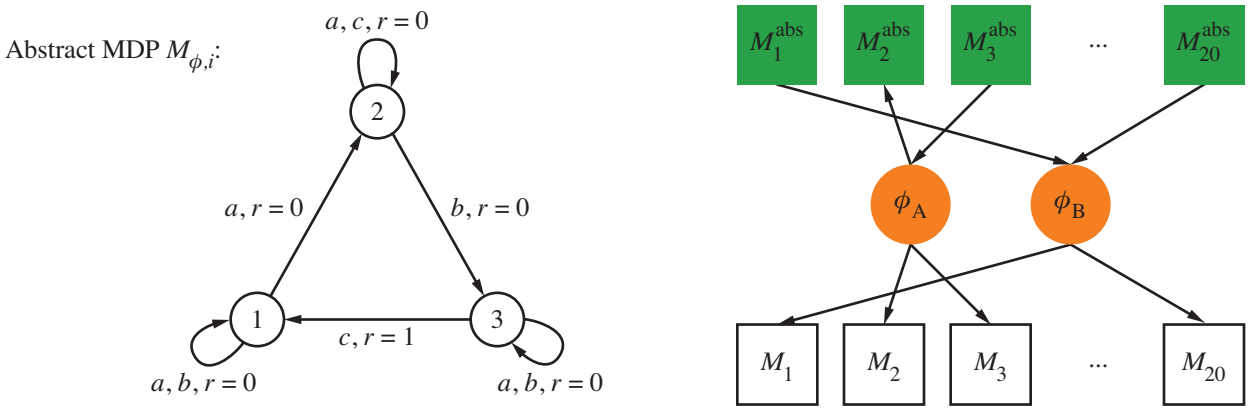


Fig 4. Transfer with Multiple State Abstractions Curriculum. A curriculum of transfer tasks is generated by first constructing the three-state MDP shown on the left. At each state, only one action causes a transition to a different state. Only one state-to-state transition is rewarded; the optimal policy is to select the correct action needed to cycle between the node states. To generate a sequence of abstract MDPs $M_1^{\text{abs}}, ..., M_{20}^{\text{abs}}$, the action labels are randomly permuted as well as the transitions generating positive reward (similar to the Diabolical Rooms Problem [3]). Two hidden state abstractions ϕ_A and ϕ_B were randomly selected to "inflate" each abstract MDP to a nine-state problem. One state abstraction was used with a frequency of 75% and the other with a frequency of 25%. The resulting MDP sequence $M_1, ..., M_{20}$ was presented to the agent, without any information about which state abstraction was used to construct the task sequence.

The posterior is also conditioned on the MDP M_t through the factor $\Pr(\phi|M_t)$. Using a loss function l (we consider both reward-maximizing and reward-predictive losses), each state abstraction ϕ can be scored and for $\beta \geq 0$, the probability of this state abstraction being suitable to solve M_t is the soft-max probability

$$\Pr(\phi|M_t) \propto e^{-\beta l(\phi)}. \tag{9}$$

To determine which state abstraction should be added into the abstraction belief set $\mathcal{B}_t$, the non-parametric Bayesian agent has access to the best scoring state abstraction $\phi_{\text{next-best}}$ not included into $\mathcal{B}_t$. The posterior $\Pr(\phi|M_t, \mathcal{B}_t, c_t)$ is computed over the set of state abstractions $\mathcal{B}_t \cup \{\phi_{\text{next-best}}\}$.[1] If α increases, the resulting prior and posterior assign a higher probability to adding the next-best state abstraction $\phi_{\text{next-best}}$ into $\mathcal{B}_t$. If β increases, then more emphasis is given on using the loss function l to determine which state abstraction should be used from the set $\mathcal{B}_t \cup \{\phi_{\text{next-best}}\}$.

Rather than using the empirical scores R_{total} or RS_{error}, the agent is allowed to observe a tabulation of all possible transitions and rewards to obtain a ground truth score for each abstraction. In these experiments, reward-maximizing state abstractions are identified by assessing how much using a state abstraction impacts the value of the policy π_ϕ relative to that of the optimal policy in the abstract MDP:

$$l_{\text{maximizing}}(\phi) = \max_{s \in \mathcal{S}} \left(V^{\pi^*}(s) - V^{\pi_\phi}(s) \right), \tag{10}$$

where V^{π^*} is the optimal discounted value function [1], π^* is the optimal policy, and V^{π_ϕ} is the discounted value function of the policy π_ϕ evaluated in the task itself.

[1] The goal is not to design an algorithm that can solve a sequence of tasks efficiently, but to analyze which state abstractions generalize across different tasks. Thus, for the moment, we assume that the agent has access to an oracle that knows the transition function of each new MDP and can score the loss for each compression. Using an oracle that tabulates the hypothesis space gives insight into which state abstractions generalize across different tasks, while being algorithm agnostic; below we relax the need for an oracle.

Reward-predictive state abstractions are scored by the loss function $l_{\text{predictive}}$ bounding the reward-sequence prediction error

$$\forall i, \forall a_1, ..., a_t,\ l_{\text{predictive}}(\phi) C_{\gamma,t} \geq |\mathbb{E}[r_t | i, a_1, ..., a_t] - \hat{r}_t|\,, \tag{11}$$

where $C_{\gamma,t}$ is a constant that depends on the action-sequence length t and discount factor γ. The loss function $l_{\text{predictive}}$ is computed using the SF model [9]. Appendix S.2 presents all details on how to compute $l_{\text{predictive}}$. If any of the two loss functions evaluates to zero for a state abstraction ϕ, then ϕ is either a globally optimal reward-maximizing or reward-predictive state abstraction.

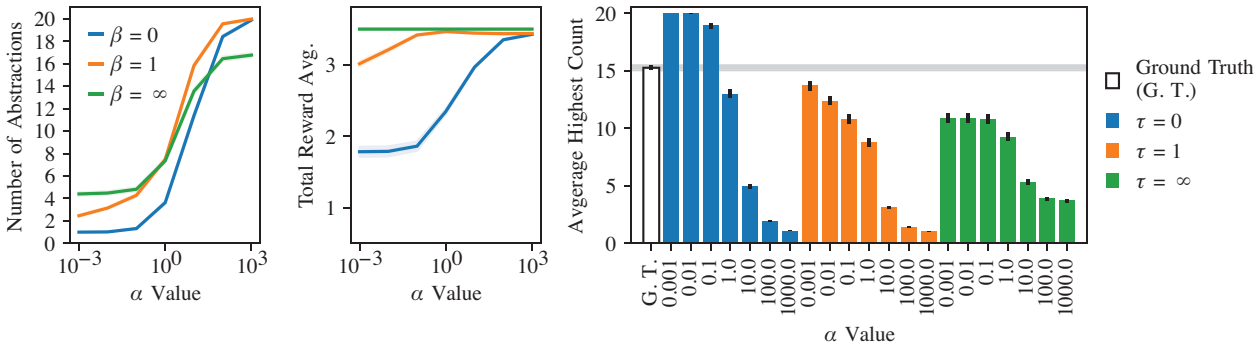
Fig 5 plots the results from testing the agent with each loss function for various α and β settings. The agent selects its policy by using the posterior to mix the policies that would be optimal in the respective abstract MDPs. Setting $\beta = \infty$ means that the probability $\Pr(\phi|M_t)$ is deterministic: The highest scoring state abstraction is assigned a probability of one and all other state abstractions are assigned a probability of zero. This case is equivalent to only using the loss function to select a state abstraction while ignoring the CRP prior $\Pr(\phi|\mathcal{B}_t, c_t)$, because the factor $\Pr(\phi|M_t)$ is either zero or one in Eq (7). For low β settings, the prior is used to determine which state abstraction is used. If α is high, then up to 20 state abstractions are added into the belief set $\mathcal{B}_t$. Because the prior influences the posterior heavily, the total reward of the resulting agent is comparably low, because the agent is not well informed about which state abstraction should be used on a given task. For $\beta = \infty$, the loss function influences the posterior strongly.

The key difference between the two loss functions becomes apparent when analyzing how the agent maintains the belief space $\mathcal{B}_t$. Using the loss function $l_{\text{predictive}}$, which identifies reward-predictive state abstractions, the agent identifies the correct ground truth state abstractions that were used to generate the task sequences. The count bar charts in Fig 5 also show that the agent correctly learns that one state abstraction occurs with a frequency of 75%. Because the agent only maintains two belief abstractions, the agent correctly estimates that the other abstraction occurs with a frequency of 25%.

