Abstract of "Encoding Reusable Knowledge in State Representations" by Lucas Lehnert, Ph.D., Brown University, May 2021.

Reusing knowledge allows intelligent systems to learn solutions to complex tasks more quicker by avoiding re-learning the components of the solution from scratch. Recent advances in reinforcement-learning research have demonstrated that deep learning algorithms can solve complex tasks. However, achieving knowledge reuse characteristic of human behaviour has been elusive. Humans are adept at flexibly transferring knowledge between different tasks, whereas established reinforcement-learning algorithms are much more limited. Reusing knowledge in reinforcement-learning algorithms is a central, yet not well understood challenge. This dissertation addresses the question of which models allow an intelligent system to reuse knowledge and provides a partial solution. Viewing knowledge representations through the lens of representation learning, we show that models that are predictive of future reward outcomes implicitly encode reusable knowledge. Through a sequence of theoretical and empirical results, this dissertation discusses different state representations and presents connections to model-based reinforcement learning, model-free reinforcement learning, and successor features. Furthermore, different transfer-learning experiments are presented, demonstrating that representations that are predictive of future reward outcomes generalize across different tasks. Lastly, we introduce a clustering algorithm to learn representations that are predictive of future reward sequences for tasks with continuous state spaces. We demonstrate under which assumptions this clustering algorithm converges to an accurate model. Furthermore, on a visual control task, we demonstrate that this learned model generalizes across different tasks and can be used to accelerate learning. These results suggest that learning a model detailed enough to predict future reward outcomes prevents overfitting to one task and allows an agent to accelerate learning across previously unseen tasks.

Encoding Reusable Knowledge in State Representations

by

Lucas Lehnert

B. Sc., McGill University, 2015

M. Sc., McGill University, 2017

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island

May 2021

This dissertation by Lucas Lehnert is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date _____ _____
Michael L. Littman, Director

Recommended to the Graduate Council

Date _____ _____
Michael J. Frank, Reader
(Department of Cognitive, Linguistic &
Psychological Sciences)

Date _____ _____
Ellie Pavlick, Reader

Approved by the Graduate Council

Date _____ _____
Andrew G. Campbell
Dean of the Graduate School

Vita

Lucas Lehnert spent most of his childhood in Neustadt an der Weinstraße, Germany and moved to Montreal, Canada as a teenager. He completed his B.Sc. in computer science and M.Sc. at McGill University in 2015 and 2017 respectively. After his Masters, he joined Brown University and is currently completing his Ph.D. in computer science. He is also affiliated with the Carney Institute for Brain Science and his research focusses on reinforcement learning and its applications to computational cognitive neuroscience. His work on Successor Feature transfer, which also appears in this dissertation, was recognized by a Best Student Paper Award (Lifelong Learning: A Reinforcement Learning Approach workshop at ICML 2017).

Acknowledgements

First, I would like to thank my advisor and mentor, Michael L. Littman, for his guidance and advice in becoming a researcher. I am grateful for his support in developing my own research directions and fostering creating thinking. I would also like to thank my mentor and advisor Michael J. Frank for his guidance, support, and encouragement in conducting interdisciplinary research. I am grateful to him for showing me how to build bridges to research questions across disciplines and showing me how to communicate my ideas to a broad interdisciplinary audience. I am grateful to both for providing me with an open and supportive environment to launch my career as a researcher.

I am also grateful for being a part of the ICON training grant program. Here, I would also like to thank Professors Chris Moore and David Badre for their input on how to present my ideas to a broad interdisciplinary audience. Through the training grant and the neuropracticum I have learned much of what I know about neuroscience and cognitive science. Furthermore, I would like to thank Professor Stefanie Tellex for her advice in developing some of the early parts of the work presented in this dissertation. I am also grateful to Professors Doina Precup, Thomas Serre, Irina Rish, George Konidaris, and James Tompkin for discussions that further strengthened my work. I would also like to thank Professors Ellie Pavlick and Stephen Bach for serving on my dissertation committee and for their thoughtful questions that have contributed to this dissertation.

Moreover, I would like to thank Alana Jaskir, Rex Liu, Aneri Soni, Abdullah P. Rashed Ahmed, and Olga Lositsky. Through our discussions I have learned much about cognitive science and neuroscience and was able to strengthen my research in many ways. I would also like to thank Scott Daniel, Christopher Black, Zhiyan Wang, Sinda Fekir, Aarit Ahuja, Jason Leng, and Jovan Kemp for many valuable discussions that have helped me communicate my ideas to a neuroscience audience. I would also like to thank Yagnesh Revar, Dilip Arumugam, Nakul Gopalan, Jun Ki Lee, Christopher Grimm, Sam Saarinen, Arvind Yalavarti, and all members of the RLab and for many discussions that have contributed to my research.

