Reward-predictive representations generalize across tasks in reinforcement learning

Lucas Lehnert\textsuperscript{1,3,*}, Michael L. Littman\textsuperscript{1}, Michael J. Frank\textsuperscript{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

Supporting information

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 Fig 7, Fig 8, and supporting S1 Fig. 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.

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 \cite{1}. In optimistic initialization, all Q-values are initialized to a specified maximum value $Q_{\text{max}}$, 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 $Q_{\text{max}} = 1$.

Supporting S2 Fig demonstrates that optimistic initialization combined with a greedy action selection strategy significantly outperforms initializing Q-values or SFs to zero and using an $\varepsilon$-greedy exploration policy as prior work did [4–6]. This result confirms that both Q-learning and SF-learning benefit from the optimistic initialization strategy used in the presented simulation experiments.

Q-learning

The Q-learning algorithm [3] 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. Only in supporting S2A Fig, the Q-learning baseline (without no transfer) used a learning rate of 0.7. 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.

SF-learning

Algorithm 1 outlines an adaptation of the SF-learning algorithm [5] that incorporates optimistic initialization. Our implementation of SF-learning maintains a matrix $G$ of size $SA \times SA$, where $S$ is the number of states and $A$ the number of actions. Each row of this matrix $G$ approximates the SF

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

(1)

where the expectation in Eq (1) is defined over all infinite length trajectories that start with state $s$ and action $a$ and then follow the policy $\pi$. Each vector $e_{s_i, a_i}$ 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 (1) is equivalent to the definition presented by [5], with the distinction that one-hot bit vector features are used.
Consider a vector $v$ of length $SA$ storing one-step reward predictions in each entry:

$$v(s, a) = r(s, a)$$  \hspace{1cm} (2)

Algorithm 1 incrementally learns a matrix $G$ and vector $v$ such that Eq (1) and Eq (2) are satisfied. The iterates in lines 8 and 9 of Algorithm 1 are obtained by substituting $\phi_t = e_{s_t, a_t}$ and $\psi^\pi(s, a) = e^\top_{s, a} G = G(s, a)$ into Algorithm 1.

If the matrix $G$ is constructed such that Eq (1) exactly holds, then

$$Q^\pi(s, a) = (G(s, a))^\top v.$$  \hspace{1cm} (3)

The SF-learning algorithm learns an optimal policy by constructing $Q$-value prediction using Eq (3). Optimistic initialization can be integrated into SF-learning by initializing the matrix $G$ and vector $v$ such that $Q(s, a) = Q_{\text{max}}$ at the beginning of training. This can be accomplished by initializing $G = I$ and every entry $v_0(s, a) = Q_{\text{max}}$ (line 2 in Algorithm 1). Hence, we have for the $Q$-values at initialization

$$Q(s, a) = (G(s, a))^\top v = e^\top_{s, a} 1_{Q_{\text{max}}} = Q_{\text{max}},$$  \hspace{1cm} (4)

where $1$ 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 A lists the tested learning rates and the best performing learning rate setting for each simulation.

<table>
<thead>
<tr>
<th>Table A. Learning rates used for SF-learning.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tested $\alpha_{SF}$</td>
</tr>
<tr>
<td>SF-learning with SF transfer, orange curve in Fig 7B</td>
</tr>
<tr>
<td>SF-learning baseline, blue curve in Fig. 7B</td>
</tr>
<tr>
<td>Reward predictive with SF, green curve in Fig. 7B</td>
</tr>
<tr>
<td>SF-learning in S1 Fig and S2A Fig</td>
</tr>
<tr>
<td>SF-learning with SF transfer in S2A Fig</td>
</tr>
<tr>
<td>SF-learning with SF and reward transfer in S1 Fig and S2A Fig</td>
</tr>
<tr>
<td>SF-learning in S2B Fig</td>
</tr>
<tr>
<td>SF-learning with SF transfer in S2B Fig</td>
</tr>
</tbody>
</table>
Algorithm 1 SF-learning with optimistic initialization, adopted from [5]

1: Input: Learning rates $\alpha_{SF}, \alpha_{r} \in (0, 1]$, the maximum Q-value $Q_{max}$.
2: Initialize $G \leftarrow I$ and $v \leftarrow 1 * Q_{max}$ where $1$ is a vector of ones.
3: loop
4: Receive current state $s$ from MDP.
5: Select action $a = \arg \max_{a'} Q(s, a')$ with Eq (3).
6: Observe transition $(s, a, r, s')$.
7: $a^* \leftarrow \arg \max_{a'} Q(s', a')$ with Eq (3).
8: $G \leftarrow G + \alpha_{SF}(e_{sa} + \gamma G(s', a^*) - G(s, a))e_{s,a}^\top$
9: $v \leftarrow v + \alpha_{r}(v(s, a) - r)e_{s,a}$
10: end loop

Learning of Reward-Maximizing and Reward-Predictive State Abstractions

The learning simulations (presented in sub-section Learning to transfer multiple state abstractions) 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 $\phi$ is obtained with a low $l_{\text{maximizing}}(\phi)$ value$^{[7]}$.