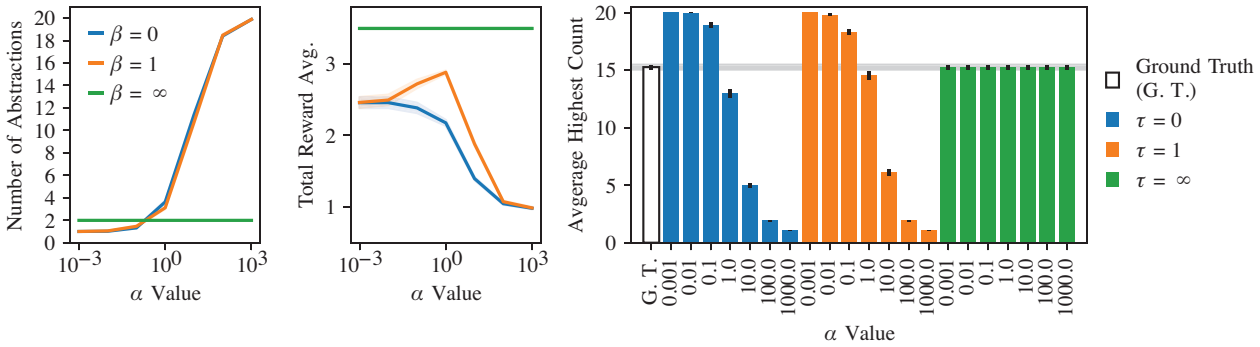
In contrast, when the loss function $l_{\text{maximizing}}$ is used, the agent can only achieve optimal reward by isolating a significantly higher number of state abstractions than the reward-predictive model. At best, using $l_{\text{maximizing}}$ and a small α value the agent is capable of isolating five state abstractions. For high α settings, the agent effectively memorizes a solution for almost every task by increasing the size of its belief set $\mathcal{B}_t$, because a previously used state abstraction does not generalize to the next task. The model is thus able to achieve optimal reward only if it constructs a new reward-maximizing abstraction for each MDP. Note that this experiment does not account for any cost associated with learning or constructing a state abstraction for each task from scratch. In Section 3.3, this assumption is removed and the presented results illustrate how constructing a reward-maximizing state abstraction results in slower learning. When using the loss function $l_{\text{predictive}}$, the agent can correctly identify which state abstraction to use for which MDP, confirming the claim that reward-predictive state abstractions generalize across different tasks.

3.3 Learning to transfer multiple state abstractions

While the previous transfer experiment presents evidence that reward-predictive state abstractions generalize across different tasks, for exposition these previously presented simulations assumed that a full tabulation of all transitions and reward is accessible for the agent to score the loss. In addition, it was possible to configure both reward-predictive and reward-maximizing models such that an optimal reward level is



(a) Reward-Maximizing State Abstractions Using $l_{\text{maximizing}}$



(b) Reward-Predictive State Abstractions Using $l_{\text{predictive}}$

Fig 5. Results for transfer with multiple state abstractions experiment. The left panels plot how different α and β model parameters influence the average size of $\mathcal{B}_t$ after training. The middle panels plot the performance of each model (average total reward per MDP) for different α and β model parameters. If $\beta = \infty$ the agent obtains an optimal total reward level when using either loss function for ten time steps in each MDP. The right panels plot the average count for the most frequently used state abstraction. As described in Fig 4, one of two possible "hidden" state abstractions, ϕ_A and ϕ_B, were embedded into each MDP. Each task sequence consists of 20 MDPs and on average 15 out of these 20 MDPs had the state abstraction ϕ_A embedded and the remaining MDPs had the state abstraction ϕ_B embedded. The white bar labelled "Ground Truth" plots the ground-truth frequency of the "hidden" state abstraction ϕ_A. If the non-parametric Bayesian model correctly detects which state abstraction to use in which task, then the average highest count will not be significantly different from the white ground truth bar. In total, 100 different task sequences, each consisting of 20 MDPs, were tested and all plots show averages across these 100 repeats (the standard error of measure is indicated by the shaded area and variations are very low if not visible).

always obtained given the agent can always construct a new reward-maximizing abstraction. This section presents an experiment where an intelligent system has to learn through trial-and-error interactions with a novel sequential decision-making task thereby simultaneously learning the transitions and making inferences about which abstraction is appropriate to reuse, in a Bayesian mixture of experts scheme [27]. If an intelligent agent is capable of extracting a particular structure from one task to accelerate learning in another task, then this agent will generate more reward in certain tasks than an agent that does not transfer any latent structure.

In the following simulation experiments, an agent is allowed to interact with a task for a certain number of trials, called *episodes*. The interaction data itself is a data set of transition quadruples of the form (s, a, r, s') that describe a transition from state s to state s' that occurs by selecting action a and is rewarded with a scalar reward r.

The generalized non-parametric Bayesian model maintains a belief space of state abstractions $\mathfrak{B}_t$ that is updated after observing a certain number of episodes in a particular task.[2] The state abstractions stored in the belief set $\mathfrak{B}_t$ are used to generalize Q-values across different states. While the agent observes transition data in a task M_t, a separate Q-learning agent is maintained for each state abstraction $\phi \in \mathfrak{B}_t$ and another for the identity state abstraction $\phi_{\text{identity}} : s \mapsto s$. (The motivation here is that the agent should consider not only the Q-values of actions that pertain to previously seen abstractions but that it should also have potential to learn Q-values in the full observable state space. We consider biological implications of this assumption in the discussion). Using a state abstraction ϕ, a state $s \in \mathcal{S}$ is mapped to a latent state $\phi(s)$ and this latent state $\phi(s)$ is given as input to the Q-learning algorithm. The Q-learning algorithm thus generalizes Q-values to multiple states that map to the same latent state. As in the Bayesian mixture of experts scheme [3, 27], the agent selects its overall policy by mixing the policies of each Q-learning agent using the posterior probabilities $\Pr(\phi|M_t, \mathfrak{B}_t, c_t)$. Specifically, the probability of selecting action a at state s is

$$\pi(s, a) = \sum_{\phi \in \mathfrak{B}_t \cup \{\phi_{\text{identity}}\}} \Pr(\phi|M_t, \mathfrak{B}_t, c_t)\pi_\phi(s, a), \tag{12}$$

where π_ϕ are the action-selection probabilities of the Q-learning algorithm corresponding to the state abstraction ϕ. For example, if the posterior probabilities place a high weight on a previously learned state abstraction $\phi \in \mathfrak{B}_t$, then the agent will effectively select actions similar to a Q-learning algorithm that is run on the latent state space constructed by the state abstraction ϕ. In this case, an optimal policy should be obtained more quickly in comparison to not using any state abstraction, assuming the state abstraction ϕ is constructed properly for the given task. If the posterior places a high weight on the identity state abstraction ϕ_{identity}, then the model will effectively select actions similarly to the usual Q-learning algorithm.

While training, the non-parametric Bayesian model also uses all observed transitions (s, a, r, s') to construct a transition and reward table. After 200 training episodes on a particular task, these transition and reward tables are used to construct either a reward-maximizing or reward-predictive state abstraction $\phi_{\text{next-best}}$. Then, the posterior probabilities $\Pr(\phi|M_t, \mathfrak{B}_t, c_t)$ are computed as described in Eq. 7 and a state abstraction is sampled using this posterior distribution. Depending on the parameter settings for α and β, the newly learned state abstraction $\phi_{\text{next-best}}$ may be added into the belief set $\mathfrak{B}_{t+1}$ for the next task or previously learned state abstraction $\phi \in \mathfrak{B}_t$ is re-used and its count $c_t(\phi)$ is increased. When training on the first task, the belief set $\mathfrak{B}_1$ is initialized to the empty set.

This model is tested on the task sequence illustrated in Fig 6. The top row depicts two different maze maps that are used to construct a curriculum of five tasks. Each map

[2] We found that 200 episodes allow each tested algorithm to converge to an optimal policy.