Most importantly, I am grateful to my parents, Irina and Bernd, and my two brothers, Linus and Jonas, for their constant loving support.

Funding Support This dissertation was supported in part by NIMH T32MH115895 Training program for Interactionist Cognitive Neuroscience (ICON) and the ONR MURI PERISCOPE project.

Contents

List of Tables

List of Figures

Chapter 1

State Representations in Reinforcement Learning

Knowledge re-use enables intelligent systems to perform complex tasks quicker and avoids repeatedly learning the same solution from scratch. Recent advances in reinforcement learning (RL) [107] research have demonstrated that deep learning algorithms can learn how to maximize rewards in complex tasks [96, 73]. Still, how to achieve flexible knowledge transfer characteristic of human behaviour in artificially intelligent systems has been elusive. Understanding how to apply knowledge re-use to RL algorithms is an important step towards further scaling artificial intelligence. In this dissertation, we partially address the question of which models allow RL systems to re-use knowledge and present research to support the following thesis statement:

> *Learning an internal representation detailed enough to predict reward sequences prevents overfitting to one task and allows accelerated learning across previously unseen tasks.*

By viewing knowledge representations through the lens of state representation learning, this dissertation presents an approach to address the question of which models allow an agent to re-use knowledge. *State representations* [67, 85, 40] compress high dimensional state inputs into a smaller simplified latent state space. Using such a state representation, a reinforcement-learning system can re-use what it has learned for one input across all inputs that correspond to the same latent state. Consequently a reinforcement-learning system can learn faster because the system does not consider each high-dimensional state input separately [26, 3]. State representations provide a framework to mitigate this "curse of dimensionality" [18] in reinforcement learning. Because state representations determine how an RL system generalizes across inputs, a state representation needs

to be carefully constructed to facilitate faster learning instead of constraining an RL system's ability to obtain an optimal solution. This dissertation studies different representation types and discusses in which cases state representations can be re-used across different tasks. Specifically, we find that state representations which are detailed enough to predict reward sequences implicitly encode task knowledge. Furthermore, we demonstrate that these *reward-predictive* representations are suitable for re-use across different tasks while other state representation types are much more constrained and overfit to a specific task solution. Because reward-predictive representations encode task knowledge, transferring reward-predictive representations is a form of knowledge re-use.

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. This structure can be generalized from a left-hand-drive car to a right-hand-drive car [40, 26] and a driver does not have to learn how to drive and how to generalize across different situations from scratch.

Prior work presents algorithms that reuse state representations to initialize learning across different reward specifications [11, 12, 123, 61] or constructs state representations using different clustering criteria [67, 2, 3]. The presented research follows a different methodology: By analyzing which properties different latent state spaces are predictive of, different models of generalization are compared leading to new connections between state representation learning, Successor Features [11], model-free RL, and model-based RL [108]. In addition, the conducted transfer experiments suggest that reward predictive state representations, which are predictive of future reward outcomes, generalize across tasks that vary in both their transitions and rewards.

This dissertation is organized into four chapters. Chapter 2 provides an introduction to Successor Features [11] and presents connections between learning Successor Features and temporal difference (TD) learning [107, Chapter 6]. In addition, transfer experiments conducted on finite state tasks suggest that learning Successor Features is akin to model-free RL, a trial-and-error approach to learning how to behave optimally. While re-using previously learned successor features speeds up learning, the re-used successor features are re-learned and adjusted for each individual task. Then, Chapter 3 introduces Linear Successor Feature Models, a model which ties learning successor features