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 $L_{\text{LSFM-mat}}$ presented by [8]:

$$L_{\text{LSFM-mat-scaled}} = \sum_{a \in A} \alpha_{w} ||\Phi w_{a} - \hat{r}_{a}||^2 + \alpha_{F} ||\Phi + \gamma \hat{P}_{a} \Phi F - \Phi F_{a}||^2,$$

(5)

where $\{\hat{P}_{a}, \hat{r}_{a}\}_{a \in A}$ are the transition and reward tables the non-parametric Bayesian model computes from the observed interaction data. Each row of the matrix $\Phi$ stores a latent feature vector that is associated with a particular state. The loss objective $L_{\text{LSFM-mat-scaled}}$ is optimized using the Adam gradient optimizer$^{[9]}$ with a learning rate of 0.01 and $\alpha_{F} = 0.01$. Otherwise Tensorflow’s$^{[10]}$ default parameters were used and the optimization process was initialized as described by [8]. After 1000 gradient update steps, the row-space of the real-valued matrix $\Phi$ 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 $\varepsilon_{r}$ and SF error $\varepsilon_{\psi}$ and a low loss value $l_{\text{predictive}}(\phi) = \varepsilon_{r} + \varepsilon_{\psi}$. During learning in

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

October 18, 2020 4/7
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) in the main article.

**Hyper-Parameters Influence Belief Space Size**

Supporting [S3 Fig](#) plots the average episode length and belief space size for different α and β settings for each model for the learning simulation presented in Fig 7.

Supporting [S3A Fig](#) and [S3C Fig](#) 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 influences 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 supporting [S3 Fig](#).

[S3B Fig](#) and [S3D Fig](#) 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.
Table B outlines the tested and best mixture model parameters for the simulations presented in Fig 7.

**Table B. Hyper-parameters tested for mixture model.**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Tested $\alpha$</th>
<th>Tested $\beta$</th>
<th>Best $\alpha$</th>
<th>Best $\beta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reward maximizing, orange curve in Fig 7A</td>
<td>$10^{-3}, 10^{-4}, 10^{-5}$, $10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}$</td>
<td>$10^{-3}$, $10^{-4}$</td>
<td>$10^{-3}$, $10^{-4}$</td>
<td>$10^{-3}$, $10^{-4}$</td>
</tr>
<tr>
<td>Reward predictive, green curve in Fig 7A</td>
<td>$10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}$</td>
<td>$10^{-5}$, $10^{-4}$</td>
<td>$10^{-5}$, $10^{-4}$</td>
<td>$10^{-5}$, $10^{-4}$</td>
</tr>
<tr>
<td>Reward predictive with SF-learning, green curve in Fig 7B</td>
<td>$10^{-9}, 10^{-5}, 10^{-1}$</td>
<td>$10^{-9}$, $10^{-5}, 10^{-1}$</td>
<td>$10^{-9}$, $10^{-5}, 10^{-1}$</td>
<td>$10^{-9}$, $10^{-5}, 10^{-1}$</td>
</tr>
</tbody>
</table>

**Hyper-Parameter for Guitar Task Experiment**

In the simulation presented in Fig 8, each learning algorithm interacted with each task for 50 episodes (trials). We make the assumption that same note cannot be repeated in the scale. This assumption ensures for simplicity that the fret board can be used as a Markovian state. For each algorithm, learning on the first task was initialized by setting both the $v$ vector and the $G$ matrix (defined in Eq (1) and (2)) were set to zero. On task 2, values were also reset to zero for the baseline algorithm. The reward-predictive model (green curve in Fig 8) only re-used a previously learned reward-predictive state abstraction and all latent SFs and reward weight vectors were also reset to zero at the beginning of training in task 2. The SF transfer algorithm (orange curve in Fig 8) re-used the previously learned SF matrix $G$ and the reward weight vector $w$ was reset to zero, similar to [6].

The SF transfer algorithm and the reward-predictive model used a learning rate of $\alpha_{SF} = 0.9$ and $\alpha_r = 0.9$. The SF-learning baseline algorithm used a learning rate of $\alpha_{SF} = 0.1$ and $\alpha_r = 0.9$. A total of nine learning rate combinations were tested by setting each learning rate to either 0.1, 0.5, or 0.9.

**References**