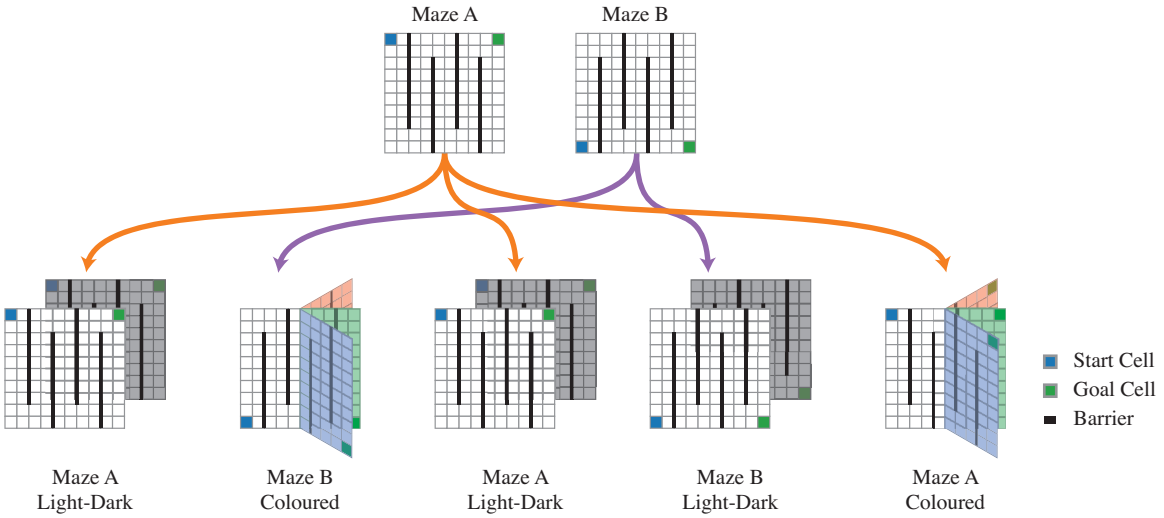


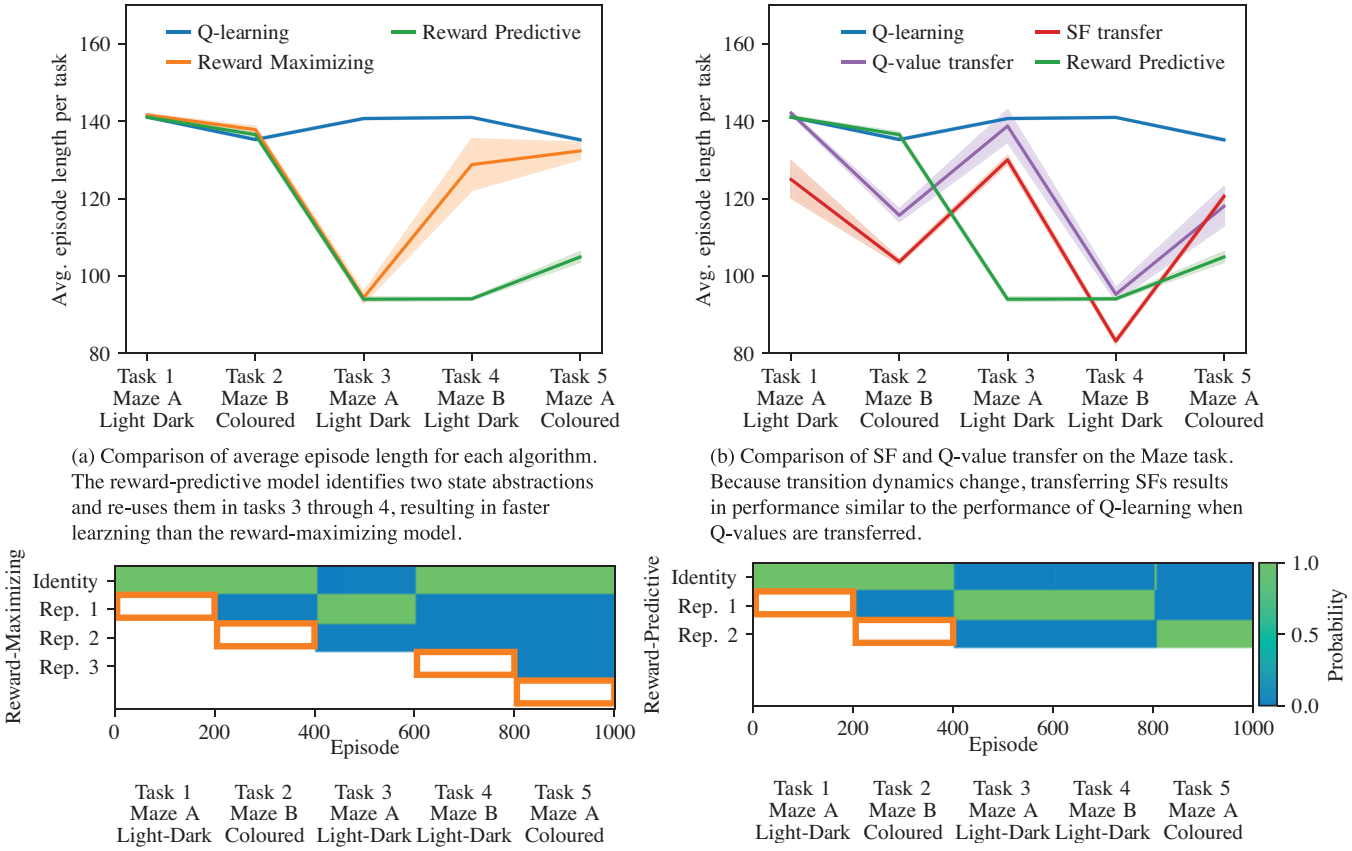
Fig 6. Maze curriculum. Maze A and Maze B are augmented with an irrelevant state variable to construct a five-task curriculum. In each maze, the agent starts at the blue grid cell and can move up, down, left, or right to collect a reward at the green goal cell. The black lines indicate barriers the agent cannot pass. Once the green goal cell is reached, the episode finishes and another episode is started. Transitions are probabilistic and succeed in the desired direction with probability 0.95; otherwise the agent remains at its current grid cell. A five-task curriculum is constructed by augmenting the state space either with a "light" or "dark" colour bit (first, third, and fourth task), or the right half of the maze is augmented with the colour red, green, or blue (second and fifth task).

is a 10×10 grid world where the agent has to navigate from the blue start state to the green goal location. Once the green goal location is entered, the agent receives a reward of $+1$ and the episode is ended. The transition dynamics are the same in each task with the difference that the agent cannot cross the black barrier. These two mazes are mirror images of another and the optimal action is different at each grid cell. Consequently, transitions, rewards, and the optimal policy of Maze A and Maze B are different at every state and cannot be immediately transferred from one maze to another.

Using these two mazes, a task sequence is constructed by adding a "light/dark" variable or a "red/green/blue" colour variable into the state that is irrelevant for navigation. This task sequence is designed to demonstrate that if an algorithm learns to correctly generalize across different states, then the algorithm can learn to solve the maze navigation task faster than an algorithm that does not generalize correctly.

The schematic in Fig 6 illustrates how the task sequence is constructed and how additional state variables are introduced. In the bottom left of Fig 6, the task "Maze A Light-Dark" is constructed by augmenting each state of the Maze A task with a binary "light/dark" variable. As an agent transitions between different grid locations, this binary variable switches with equal probability. By adding this variable, the state space is doubled to 200 states. Note that the state s will be communicated to the agent as an index that ranges from 0 to 199. The agent is not informed about the fact that states are augmented by a binary variable. To determine how this 200 state light-dark maze can be compressed, the agent would have to infer that state 0 and state 99 are equivalent and can be compressed to one latent state, for example.

The task "Maze B Coloured" (second map in bottom row of Fig 6) is constructed by augmenting the right half of the maze with a "red/green/blue" colour variable. In this case, states corresponding to the left half of the maze are not changed, but states that correspond to the right half are augmented with either the colour red, green, or blue. Intuitively, as the agent transitions into the right half of the maze, it will observe a



(a) Comparison of average episode length for each algorithm. The reward-predictive model identifies two state abstractions and re-uses them in tasks 3 through 4, resulting in faster learzning than the reward-maximizing model.

(b) Comparison of SF and Q-value transfer on the Maze task. Because transition dynamics change, transferring SFs results in performance similar to the performance of Q-learning when Q-values are transferred.

(c) Posterior probabilities during learning for the reward-predictive model (left panel) and the reward-maximizing model (right panel).

Fig 7. Transferring state representations influences learning speed on the maze curriculum. 7(a) and 7(b): Each experiment was repeated ten times and the average across all repeats was plotted. The shaded areas indicate the standard errors of measure. For each experiment, different learning rates and hyper-parameter settings were tested and the settings resulting in the lowest average episode length are plotted. Section S.3 describes the tested implementation and hyper-parameters in detail. 7(c): Plot of the posterior distribution as a function of training episode. The orange rectangle indicates tasks in which the agent used the identity abstraction to learn a new state representation that was added into the belief set after 200 episodes of learning.

coloured grid cell and colours will randomly switch between either red, green, or blue. Conforming to the light-dark maze construction, the state is presented to the agent as an index ranging from 0 to 199. The agent is not given a state in a factored form, for example a grid position and colour.

The bottom row of Fig 6 depicts the five-task curriculum. In this experiment, an agent can either learn how to maximize reward in each of the 200-state tasks or learn how to compress each task into 100 latent states, generalize information across different tasks, and ultimately learn an optimal policy faster and generate higher total reward.

Fig 7 presents the results of the learning experiment conducted on the maze-task curriculum. The average-per-task episode length of each algorithm is plotted in Fig 7(a). Because each task is a navigation problem, a low average episode length indicates that an algorithm reaches the rewarding goal using fewer time steps and can generate on average more reward per time step. For Q-learning, the average episode length per task remains roughly constant (blue curve in Fig 7(a)), because Q-learning does not transfer information across tasks. In comparison, the reward-predictive non-parametric Bayesian

model achieves a significantly lower average episode length on tasks three through four. This behaviour is explained by the posterior plotted in the top panel in Fig 7(c). On the first two tasks, the reward-predictive model adds two new state abstractions into its belief set. During training on task 1 and task 2, this model uses the identity state abstraction and does not (and cannot) generalize across two different states. Consequently, there is no difference in performance between the reward-predictive model and Q-learning. Task 1 and task 2 expose the agent for the first time to a light-dark and a coloured maze and after learning in these two tasks the reward-predictive model adds a new state abstraction into its belief set (orange boxes in Fig 7(c), left panel). From task 3 onward, the agent detects within the first few episodes which state abstraction to re-use in which task, resulting in faster learning and consequently shorter average episode lengths on these tasks. These results demonstrate that the reward-predictive model is capable of extracting two state abstractions, one for the light-dark scenario and one for the coloured scenario, and re-using these state abstractions.

In contrast, the reward-maximizing model only performs comparably to the reward-predictive model on the third task (orange curve in Fig 7(a)). The posterior probability plot for this model (right panel in Fig 7(c)) indicates that only on task 3 a previously learned state abstraction is re-used. This re-use occurs because the first and third tasks are identical and the first task's solution can be repeated on the third task. For all other tasks, the reward-maximizing model introduces a new state abstraction into its belief set. This supports the hypothesis that the reward-maximizing model effectively memorizes a solution for each task and can only repeat previously learned solutions.