to learning reward-predictive representations. Reward-predictive representations construct a latent state space detailed enough for an RL system to predict reward sequences for arbitrary decision sequences. This property of reward-predictive representations ties them to model-based RL, an approach where RL systems use predictions of reward sequences to plan their future decisions. Because LSFMs learn reward-predictive representations through TD-learning, these models provide a novel link between model-based RL and TD-learning. Reward-predictive representations only provide information about how to generalize across inputs and do not contain any specifics about transitions or rewards. Consequently, reward-predictive representations are robust to transfer across tasks where these specifics in transitions and rewards change. This property stands in contrast to prior work on successor features, which only considers generalization across tasks that vary in rewards but share common transitions [11, 12, 123, 61, 75, 89, 100]. Using these models, Chapter 4 supports the stated thesis with a sequence of transfer simulations and compares reward predictive state representations with other representation models. These results suggest that reward predictive state representations are suitable for transfer, while other representation models, for example models that only predict the optimal policy in one task, do not generalize across different tasks. Consequently, learning a model detailed enough to predict future reward outcomes prevents overfitting to one specific task and allows one to accelerate learning across previously unseen tasks. Chapter 5 presents an algorithm to learn reward-predictive representations for tasks with arbitrary state spaces. The theoretical analysis presented in Chapter 3 already provides a foundation for this case, yet the representation learning algorithms presented in the previous chapters are designed for finite state tasks. We demonstrate how to combine this representation learning algorithm with deep learning techniques [48] to extract reward-predictive representations from two control tasks where the state is described by an image. Furthermore, we establish under which assumptions this algorithm converges to a reward-predictive representations that constructs as few latent states as possible. Lastly, we study empirically different transition datasets and find that a reward-predictive representation can only be found if the provided transition dataset outlines all aspects of the control task at hand.

The content presented in Chapter 2 previously appeared in [64]. The content of Chapter 3 and partially Chapter 2 is published in [63]. Chapter 4 is published in [65]. The results presented in Chapter 5 appear in this dissertation for the first time.

1.1 Reinforcement Learning

Before introducing state representations, this section provides a brief introduction to RL that will serve as a foundation for the remaining chapters. The interaction between a RL system, also called *agent*, is formalized as a Markov Decision Process (MDP). For example, consider a self-driving car which makes decisions based on its sensory data. We refer to all sensor inputs at a particular point in time as the *state* of the self-driving car. This state s lies in the state space $\mathcal{S}$, which is the set of all possible states. At every time point, the self-driving car makes a decision, for example to apply the brakes or to steer at a particular angle. We refer to such decisions as *actions* and assume that the space of all actions, called the *action space* $\mathcal{A}$, is finite. Once a decision is made and an action is chosen, the self-driving car will observe the effect of its action and receive a state update. This change is referred to as a state *transition* and each transition is evaluated with a reward, a single scalar number.

This framework makes two fundamental assumptions:

1. Time is a discrete entity and changes in states are counted in steps.
2. The outcome of a transition can be conditioned on the previous state and the selected action.

The second assumption is commonly referred to as the *Markov assumption* and implies that observing the full state is sufficient to make an optimal decision. An intelligent agent would not have to reason about past experiences and does not have to take into consideration how the agent arrived at the current state [53]. This dissertation also assumes a finite action space $\mathcal{A}$ and leaves extensions to arbitrary action spaces to future work. However, the state space $\mathcal{S}$ is assumed to be an arbitrary set, including finite sets or uncountably infinite sets. Formally, an MDP is defined as follows.

Definition 1 (Markov Decision Process)**.** *A Markov Decision Process (MDP) is a five-tuple* $M = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$ *where*

1. *the state space* $\mathcal{S}$ *is an arbitrary set of states,*
2. *the action space* $\mathcal{A}$ *is a (finite) set of decisions an agent can select from,*
3. *the transition function* $p(s'|s, a)$ *describes the probability or density with which a transition from state* s *to state* s' *occurs when action* a *is selected,*
4. *the reward function* r *describes the reward for each transition with* $r(s, a, s')$*, and*

5. *the discount factor* $\gamma \in [0, 1)$ *is a single scalar number describing a trade-off between short-term and long-term rewards. If* γ *is close to zero, then an agent should only consider short-term rewards and if* γ *is close to one, then an agent should consider long-term rewards that lie further into the systems future.*

A *policy* π describes an agent's decision-making strategy and specifies a probability $\pi(s, a)$ of selecting an action $a \in \mathcal{A}$ at state $s \in \mathcal{S}$. A policy's performance is described by the value function

$$V^{\pi}(s) = \mathbb{E}_{p,\pi}\left[\sum_{t=1}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) \middle| s_1 = s\right], \tag{1.1}$$

which predicts the expected discounted return generated by selecting actions according to π when trajectories are started at the state s. The expectation[1] in Equation (1.1) is computed over all infinite length trajectories that select actions according to π and start at state s. Similarly, the Q-value function is defined as

$$Q^{\pi}(s, a) = \mathbb{E}_{p,\pi}\left[\sum_{t=1}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) \middle| s_1 = s, a_1 = a\right]. \tag{1.2}$$

The expectation of the Q-value function in Equation (1.2) is computed over all infinite length trajectories that start at state s with action a and then follow the policy π.