Fig 7(b) compares the average episode length of the reward-predictive model with transferring and adjusting SFs, the system used in prior work [11–14, 17, 18, 28–30].[3] On task 1 and task 2, the SF-learning algorithm outperforms Q-learning and the reward-predictive model, because the SF-learning algorithm converges faster than Q-learning on the tested navigation tasks, as described in Section S.2.1. On tasks 3 and 5, the reward-predictive model outperforms the SF transfer method because the reward-predictive model identifies which state abstraction to use in which task, as previously discussed. Nevertheless, the SF transfer method outperforms the reward-predictive model on the fourth task. Note that when transitioning from task 3 to task 4, the underlying light-dark state abstraction is not changed whereas in all other task changes the underlying state abstraction is changed as well. This result suggests that SFs themselves implicitly incorporate parts of the state abstraction helping the SF-learning algorithm to converge faster in task 4.

To further investigate how the average episode length is influenced by transferring SFs or Q-values, Fig 7(b) plots the average episode length when the Q-learning algorithm is used and learning is initialized with the Q-values of the previous tasks. The green curve plots the average episode length when the SF-learning algorithm is used and learning is initialized with the SFs learned for the previous task. Fig 7(b) demonstrates that while the SF-learning algorithm generally seems to converge faster on the presented tasks, there is no significant qualitative difference in reward-generating behaviour between both systems. Both systems generate a shorter average episode length on Maze B than on Maze A. In contrast to the reward-predictive agent, the SF-learning algorithm does not obtain a shorter episode length on tasks 3 through 4, suggesting that a generalization rule cannot be learned. The magenta curve in Fig 7(b) indicates that transferring Q-values does not outperform the reward-predictive model.

[3] Section S.2.1 provides a description and re-production of how re-using previously learned SFs leads to faster convergence.

4 Discussion

In reinforcement learning, the agent's goal is to find a reward-maximizing policy. But, whereas typical RL applications pertain to a single MDP, in a lifelong learning scenario (such as that confronted by biological agents), the objective is to maximize reward across a variety of environments. For this purpose, it is critical to discover state abstractions that can be efficiently re-used and transferred across a variety of situations. While several approaches exist for discovering useful state abstractions that reduce the complexity of a high dimensional task environment (e.g., using deep neural networks) in an attempt to maximize reward, this article demonstrates that, for longer term benefits, an agent should focus on learning reward-predictive state abstractions. Our findings indicate that such abstractions permit an agent to discover state spaces that can be re-used by way of analogy to previously seen state spaces, without requiring the details of the task (transitions, reward functions, or optimal policy) to be preserved.

Our initial simulations considered situations in which a single abstraction could be transferred to a subsequent MDP. However, in a lifelong learning scenario, one must consider multiple possible abstract structures that may pertain to any novel situation. When a musician picks up a banjo, they may quickly recognize its similarity to other string instruments—even those with alternate tuning—and efficiently learn to play a scale; the same musician may re-use a different structure when attempting to master the accordion. Previous theoretical work relied on non-parametric Bayesian clustering models that assess which of several previously seen structures might apply to a novel situation and be flexibly combined in a compositional fashion [3], a strategy supported by empirical studies in humans [16]. However, such an approach still requires the agent to recognize that the specific transition function and/or the reward function is portable to new situations. Here, we applied a similar non-parametric Bayesian agent to cluster reward-predictive state abstractions, affording "zero-shot" transfer of state representations to novel environments that are only similar by way of analogy to previously seen scenarios. Because the reward-predictive model can identify which state abstractions are embedded in a task and re-use these state abstraction to accelerate learning, the presented results suggest that reward-predictive state representations generalize across tasks.

Biologically, our findings motivate studies to investigate whether brain systems involved in representing state spaces, such as the hippocampus and orbitofrontal cortex [31–33], have learning rules that are guided by minimizing reward-predictive loss, rather than simply minimizing the Bellman error as in classical temporal difference learning rules leveraged by striatal dopaminergic systems [34, 35]. Indeed, dopaminergic learning signals themselves are diverse, not only conveying reward prediction errors used for optimizing actions, but with some signals (perhaps projecting to distinct circuits) appearing to be used to learn about state transitions that permit subsequent transfer [11, 18, 36, 37]. Our simulations motivate more tailored experiments to investigate the potential role of such signals in compressing state representations such that they can be analogically reused.

The Tolman-Eichenbaum machine [38, 39] presents a model for generalization in the hippocampal-entorhinal system [40]. Similar to reward-predictive state abstractions, this model learns a latent representation that is predictive of future outcomes or stimuli but is also tied to a fixed transition function. While this model is not formulated in the usual RL framework, predicting future outcomes or stimuli can also be understood as a form of reward prediction. However, this model is trained directly on entire interaction sequences to predict future outcomes, and the learned representations are thus tied to the transition function. The presented transfer examples and simulations illustrate that reward-predictive state abstractions are not restricted by these limitations and can be directly re-used, assuming certain state equivalences are preserved.

Our approach also stands in contrast to prior attempts to leverage SFs [11–14, 17, 18, 28, 29] in which the SFs themselves are used to initialize learning in novel environments. Such an approach can accelerate learning in some situations, but it can be fragile to changes in the optimal policy [14] and transition function. A similar effect has also been shown for variations of reward-maximizing state abstractions [5], but these abstractions are also adjusted to each task, similar to SFs. While prior work mitigates this re-learning by associating a novel task with one out of multiple previously learned SFs [13, 30], these methods still rely on initializing learning with a previously learned representation to obtain a performance gain over solving a task from scratch. In contrast, reward-predictive representations have the ability to abstract away irrelevant task features and these abstractions can be re-used without re-learning them. While the presented reward-predictive model transfers state abstractions across tasks, this model has to re-learn how individual latent states are associated with one-step rewards or SFs for each task. In fact, the presented abstraction transfer models could be combined with prior work [3, 13, 16, 30] that transfers SFs, latent transition functions, or latent reward functions to integrate the benefits of each transfer system.

In related work [17], the SR of an MDP was compressed using PCA and the obtained representations were demonstrated to be suitable for transfer and connections to place cells and grid cells in the hippocampus. However, this compressed SR constructs a representation of the transition function itself, and hence transfer is again limited to environments that share the same transition function. In contrast, reward-predictive state abstractions separate the transition dynamics (and the SR) from the compression on the state space itself, and thus generate a latent state representation of a task exploiting analogical task equivalences. Latent state abstractions are not tied to particular transitions [3, 4], and can thus circumvent this dependency without adjusting the transferred representation itself.

While reward-predictive state abstractions do not limit an agent's ability to obtain an optimal policy for an MDP [9, 41], the solution space of possible reward-predictive state abstractions is far more constrained. Prior deep learning models [42] construct latent state representations as part of a model-free and model-based hybrid model that constructs a latent state representation and extracts the underlying state-transition dynamics. In contrast to their method, reward-predictive state abstractions compress the state space by generalizing across states that generate identical future expected reward sequences. While this article uses an existing SF model [9] to compute $l_{\text{predictive}}$, several other methods exist to evaluate reward-predictive state abstractions [43–45].

4.1 Limitations and Future Directions

With the exception of Fig 3(c), each simulation experiment assumes that a given task has an (unknown) state-abstraction embedding. In this case, there always exists a state abstraction which, if discovered, would allow any learning algorithm to find an optimal policy. A case that has not been studied in this article and is left for future work is the case when a task is over-compressed (i.e., lossy compression). Over-compressing a task induces approximation errors, because the compression removes too much information or detail from the state space such that accurate predictions are no longer possible [5, 9]. If only the latent state is given as state input to an algorithm like Q-learning, the algorithm may not converge and learn an optimal policy because the latent state is only providing partial information about the actual state of the task. One could analyze the problem as a partially observable MDP (POMDP) [46], but algorithms that can solve POMDPs also maintain a belief about which actual state they are in. In this literature, the actual state is assumed to be unknown to the agent. Because this work assumes that the actual state is known to the agent, the benefit of using such an algorithm is not clear in the case where a task is over-compressed by a state abstraction. Under what

assumptions algorithms like Q-learning can be combined with state abstractions that over-compress a task is left for future work.

The presented results consider finite MDPs, allowing the algorithm to tabulate a value or latent state for each possible state. Another direction of future work is to extend the presented models and algorithms to larger state spaces, such as images. Such an extension would integrate neural networks or deep learning techniques, and allow the presented models to be applied to more complex tasks, such as computer games [22] or visual transfer tasks that can also be used in a human subject study [4].

5 Conclusion

The presented results suggest that reward-predictive state abstractions generalize across tasks with different transition and reward functions, motivating the design of future transfer algorithms. The discussed connections to predictive representations in the brain and generalization in human and animal learning motivate further experiments to investigate if biological systems learn reward-predictive representations.

Acknowledgments

We would like to thank Rex Liu, Nicholas T. Franklin, and Alana Jaskir for insightful comments on previous drafts of this article. Lucas Lehnert was supported by the NIH T32MH115895 Training program for Interactionist Cognitive Neuroscience (ICoN). Michael J. Frank was supported by grant NIH MH084840.