Because transitions between states are assumed to be markovian, value functions can be re-written in a recursive form:

$$\begin{aligned}
V^{\pi}(s) &= \mathbb{E}_{p,\pi}\left[\sum_{t=1}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) \middle| s_1 = s\right] && (1.3)\\
&= \mathbb{E}_{p,\pi}\left[r(s_1, a_1, s_2) + \gamma \sum_{t=2}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) \middle| s_1 = s\right] && (1.4)\\
&= \mathbb{E}_{p,\pi}\left[r(s_1, a_1, s_2) + \gamma \mathbb{E}_{p,\pi}\left[\sum_{t=2}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) \middle| s_2\right] \middle| s_1 = s\right] && (1.5)\\
&= \mathbb{E}_{p,\pi}\left[r(s_1, a_1, s_2) + \gamma V^{\pi}(s_2) | s_1 = s\right] && (1.6)
\end{aligned}$$

where Equation (1.5) follows by linearity of the expectation operator and Equation (1.6) follows by substitution with Equation (1.1). Equation (1.6) is called the Bellman fix point equation [20] and

[1] The subscript of the expectation operator $\mathbb{E}$ denotes the probability distributions or densities over which the expectation is computed.

any solution to this equation is a value function that predicts the expected discounted return.

The Bellman fix point equation can also be used to test if the value function of a policy maximizes the discounted return at every state and to determine if a policy π^* is optimal. If a policy π^* is optimal, then this policy deterministically selects the action that generates the highest discounted return and

$$\pi^*(s, a^*) = 1 \text{ if } a^* = \arg\max_a Q^{\pi^*}(s, a), \tag{1.7}$$

where Q^{π^*} is the Q-value function of the policy π^*. If the policy π^* would select another action that does not generate the highest possible return, then this policy does not maximize the discounted return at every state and be sub-optimal. By this argument, an optimal policy is always greedy with respect to its own Q-value function. Further, the value of an optimal policy π^* at any state s is

$$V^{\pi^*}(s) = \max_a Q^{\pi^*}(s, a). \tag{1.8}$$

Repeating the derivation in Equation (1.6) for Q^{π^*} leads to the Bellman optimality condition:

$$\begin{aligned}
Q^{\pi^*}(s, a) &= \mathbb{E}_{p,\pi^*}\left[\sum_{t=1}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) \middle| s_1 = s, a_1 = a\right] && (1.9)\\
&= \mathbb{E}_{p,\pi^*}\left[r(s_1, a_1, s_2) + \gamma \sum_{t=2}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) \middle| s_1 = s, a_1 = a\right] && (1.10)\\
&= \mathbb{E}_{p}\left[r(s_1, a_1, s_2) + \gamma \mathbb{E}_{p,\pi^*}\left[\sum_{t=2}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) \middle| s_2\right] \middle| s_1 = s, a_1 = a\right] && (1.11)\\
&= \mathbb{E}_{p}\left[r(s_1, a_1, s_2) + \gamma V^{\pi^*}(s_2) \middle| s_1 = s, a_1 = a\right] && (1.12)\\
&= \mathbb{E}_{p}\left[r(s, a, s') + \gamma \max_{a' \in \mathcal{A}} Q^{\pi^*}(s', a') \middle| s, a\right], && (1.13)
\end{aligned}$$

where line (1.13) was obtained by substitution with Equation 1.8. The expectation in Equation (1.13) only depends on the transition function p and ranges over all possible states s' that can be reached from state s by selecting action a. If the Q-value function of a policy π^* satisfies Equation (1.13), then the policy π^* is optimal and maximizes the discounted return at every state [20, Chapter 2].

1.1.1 Temporal Difference Learning

Temporal Difference (TD) learning [102] is a method commonly used to design algorithms that either learn the value function of particular policy or find the value function of the optimal policy.