Supporting information

S.1 State Abstractions

For finite state and action MDPs, the transition function can be presented as a set of left-stochastic transition matrices $\{\boldsymbol{M}_a\}_{a\in\mathcal{A}}$ where

$$\forall a \in \mathcal{A},\ \forall i,j \in \mathcal{S},\ \boldsymbol{M}_a(i,j) = p(i,a,j), \tag{13}$$

and a set of expected reward vectors $\{\boldsymbol{r}_a\}_{a\in\mathcal{A}}$ where

$$\forall a \in \mathcal{A},\ \forall i \in \mathcal{S},\ \boldsymbol{r}_a(i) = \mathbb{E}_{s'}[r(s,a,s')]. \tag{14}$$

A state abstraction is an MDP homomorphism [23] mapping an MDP $M = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma\rangle$ into an abstract MDP $M_\phi = \langle \mathcal{S}_\phi, \mathcal{A}, m, w, \gamma\rangle$. Using a weighting function ω, the abstract MDP "aggregates" the transition and reward function across states $s \in \mathcal{S}$ that map to the same abstract state $s_\phi \in \mathcal{S}_\phi$. Specifically, the abstract reward function w is constructed as

$$w(s_\phi, a) = \sum_{s:\phi(s)=s_\phi} \omega(s)\boldsymbol{r}_a(s). \tag{15}$$

The transition function between latent states m is constructed as

$$m(s_\phi, a, s'_\phi) = \sum_{s:\phi(s)=s_\phi} \sum_{s':\phi(s')=s'_\phi} \omega(s)p(s,a,s'). \tag{16}$$

The weighting function is assumed to be non-negative and the sum across a state partition evaluates to one: $\sum_{s:\phi(s)=s_\phi} \omega(s) = 1$. The simulations presented in

Sections 3.1 and 3.2 assume a uniform weighting function which averages across state partitions. Similar to Eq (13) and Eq (14), the abstract transition matrices $\{\boldsymbol{M}_a\}_{a\in\mathcal{A}}$ and abstract reward vectors $\{\boldsymbol{w}_a\}_{a\in\mathcal{A}}$ can be computed and

$$\forall a \in \mathcal{A},\ \forall i,j \in \mathcal{S}_\phi,\ \boldsymbol{M}_a(i,j) = m(i,a,j) \text{ and } \boldsymbol{w}_a(i) = w(i,a). \tag{17}$$

Note that function m defines transition probabilities between latent states in the same way the transition function p defines transition probabilities between states [6]. Consequently, the latent model described by the matrices and vectors $\{\boldsymbol{M}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$ can be used as a normal MDP, with the only difference that this latent MDP is defined on latent states. A policy that is optimal with respect to this compressed or latent MDP $\{\boldsymbol{M}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$ can be computed by performing value iteration [1, Chapter 4.4] on the matrices and vectors $\{\boldsymbol{M}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$. Such a policy is then used as the optimal policy for an agent that generalizes across states according to the state abstraction ϕ.

S.2 Successor Features Identify Reward-Predictive State Abstractions

An alternative model of SFs, called Linear Successor Feature Models (LSFMs), can be used to identify reward-predictive state abstractions [9]. For an MDP, the SR [10] is

$$\boldsymbol{\Psi}^\pi = \sum_{t=1}^{\infty} \gamma^{t-1} \boldsymbol{P}^\pi, \tag{18}$$

where $\boldsymbol{P}^\pi$ is the transition matrix under a policy π. The action-conditional SR can be defined as

$$\boldsymbol{\Psi}^{a,\pi} = \boldsymbol{I} + \gamma \boldsymbol{P}^a \sum_{t=1}^{\infty} \gamma^{t-1} \boldsymbol{P}^\pi = \boldsymbol{I} + \gamma \boldsymbol{P}^a \boldsymbol{\Psi}^\pi. \tag{19}$$

Using the transition matrices $\{\boldsymbol{M}_a\}_{a\in\mathcal{A}}$ for the abstract MDP, the action conditional SR can be defined in the abstract state space as

$$\boldsymbol{F}_a = \boldsymbol{I} + \gamma \boldsymbol{M}_a \sum_{t=1}^{\infty} \gamma^{t-1} \boldsymbol{M}^\pi, \tag{20}$$

where $\boldsymbol{M}^\pi$ is the transition matrices between latent states according to policy π. To measure how well a LSFM $\{\boldsymbol{F}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$ predicts future reward outcomes, the a state abstraction ϕ is first assumed to map each state to a specific one-hot column vector. A one-hot column vector $\boldsymbol{e}_i$ is set to one in its ith entry and all other entries are set to zero. For any state s, the output of the state abstraction function $\phi(s) = \boldsymbol{e}_i$ for some index i and the state s is mapped to the latent state i in this case. If the state abstraction's output vector $\boldsymbol{e}_i$ is n dimensional, then the state abstraction compresses the state space into n latent states. The one-step-reward-prediction error is then

$$\varepsilon_r = \max_{s,a} \left| r(s,a) - (\phi(s))^\top \boldsymbol{w}_a \right| \tag{21}$$

and the SF error

$$\varepsilon_\psi = \max_{s,a} \left| \left| (\phi(s))^\top + \gamma \mathbb{E}\left[(\phi(s'))^\top \overline{\boldsymbol{F}} \middle| s,a \right] - (\phi(s))^\top \boldsymbol{F}_a \right| \right| \tag{22}$$

are computed [9]. In Eq (22), the expectation $\mathbb{E}\left[(\phi(s'))^\top \overline{\boldsymbol{F}} \middle| s,a \right]$ is computed over all states s' that can be reached from state s by selecting action a and $\overline{\boldsymbol{F}} = \frac{1}{|\mathcal{A}|} \sum_{a\in\mathcal{A}} \boldsymbol{F}_a$. These two errors can then be used to bound the reward-sequence prediction error and

$$\left| \phi^\top(s) \boldsymbol{M}_{a_1} \cdots \boldsymbol{M}_{a_{t-1}} \boldsymbol{w}_{a_t} - \mathbb{E}[r_t | s, a_1, ..., a_t] \right| \leq \varepsilon_r + C_{\gamma,t} \varepsilon_\psi \tag{23}$$

where $C_{\gamma,t}$ is a constant depending on the discount factor γ and sequence length T. In this article all abstract states $\phi(s)$ are encoded as one-hot bit vectors, a vector of zeros with only one entry set to one. The product $\phi^\top(s)\boldsymbol{M}_{a_1}\cdots\boldsymbol{M}_{a_{t-1}}$ evaluates to a row probability vector specifying with which probability an abstract state is reached starting at the abstract state $\phi(s)$ and following the action sequence $a_1, ..., a_{t-1}$. By computing the dot-product between this vector and the reward vector $\boldsymbol{w}_{a_t}$, the prediction of the expected reward is calculated after following the actions sequence $a_1, ..., a_t$ starting at state s with $\phi^\top(s)\boldsymbol{M}_{a_1}\cdots\boldsymbol{M}_{a_{t-1}}\boldsymbol{w}_{a_t}$. The loss function $l_{\text{predictive}}$ is then computed with

$$l_{\text{predictive}}(\phi) = \varepsilon_r + \varepsilon_\psi. \tag{24}$$

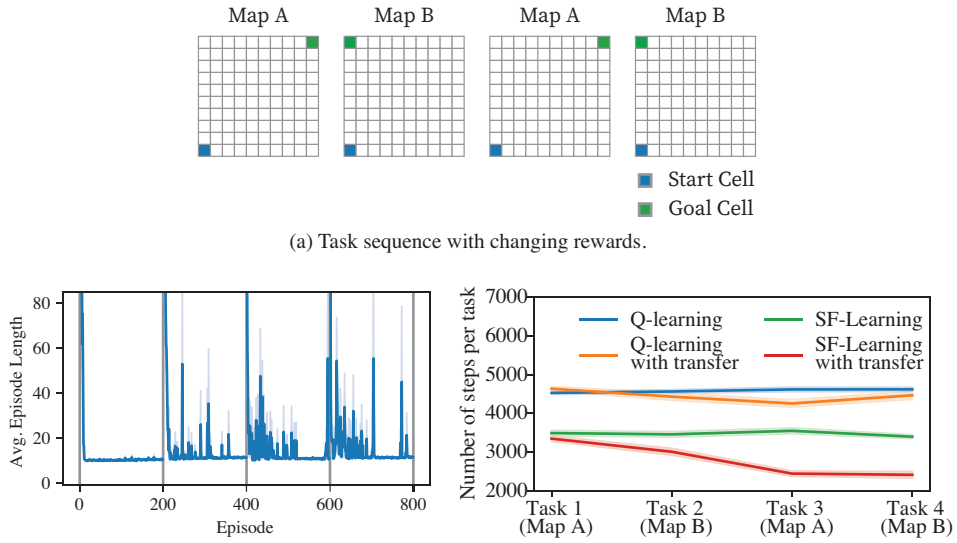
If $l_{\text{predictive}}(\phi) = 0$, the ϕ is a reward-predictive state abstraction. A more detailed description of these bounds and LSFMs was previously presented by [9].

S.2.1 Transferring SFs Across Tasks

To establish a baseline and investigate to what extend re-using previously learned Q-values or SFs accelerates learning, Fig 8 re-produces the navigation experiment presented in prior work [14] and demonstrates the benefit of transferring SFs between different tasks as in prior work [11, 12, 18, 47]. In this experiment, the Q-learning [19] and SF-learning [12] algorithms are tested on a sequence of four grid-world navigation tasks (Fig 8(a)). SF-learning is an algorithm that learns a SF vector $\boldsymbol{\psi}^\pi(s, a)$ for each state s and action a. Similar to Q-learning, SF-learning iteratively improves its current behaviour policy until an optimal policy is found [12]. In Appendix S.3 we present all implementation details on how these algorithms are implemented and evaluated. Each agent was allowed to attempt each navigation task for 200 episodes. After 200 episodes, each agent is signaled that the grid-world map has changed. While interacting with a task, each algorithm updates its internal values and attempts to find an optimal policy that reaches the goal location as fast as possible.