Most TD-learning algorithms are incremental and observe a stream to iteratively improve their value predictions. For example, the TD(0) algorithm [102] samples a sequence of transition quadruples (s, a, r, s') using a fixed policy π and moves its value predictions towards a solution that satisfies the Bellman fix point. Using moving average updates, a particular value estimate $v_t(s)$ at a state s at update iteration t is updated with

$$v_{t+1}(s) = (1 - \alpha)v_t(s) + \alpha(r + \gamma v_t(s')), \tag{1.14}$$

where $\alpha \in (0, 1]$ is a learning rate parameter. Assuming that the state space of an MDP is a finite set, the value estimates $v_t(s)$ can be stored in a table. If $v_{t+1}(s) = V^\pi(s)$ for all states s, then the update iterate in Equation (1.14) does not change the value estimate and the TD(0) has converged. The update iterate in Equation (1.14) can be re-written as

$$v_{t+1}(s) = v_t(s) + \alpha \underbrace{(r + \gamma v_t(s') - v_t(s))}_{=\delta}, \tag{1.15}$$

where the term δ is called the TD-error. Intuitively, this TD-error describes the magnitude and direction by which value predictions have to be corrected.

TD(0) is a value prediction algorithm that learns to predict the discounted return for one fixed policy π. Q-learning [115] re-uses these ideas and implements an algorithm to search for an optimal policy. Similar to TD(0), Q-learning [115] samples a sequence of transition quadruples (s, a, r, s') using some control policy η. This control policy η need not be fixed and is allowed to change over time. The Q-learning algorithm then iteratively improves its Q-value estimates of the optimal policy over time using the iterate

$$q_{t+1}(s, a) = (1 - \alpha)q_t(s, a) + \alpha(r + \gamma \max_{a'} q_t(s', a')), \tag{1.16}$$

where $\alpha \in (0, 1]$ is a learning rate parameter. The iterate in Equation (1.16) differs from TD(0) only in that it maintains estimates for Q-values rather than state values and that the update target $r + \gamma \max_{a'} q_t(s', a')$ is constructed using the action with the highest value prediction. If $q_t(s, a) = Q^{\pi^*}(s, a)$ for all states s and action a, then the iterate in Equation (1.16) will not change the value estimates and Q-learning has found a solution to the Bellman optimality condition (1.13). Because

the optimal policy is greedy with respects to its own value function, Q-learning is an algorithm which computes the optimal policy given the observed transition data.

1.2 Predictive State Representations

Consider a self-driving car that collects and combines input data from cameras, radar sensors, and GPS, for example. While this sensor data describes the state of a self-driving car in great detail, deciding on a sequence of control commands such as "steer left" or "apply brakes" becomes very difficult, because many different states would have to be taken into consideration to make a decision. This "curse of dimensionality" can be overcome by compressing high dimensional sensor data into a lower dimensional latent state space. In the self-driving car example, if the car is following another vehicle that is slowing down, detecting brake lights is sufficient to make the decision to slow down and avoid an accident. Other information such as the colour of the car in-front can be ignored. This ability of generalizing across different states can be modelled using state representations, which compress the state space of an MDP into a lower-dimensional, more tractable latent state space.

A latent state space $\mathcal{S}_\phi$ is constructed using a *state representation* function $\phi : \mathcal{S} \to \mathcal{S}_\phi$. A state representation can be understood as a compression of the state space, because two different states s and $\tilde{s}$ can be assigned to the same latent state $\phi(s) = \phi(\tilde{s})$. In this case, the state representation ϕ aliases s and $\tilde{s}$. Figure 1.1 provides an overview of the state representation types studied in this dissertation.

Reward-Predictive State Representations A reward-predictive state representation constructs a latent state space that is predictive of any future expected reward sequence $r_1, ..., r_t$ that is observed after executing any abitrary 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 this expected reward sequence

$$(r_1, ..., r_t) = \mathbb{E}[(R_1, ..., R_t)|s, a_1, ..., a_t]. \tag{1.17}$$

The expectation in Equation (1.17) 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

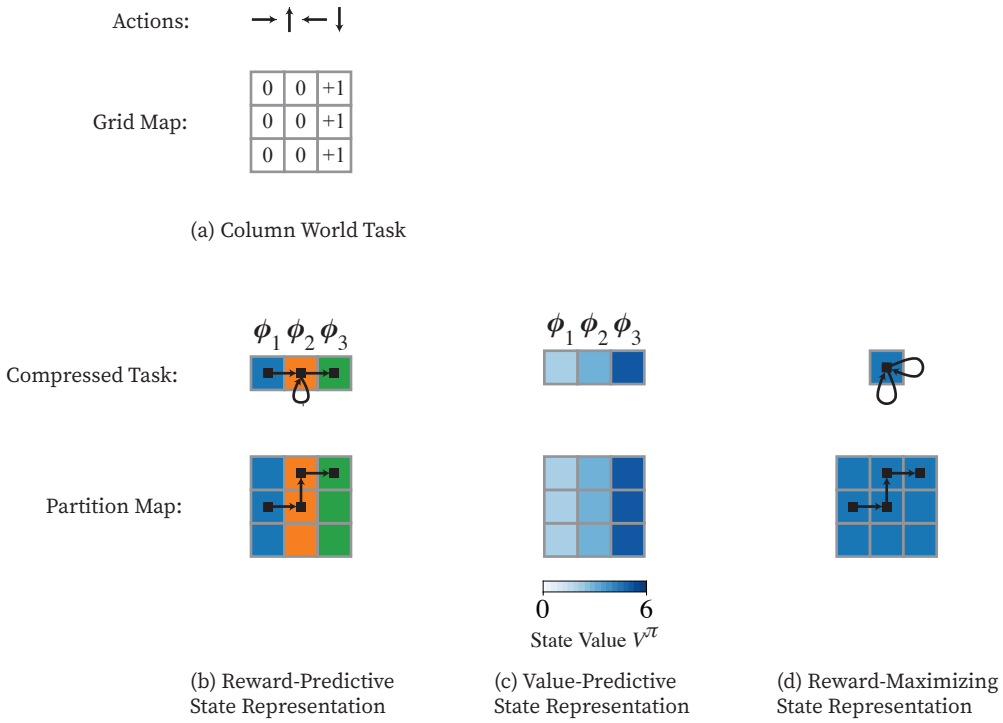


Figure 1.1: State Representations Construct Lower Dimensional Latent State Spaces. 1.1(a): The column world example is a 3×3 grid world where an agent can move up (action $\uparrow$), down (action $\downarrow$), left (action $\leftarrow$), or right (action $\rightarrow$) to adjacent grid cells and entering the right column is rewarded. Rewards are indicated by the numbers in the individual grid cells. 1.1(b): A reward-predictive state representation, 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. 1.1(c): The lower panel presents a matrix plot of the state values V^π for a policy that selects actions uniformly at random. Grid cells of the same column have equal distance to the rewarding column and thus equal state values. Because this state representation only generalizes across states of the same column, the constructed latent state space can be used to predict the value function V^π as well. In this example, there is no difference across which states a value-predictive and a reward-predictive state representation generalizes. In Chapter 2 we will present examples illustrating how value-predictive state representations compress the state space further than reward-maximizing state representations. 1.1(d): A reward-maximizing state representation compresses all states into one latent state. Because the optimal action is to move right in every grid cell, a reward-maximizing state representation compresses the entire grid into one latent state. An optimal policy can still be found in this case, because only the move right action receives positives reinforcement while all other actions are not reinforced. This representation maps any path in the 3×3 grid to a sequence of self-loops in the latent state space where different actions are chosen.

abstraction needs to satisfy for any start state s and action sequence $a_1, ..., a_n$ that

$$\mathbb{E}[(R_1, ..., R_t)|s, a_1, ..., a_t] = \mathbb{E}[(R_1, ..., R_t)|\phi(s), a_1, ..., a_t]. \tag{1.18}$$

Equation 1.18, the expectation on the right-hand side is conditioned on the latent state $\phi(s)$ and is computed over all possible trajectories in the latent space $\mathcal{S}_\phi$ constructed by the representation ϕ. In contrast, the expectation on the left-hand side is computed over all possible trajectories that follow the action sequence $a_1, ..., a_t$ and start at state s. A reward-predictive state representation 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. Figure 1.1(a) presents a grid world example where a 3×3 grid is compressed into three latent states $\{\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \boldsymbol{\phi}_3\}$. These three latent states effectively construct a smaller 3×1 grid world. The schematic also illustrates a trajectory that starts in the left (blue) column and follows the action sequence "move right", "move up", "move right" and generates a reward sequence of $0, 0, 1$. One can observe that the state representation is constructed such that mapping this trajectory into the compressed task preserves the reward sequence of $0, 0, 1$. In the compressed task, only the transition from $\boldsymbol{\phi}_2$ to $\boldsymbol{\phi}_3$ is rewarded and the trajectory is mapped to the latent state sequence $\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \boldsymbol{\phi}_2, \boldsymbol{\phi}_3$ (the "move up" action is realized as a self-loop).