Fig 8(b) plots the episode length of the SF-learning algorithm as a function of the number of episodes. The gray lines indicate when the interaction in one of the four tasks begins. This plot shows that at the end of learning in each task, a close to optimal policy is found because the agent can repeatedly navigate from start to goal location within about ten time steps. However, at the beginning of learning in each map (episode 0, 200, 400, and 600), the episode length spikes up. The first spike at episode zero occurs because the SF-learning algorithm is initialized and has to learn an optimal policy from scratch. At episodes 200, 400, and 600, the SF-learning algorithm is reset but is allowed to preserve its previously learned SFs. Because the change in goal location also leads to a change in optimal policy, the episode length spikes up when the goal location is changed, but the SF-learning algorithm can subsequently recover an optimal policy for that new task [13].

Fig 8(c) plots the average episode length for each task and tested algorithm. In this plot, each point is averaged across the 200 episodes an agent spent in each task. Because the task is to navigate across a grid world, a shorter episode or trial length is indicative of a policy that generates more reward per time step. For both Q-learning and SF-learning the average episode length does not vary significantly across the different tasks. This behaviour is expected, because both algorithms are reset when the grid map is changed and an optimal policy is learned from scratch. Furthermore, these simulations indicate that SF-learning converges to an optimal policy faster than Q-learning in this particular experiment, because the average episode length is significantly lower for SF-learning than for Q-learning. The orange curve in Fig 8(c) plots the episode length when Q-learning re-uses previously learned Q-values. While re-using previously learned Q-values reduces the average episode length on tasks two



(a) Task sequence with changing rewards.

(b) Episode length when SFs are transferred.

(c) Avg. episode length for each task

Fig 8. Transfer experiment adopted from [14]. 8(a): In this experiment each algorithm was simulated on a sequence of grid-world maps. For each grid map, the agent starts at the blue grid cell and navigates to the green goal cell to collect a reward. The transitions are the same as described in Fig 6, but these tasks do not contain any barriers. 8(b): Plot of the average episode length as a function of the episode for the SF-learning agent. The gray lines indicate the start or end of learning in one of the four tasks. After a certain number of episodes, the SF-learning algorithm can find an optimal policy that navigates across the map in about ten time steps. 8(c): Plot of the average episode length for each task and each tested algorithm. Each simulation was repeated 20 times and averages across repeats are plotted. Standard errors indicated by the shaded areas.

through four, re-using Q-values does not result in a significant performance improvement and this algorithm is outperformed by the SF-learning algorithm. The red curve in Fig 8(c) plots the episode length when the SF-learning algorithm can re-use previously learned SFs. In this case, the average episode length is significantly lower on tasks two through four when the SF-learning algorithm is allowed to transfer previously learned SFs. While SFs have to be re-learned [12, 13], re-using previously learned SFs leads to faster convergence. This experiment re-produces previous results demonstrating that re-using SFs can speed up learning [11–14, 17, 18, 28–30]. This method of transferring SFs is used as a baseline in Section 3.3.

S.3 Hyper-Parameter Selection and Implementation of Learning Experiments

In this section we present implementation details necessary to reproduce the learning experiments presented in Figures 6, 7, and 8. Because the Q-learning algorithm and SF-learning algorithm use a learning rate parameter, the resulting episode lengths are sensitive to hyper-parameters and learning rates. To account for this dependency, each algorithm was tested on a range of different hyper-parameter combinations and the hyper-parameter setting for each algorithm that leads to the best performance was used for the presented experiments.

S.3.1 Optimistic initialization and exploration

To ensure that each algorithm learns an optimal policy using only interaction data, each algorithm was initialized optimistically to ensure efficient exploration [48]. In optimistic initialization, all Q-values [1, 19] are initialized to a specified maximum value $Q_{\max}$, which upper-bounds the actual Q-values of the task. A Q-learning agent then learns to incrementally decrease its value predictions until prediction errors are low. By combining this initialization scheme with an exploration policy that is greedy with respect to the current Q-value predictions, the Q-learning agent will repeat actions where Q-value predictions overestimate the actual return. All tested tasks are navigation tasks where an agent always receives a reward of one once the goal location is reached and a reward of zero otherwise. Because the reward of one is incurred only once, all Q-values lie in the interval $[0, 1]$ and $Q_{\max} = 1$.

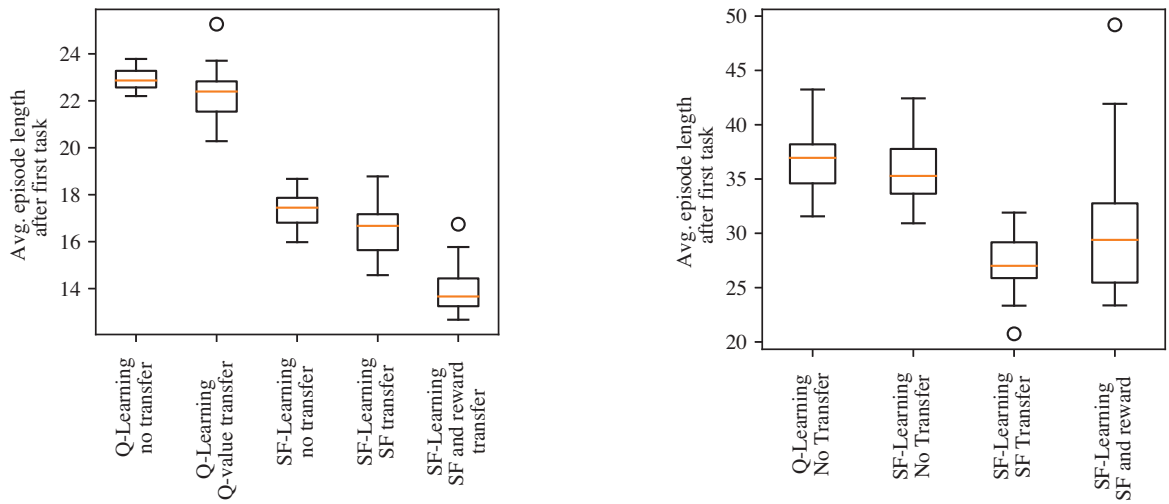
Fig 9 demonstrates that optimistic initialization combined with a greedy action selection strategy significantly outperforms initializing Q-values or SFs to zero and using an ε-greedy exploration policy as prior work did [12, 14, 47]. This result confirms that both Q-learning and SF-learning benefit from the optimistic initialization strategy used in the presented simulation experiments.

S.3.2 Q-learning

The Q-learning algorithm [19] was tested for three different learning rate settings: 0.1, 0.5, and 0.9. In all experiments, using a learning rate of 0.9 leads to the fastest learning and overall best performance, even when the algorithm was combined with a state abstraction. When Q-learning was used in combination with a state abstraction, each transition (s, a, r, s') was mapped to a transition $(\phi(s), a, r', \phi(s'))$ between latent states.

S.3.3 SF-learning

Algorithm 1 outlines an adaptation of the SF-learning algorithm [47] that incorporates optimistic initialization. Our implementation of SF-learning maintains a matrix $\boldsymbol{G}$ of



(a) Optimistic Initialization

(b) Zero value initialization and ε-greedy exploration.

Fig 9. Episode length averaged across tasks two through four from Fig 8(a). 9(a): Episode length of Q-learning and SF-learning under different transfer strategies when optimistic initialization is used. The configuration "Q-Learning Q-val. Transfer" re-used previously learned Q-values. The configuration "SF-Learning SF transfer" only re-uses previously learned SFs while the configuration "SF-Learning SF and reward transfer" re-uses both SFs and the learned one-step reward predictions. 9(b): In this experiment both algorithms are initialized to produce zero Q-values and an ε-greedy exploration policy is used. This exploration strategy selects actions uniformly at random with ε probability and with $1-\varepsilon$ probability actions are selected greedily with respect to the current Q-value predictions. At the beginning of training $\varepsilon = 1$ (uniform random action selection) and by episode 80 ε was decreased to zero (greedy action selection) using linear interpolation.

size $SA \times SA$, where S is the number of states and A the number of actions. Each row of this matrix $\boldsymbol{G}$ approximates the SF

$$\boldsymbol{G}(s,a) \approx \mathbb{E}\left[\boldsymbol{e}_{s_1,a_1} + \gamma \boldsymbol{e}_{s_2,a_2} + \gamma^2 \boldsymbol{e}_{s_3,a_3} + \cdots \middle| s_1 = s, a_1 = a, \pi\right], \tag{25}$$

where the expectation in Eq (25) is defined over all infinite length trajectories that start with state s and action a and then follow the policy π. Each vector $\boldsymbol{e}_{s_i,a_i}$ is a one-hot bit vector associated with the state-action pair (s_i, a_i). This vector has a dimension of SA and the entry corresponding to the state action pair (s_i, a_i) is set to one with all other entries being zero. The definition presented in Eq (25) is equivalent to the definition presented by [47], with the distinction that one-hot bit vector features are used. Consider a vector $\boldsymbol{v}$ of length SA storing one-step reward predictions in each entry:

$$\boldsymbol{v}(s,a) = r(s,a) \tag{26}$$

Algorithm 1 incrementally learns a matrix $\boldsymbol{G}$ and vector $\boldsymbol{v}$ such that Eq (25) and Eq (26) are satisfied. The iterates in lines 8 and 9 of Algorithm 1 are obtained by substituting $\boldsymbol{\phi}_t = \boldsymbol{e}_{s_t,a_t}$ and $\psi^\pi(s,a) = \boldsymbol{e}_{s,a}^\top \boldsymbol{G} = \boldsymbol{G}(s,a)$ into [14, Algorithm 1].