Value-Predictive State Representations A value-predictive state representation constructs a latent state space that is predictive of Q-values. Figure 1.1(c) presents such a state representation on the 3×1 grid example. Suppose each latent state is represented as a one-hot bit vector[2] in three dimensions and $\mathcal{S}_\phi = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3\}$. Because each grid cell is mapped to a one-hot bit vector, $\phi(s) = \boldsymbol{e}_i$ for some index i. If the state representation ϕ is value-predictive, then

$$Q^\pi(s, a) = (\phi(s))^\top \boldsymbol{q}_a = \boldsymbol{e}_i^\top \boldsymbol{q}_a, \tag{1.19}$$

where $\boldsymbol{q}_a$ is a real-valued vector in $\mathbb{R}^3$ that is associated with the action a. Because the state representation ϕ outputs one-hot bit vectors, each entry of the vector $\boldsymbol{q}_a$ contains the Q-value associated with one of the three columns. Note that prior work [104, 59] provides heuristics for constructing

[2] A one-hot bit vector $\boldsymbol{e}_i$ is a column vector of zeros but with the ith entry set to one. Vectors are denoted with bold lower-case letters. Matrices are denoted with bold capitalized letters.

such value-predictive state representations. Because value-predictive state representations are only required to be predictive of a particular policy π, they implicitly depend on the policy π.

Reward-Maximizing State Representations A reward-maximizing state representation preserves an agent's ability to maximize reward in one MDP. For example, in Figure 1.1(d) a reward-maximizing state representation compresses the entire state space of the 3×3 grid into a single state. Because only the move-right action is rewarded in this example, compressing the 3×3 grid into one latent state still provides the agent with the ability to learn that only the move-right action should be used. 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.

Learning State Representations Figure 1.1 presents examples of the studied state representations. In the following chapters, we will present learning algorithms that will search the space of all possible state representations to identify approximations of reward-predictive, value-predictive, and reward-maximizing state representations.

Chapter 2

Successor Features as a Model-Free Mechanism

Successor features (SF) are representations used for factoring Q-values into a SF vector and a reward weight vector that models the reward function of an MDP. Because the same SFs can be combined with different reward weight vectors, SFs are used for fast Q-value computation for different reward functions. Consequently, SFs are a suitable for transfer across tasks with different rewards but common transitions. In this chapter we provide an introduction to SFs, outline how learning SFs is a form of TD-learning, and discuss how learning SFs is a model-free RL mechanism. Lastly, we present a set of simulations on finite MDPs to illustrate the benefits of re-using SFs across tasks that vary in their rewards but have the same transitions. This chapter serves as a baseline to contrast reward-predictive representations with the SF framework and model-free RL.

SFs are a generalization of the Successor Representation (SR) [29], a state representation that predicts the frequency with which states are visited under a specific policy. For finite state and action spaces, the transition probabilities while selecting actions according to a policy π can be written as a stochastic transition matrix $\boldsymbol{P}^{\pi}$ where all rows sum to one. If the start state with index s is represented as a one-hot bit vector $\boldsymbol{e}_s$, the probability distribution of reaching a state after one time step of executing policy π can be written as a row vector $\boldsymbol{e}_s^{\top}\boldsymbol{P}^{\pi}$. After t time steps, the probability distribution over states can be written as the vector $\boldsymbol{e}_s^{\top}(\boldsymbol{P}^{\pi})^t$. Suppose the path across the state space has a random length that follows the Geometric distribution with parameter γ: At each time step, a biased coin is flipped and the path continues with probability γ. In this model, the probability vector of reaching different states in t time steps is $(1-\gamma)\gamma^{t-1}\boldsymbol{e}_s^{\top}(\boldsymbol{P}^{\pi})^t$. Omitting

the factor $(1-\gamma)$, the SR recursively computes the marginal probabilities over all time steps:

$$\boldsymbol{\Psi}^{\pi} = \sum_{t=1}^{\infty} \gamma^{t-1} (\boldsymbol{P}^{\pi})^{t-1} = \boldsymbol{I} + \gamma \boldsymbol{P}^{\pi} \boldsymbol{\Psi}^{\pi}. \tag{2.1}$$