If the matrix $\boldsymbol{G}$ is constructed such that Eq (25) exactly holds, then [47]

$$Q^\pi(s,a) = (\boldsymbol{G}(s,a))^\top \boldsymbol{v}. \tag{27}$$

The SF-learning algorithm learns an optimal policy by constructing Q-value prediction using Eq (27). Optimistic initialization can be integrated into SF-learning by initializing the matrix $\boldsymbol{G}$ and vector $\boldsymbol{v}$ such that $Q(s,a) = Q_{\max}$ at the beginning of training. This can be accomplished by initializing $\boldsymbol{G} = \boldsymbol{I}$ and every entry $\boldsymbol{v}_0(s,a) = Q_{\max}$ (line 2 in

Table 1. Learning rates used for SF-learning.

	Tested α_{SF}	Tested α_r	Best α_{SF}	Best α_r
SF-learning with SF transfer in Fig. 7	0.1, 0.5, 0.9	0.1, 0.5, 0.9	0.5	0.5
SF-learning in Figs. 8 and 9(a)	0.1, 0.3, 0.5, 0.7, 0.9	0.1, 0.3, 0.5, 0.7, 0.9	0.3	0.9
SF-learning with SF transfer in Figs. 8 and 9(a)	0.1, 0.3, 0.5, 0.7, 0.9	0.1, 0.3, 0.5, 0.7, 0.9	0.3	0.9
SF-learning with SF and reward transfer in Fig. 9(a)	0.1, 0.3, 0.5, 0.7, 0.9	0.1, 0.3, 0.5, 0.7, 0.9	0.3	0.9
SF-learning in Fig. 9(b)	0.1, 0.3, 0.5, 0.7, 0.9	0.1, 0.3, 0.5, 0.7, 0.9	0.1	0.1
SF-learning with SF transfer in Fig. 9(b)	0.1, 0.3, 0.5, 0.7, 0.9	0.1, 0.3, 0.5, 0.7, 0.9	0.1	0.3
SF-learning with SF and reward transfer in Fig. 9(b)	0.1, 0.3, 0.5, 0.7, 0.9	0.1, 0.3, 0.5, 0.7, 0.9	0.1	0.9

Algorithm 1). Hence, we have for the Q-values at initialization

$$Q(s,a) = (\boldsymbol{G}(s,a))^\top \boldsymbol{v} = \boldsymbol{e}_{s,a}^\top \mathbf{1} Q_{\max} = Q_{\max}, \tag{28}$$

where $\mathbf{1}$ is a vector with each entry being set to one. By combining this initialization scheme and selecting actions greedily with the predicted Q-values, we found that the SF-learning algorithm converges faster to the Q-learning algorithm. Table 1 lists the tested learning rates and the best performing learning rate setting for each simulation.

Algorithm 1 SF-learning with optimistic initialization, adopted from [47]

Input: Learning rates $\alpha_{\text{SF}}, \alpha_r \in (0,1]$, the maximum Q-value $Q_{\max}$.
Initialize $\boldsymbol{G} \leftarrow \boldsymbol{I}$ and $\boldsymbol{v} \leftarrow \mathbf{1} * Q_{\max}$ where $\mathbf{1}$ is a vector of ones.
loop
  Receive current state s from MDP.
  Select action $a = \arg\max_{a'} Q(s,a')$ with Eq (27).
  Observe transition (s,a,r,s').
  $a^* \leftarrow \arg\max_{a'} Q(s',a')$ with Eq (27).
  $\boldsymbol{G} \leftarrow \boldsymbol{G} + \alpha_{\text{SF}}(\boldsymbol{e}_{sa} + \gamma \boldsymbol{G}(s',a^*) - \boldsymbol{G}(s,a))\boldsymbol{e}_{s,a}^\top$
  $\boldsymbol{v} \leftarrow \boldsymbol{v} + \alpha_r(\boldsymbol{v}(s,a) - r)\boldsymbol{e}_{s,a}$
end loop

S.3.4 Learning of Reward-Maximizing and Reward-Predictive State Abstractions

The experiments presented in Section 3.3 learn reward-maximizing state abstractions by clustering Q-values. These Q-values are computed by performing value iteration on the constructed transition and reward tables. By clustering Q-values a state abstraction ϕ is obtained with a low $l_{\text{maximizing}}(\phi)$ value[4] [49].

A reward-predictive state abstraction is computed by computing a LSFM with low prediction errors. Such an LSFM is obtained by using a rescaled version of the loss objective $\mathcal{L}_{\text{LSFM-mat}}$ presented by [9]:

$$\mathcal{L}_{\text{LSFM-mat-scaled}} = \sum_{a \in \mathcal{A}} \alpha_w ||\boldsymbol{\Phi}\boldsymbol{w}_a - \widehat{\boldsymbol{r}}_a||_2^2 + \alpha_F ||\boldsymbol{\Phi} + \gamma \widehat{\boldsymbol{P}}_a \boldsymbol{\Phi}\overline{\boldsymbol{F}} - \boldsymbol{\Phi}\boldsymbol{F}_a||_2^2, \tag{29}$$

where $\{\widehat{\boldsymbol{P}}_a, \widehat{\boldsymbol{r}}_a\}_{a \in \mathcal{A}}$ are the transition and reward tables the non-parametric Bayesian model computes from the observed interaction data. Each row of the matrix $\boldsymbol{\Phi}$ stores a latent feature vector that is associated with a particular state. The loss objective

[4] If each state partition has Q-values that differ by at most ε from another, then $l_{\text{maximizing}}(\phi) \leq 2\varepsilon/(1-\gamma)^2$.

$\mathcal{L}_{\text{LSFM-mat-scaled}}$ is optimized using the Adam gradient optimizer [50] with a learning rate of 0.01 and $\alpha_F = 0.01$. Otherwise Tensorflow's [51] default parameters were used and the optimization process was initialized as described by [9]. After 1000 gradient update steps, the row-space of the real-valued matrix $\mathbf{\Phi}$ was clustered using agglomerative clustering to construct a partitioning of the state space and a state abstraction.

The resulting state representation will then have a low one-step-reward-prediction error ε_r and SF error ε_ψ and by Eq (24), a low $l_{\text{predictive}}(\phi)$ value [9]. During learning in a particular task, these loss values are computed on an incomplete transition table to compute the posterior over state abstraction needed for the mixture policy described in Eq (12).

S.3.5 Hyper-Parameters Influence Belief Space Size

Fig 10 plots the average episode length and belief space size for different α and β settings for each model for the learning experiment presented in Section 3.3. The right panels indicate that a decrease in α corresponds in most cases to a smaller belief space size. As the parameter α is decreased, the CRP prior will emphasize re-use of previously learned state representations, leading to a smaller belief space size after observing all five tasks. Nevertheless, the β parameter also influence the belief space size. For high β parameters, the scores $l_{\text{predictive}}$ or $l_{\text{maximizing}}$ influence the posterior more strongly and if the loss values are low enough another state abstraction is added into the belief set despite the CRP prior. Consequently, a higher β parameter corresponds to a larger belief set size as plotted in the right panels of Fig 10. The left panels in Fig 10 show that a small belief space (where $\beta = 1$) results in a high average episode length and thus poor performance on the tested task sequence. If the α parameter is increased for $\beta = 1$, then the average episode length decreases because the model tends to use multiple state abstractions in its belief set instead of enforcing the re-use of the same state abstraction. If the same state abstraction is re-used on all tasks, then the used state abstraction may remove information from the state that is needed to find an optimal policy, resulting in poor performance. For both models, the average episode length is lowest when the learned belief space has a small but not too small size. This suggests that learning and re-using a small set of state abstractions results in a shorter average episode length and better performance. If the belief space is too large, then the average episode length also increases because state abstractions are not re-used to accelerate learning in subsequent tasks and the model does not transfer previously learned structures. Note that for a setting of $\beta = 100$ only the reward-predictive model constructs a belief set of size two for $\alpha \leq 0.01$ for all ten repeats. Because the task sequence was constructed by adding irrelevant state variables in one of two ways, this result suggests that the reward-predictive model detects these two generalization patterns.

References

1. Sutton RS, Barto AG. Reinforcement learning: An introduction. MIT press; 2018.

2. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, et al. Human-level control through deep reinforcement learning. Nature. 2015;518(7540):529–533.

3. Franklin NT, Frank MJ. Compositional clustering in task structure learning. PLoS computational biology. 2018;14(4):e1006116.

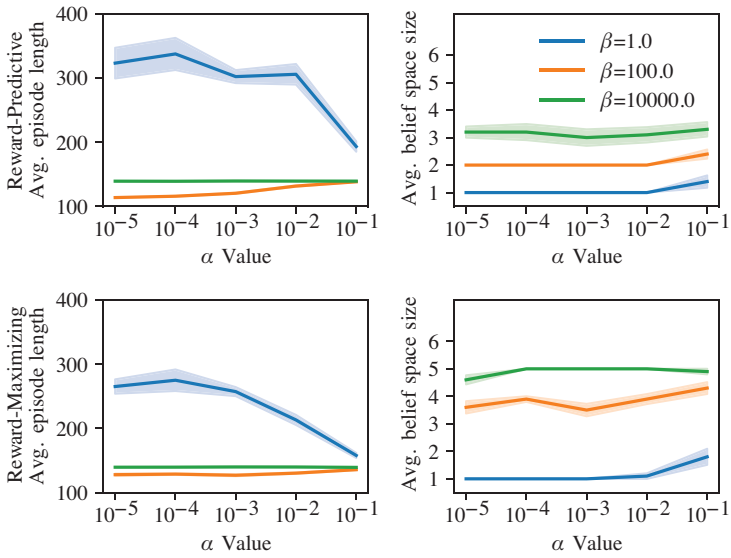


Fig 10. Model parameters α and β control the belief space size of the non-parameteric Bayesian model presented in Section 3.3.

4. Collins AGE, Frank MJ. Neural signature of hierarchically structured expectations predicts clustering and transfer of rule sets in reinforcement learning. Cognition. 2016;152:160–169.

5. Abel D, Arumugam D, Lehnert L, Littman M. State Abstractions for Lifelong Reinforcement Learning. In: Dy J, Krause A, editors. Proceedings of the 35th International Conference on Machine Learning. vol. 80 of Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR; 2018. p. 10–19. Available from: `http://proceedings.mlr.press/v80/abel18a.html`.

6. Li L, Walsh TJ, Littman ML. Towards a Unified Theory of State Abstraction for MDPs. In: ISAIM; 2006.

7. Puterman ML. Markov decision processes: discrete stochastic dynamic programming. John Wiley & Sons; 1994.

8. Taylor ME, Stone P. Transfer learning for reinforcement learning domains: A survey. Journal of Machine Learning Research. 2009;10(Jul):1633–1685.

9. Lehnert L, Littman ML. Successor Features Combine Elements of Model-Free and Model-based Reinforcement Learning. arXiv preprint arXiv:190111437v2. 2019;.

10. Dayan P. Improving generalization for temporal difference learning: The successor representation. Neural Computation. 1993;5(4):613–624.

11. Momennejad I, Russek EM, Cheong JH, Botvinick MM, Daw N, Gershman SJ. The successor representation in human reinforcement learning. Nature Human Behaviour. 2017;1(9):680.

12. Barreto A, Dabney W, Munos R, Hunt JJ, Schaul T, van Hasselt HP, et al. Successor features for transfer in reinforcement learning. In: Advances in neural information processing systems; 2017. p. 4055–4065.

13. Barreto A, Borsa D, Quan J, Schaul T, Silver D, Hessel M, et al. Transfer in Deep Reinforcement Learning Using Successor Features and Generalised Policy Improvement. In: Proceedings of the 35th International Conference on Machine Learning. vol. 80 of Proceedings of Machine Learning Research. PMLR; 2018. p. 501–510.

14. Lehnert L, Tellex S, Littman ML. Advantages and Limitations of using Successor Features for Transfer in Reinforcement Learning. arXiv preprint arXiv:170800102. 2017;.

15. Badre D, Frank MJ. Mechanisms of hierarchical reinforcement learning in cortico–striatal circuits 2: Evidence from fMRI. Cerebral cortex. 2011;22(3):527–536.

16. Franklin NT, Frank MJ. Generalizing to generalize: when (and when not) to be compositional in task structure learning. bioRxiv. 2019;doi:10.1101/547406.

17. Stachenfeld KL, Botvinick MM, Gershman SJ. The hippocampus as a predictive map. Nature Neuroscience. 2017;20:1643 EP –.

18. Russek EM, Momennejad I, Botvinick MM, Gershman SJ, Daw ND. Predictive representations can link model-based reinforcement learning to model-free mechanisms. PLoS computational biology. 2017;13(9):e1005768.

19. Watkins CJCH, Dayan P. Q-learning. Machine Learning. 1992;8(3):279–292.

20. Sutton RS. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: Proceedings of the seventh international conference on machine learning; 1990. p. 216–224.

21. Brafman RI, Tennenholtz M. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. Journal of Machine Learning Research. 2002;3(Oct):213–231.

22. Bellemare MG, Naddaf Y, Veness J, Bowling M. The Arcade Learning Environment: An Evaluation Platform for General Agents. CoRR. 2012;abs/1207.4708.

23. Ravindran B, Barto AG. Approximate Homomorphisms: A framework for non-exact minimization in Markov Decision Processes. 2004;.

24. Knuth DE. The Art of Computer Programming, Volume 4, Fascicle 3: Generating All Combinations and Partitions. Addison-Wesley; 2005.

25. Gershman SJ, Blei DM. A tutorial on Bayesian nonparametric models. Journal of Mathematical Psychology. 2012;56(1):1–12.

26. Teh YW, Jordan MI, Beal MJ, Blei DM. Hierarchical Dirichlet Processes. Journal of the American Statistical Association. 2006;101(476):1566–1581. doi:10.1198/016214506000000302.

27. Frank MJ, Badre D. Mechanisms of hierarchical reinforcement learning in corticostriatal circuits 1: computational analysis. Cerebral cortex. 2012;22(3):509–526.

28. Zhang J, Springenberg JT, Boedecker J, Burgard W. Deep reinforcement learning with successor features for navigation across similar environments. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE; 2017. p. 2371–2378.

29. Kulkarni TD, Saeedi A, Gautam S, Gershman SJ. Deep successor reinforcement learning. arXiv preprint arXiv:160602396. 2016;.

30. Madarasz TJ, Behrens T. Better Transfer Learning Through Inferred Successor Maps. 2019;.

31. Redish AD, Jensen S, Johnson A, Kurth-Nelson Z. Reconciling reinforcement learning models with behavioral extinction and renewal: implications for addiction, relapse, and problem gambling. Psychological review. 2007;114(3):784.

32. Wilson RC, Takahashi YK, Schoenbaum G, Niv Y. Orbitofrontal cortex as a cognitive map of task space. Neuron. 2014;81(2):267–279.

33. Schuck NW, Cai MB, Wilson RC, Niv Y. Human orbitofrontal cortex represents a cognitive map of state space. Neuron. 2016;91(6):1402–1412.

34. Montague PR, Dayan P, Sejnowski TJ. A framework for mesencephalic dopamine systems based on predictive Hebbian learning. Journal of neuroscience. 1996;16(5):1936–1947.

35. Collins AG, Frank MJ. Opponent actor learning (OpAL): Modeling interactive effects of striatal dopamine on reinforcement learning and choice incentive. Psychological review. 2014;121(3):337.

36. Sharpe MJ, Chang CY, Liu MA, Batchelor HM, Mueller LE, Jones JL, et al. Dopamine transients are sufficient and necessary for acquisition of model-based associations. Nature Neuroscience. 2017;20(5):735.

37. Nasser HM, Calu DJ, Schoenbaum G, Sharpe MJ. The dopamine prediction error: contributions to associative models of reward learning. Frontiers in psychology. 2017;8:244.

38. Whittington J, Muller T, Mark S, Barry C, Behrens T. Generalisation of structural knowledge in the hippocampal-entorhinal system. In: Bengio S, Wallach H, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R, editors. Advances in Neural Information Processing Systems 31. Curran Associates, Inc.; 2018. p. 8484–8495.

39. Whittington JC, Muller TH, Mark S, Chen G, Barry C, Burgess N, et al. The Tolman-Eichenbaum Machine: Unifying space and relational memory through generalisation in the hippocampal formation. bioRxiv. 2019; p. 770495.

40. Behrens TE, Muller TH, Whittington JC, Mark S, Baram AB, Stachenfeld KL, et al. What is a cognitive map? Organizing knowledge for flexible behavior. Neuron. 2018;100(2):490–509.

41. Givan R, Dean T, Greig M. Equivalence notions and model minimization in Markov decision processes. Artificial Intelligence. 2003;147(1):163–223.

42. François-Lavet V, Bengio Y, Precup D, Pineau J. Combined Reinforcement Learning via Abstract Representations. arXiv preprint arXiv:180904506. 2018;.

43. Comanici G, Precup D, Panangaden P. Basis refinement strategies for linear value function approximation in MDPs. In: Advances in Neural Information Processing Systems; 2015. p. 2899–2907.

44. Ferns N, Precup D. Bisimulation Metrics are Optimal Value Functions. In: UAI. Citeseer; 2014. p. 210–219.

45. Ferns N, Panangaden P, Precup D. Bisimulation metrics for continuous Markov decision processes. SIAM Journal on Computing. 2011;40(6):1662–1714.

46. Littman ML, Cassandra AR, Kaelbling LP. Learning policies for partially observable environments: Scaling up. In: Machine Learning Proceedings 1995. Elsevier; 1995. p. 362–370.

47. Barreto A, Munos R, Schaul T, Silver D. Successor Features for Transfer in Reinforcement Learning. CoRR. 2016;abs/1606.05312.

48. Kaelbling LP, Littman ML, Moore AW. Reinforcement learning: A survey. Journal of artificial intelligence research. 1996;4:237–285.

49. Abel D, Hershkowitz D, Littman M. Near Optimal Behavior via Approximate State Abstraction. In: Proceedings of The 33rd International Conference on Machine Learning; 2016. p. 2915–2923.

50. Kingma DP, Ba J. Adam: A Method for Stochastic Optimization. CoRR. 2014;abs/1412.6980.

51. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, et al.. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems; 2015. Available from: `https://www.tensorflow.org/`.