Each entry (i, j) of the matrix $(1-\gamma)\boldsymbol{\Psi}^{\pi}$ contains the marginal probability across all possible durations of transitioning from state i to state j. Intuitively, the entry (i, j) of the matrix $\boldsymbol{\Psi}^{\pi}$ can be understood as the frequency of encountering state j when starting a path at state i and following the policy π. An *action conditioned SR* describes the marginal probability across all possible durations of transitioning from state i to state j, but first a particular action a is selected, and then the policy π is followed:

$$\boldsymbol{\Psi}_a^{\pi} \overset{\text{def.}}{=} \sum_{t=1}^{\infty} \gamma^{t-1} \boldsymbol{P}_a (\boldsymbol{P}^{\pi})^{t-2} = \boldsymbol{I} + \gamma \boldsymbol{P}_a \boldsymbol{\Psi}^{\pi}, \tag{2.2}$$

where $\boldsymbol{P}_a$ is the stochastic transition matrix describing all transition probabilities when action a is selected. Because $\boldsymbol{P}_a$ is a stochastic matrix, it can be shown that $\boldsymbol{\Psi}^{\pi}$ is invertible and that there exists a one-to-one correspondence between each transition matrix and action-conditional SR matrix.[1]

SFs generalize this idea to arbitrary representations. Given a state representation function ϕ that maps each state s to a real-valued vector $\boldsymbol{\phi}_s$, the SF is a column vector defined for each state and action pair [61, 123] and

$$\boldsymbol{\psi}^{\pi}(s, a) = \mathbb{E}_{p,\pi} \left[\sum_{t=1}^{\infty} \gamma^{t-1} \boldsymbol{\phi}_{s_t} \middle| s_1 = s, a_1 = a \right], \tag{2.3}$$

where the expectation in Equation (2.3) is computed over all infinite length trajectories that follow the policy π and start in state s with action a. If following the policy π leads to encountering a particular state vector $\boldsymbol{\phi}_{s'}$ many times, then this state vector will occur in the summation in Equation (2.3) many times.[2] Depending on the state representation ϕ, the vector $\boldsymbol{\psi}^{\pi}(s, a)$ will be similar to the state vector $\boldsymbol{\phi}_{s'}$ if the state s' is visited often and $\boldsymbol{\psi}^{\pi}(s, a)$ will be dis-similar to a state vector $\boldsymbol{\phi}_{\tilde{s}}$ if the $\tilde{s}$ is visited infrequently. A SF vector $\boldsymbol{\psi}^{\pi}$ can be understood as a statistic measuring how frequent different latent states are encountered.

The following two sections draw connections between SFs, reward-predictive, and value-predictive

[1] By Equation (2.1), $\boldsymbol{\Psi}^{\pi} = \boldsymbol{I} + \gamma \boldsymbol{P}^{\pi} \boldsymbol{\Psi}^{\pi} \iff \boldsymbol{I} = (\boldsymbol{I} - \gamma \boldsymbol{P}^{\pi}) \boldsymbol{\Psi}^{\pi} \iff (\boldsymbol{\Psi}^{\pi})^{-1} = \boldsymbol{I} - \gamma \boldsymbol{P}^{\pi}$. Equation (2.2) outlines how to construct $\boldsymbol{\Psi}_a^{\pi}$ from $\boldsymbol{P}_a$ for all actions. The reverse direction follows from $\boldsymbol{\Psi}_a^{\pi} = \boldsymbol{I} + \gamma \boldsymbol{P}_a \boldsymbol{\Psi}^{\pi} \iff (\boldsymbol{\Psi}_a^{\pi} - \boldsymbol{I})(\boldsymbol{\Psi}^{\pi})^{-1} / \gamma = \boldsymbol{P}_a$.

[2] To simplify notation, the vector $\boldsymbol{\phi}_s$ always denotes the output of ϕ at state s.

state representations and outline under what assumptions learning SFs is equivalent to learning reward- or value-predictive state representations.

Barreto et al. present SFs as a factorization of the Q-value function for an arbitrary fixed policy π. This factorization assumes a state-action representation function $\xi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^m$ that serves as a basis function for one-step reward predictions and

$$\forall s \in \mathcal{S},\ \forall a \in \mathcal{A},\ \boldsymbol{\xi}_{s,a}^\top \boldsymbol{w} = \mathbb{E}_p \left[r(s,a,s') | s, a \right], \tag{2.4}$$

where $\boldsymbol{\xi}_{s,a} = \xi(s,a)$. Using the state-action representation function, the Q-value function can be factored:

$$\begin{aligned}
Q^\pi(s,a) &= \mathbb{E}_{p,\pi}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t, s_{t+1}) \middle| s_1 = s, a_1 = a \right] && (2.5)\\
&= \mathbb{E}_{p,\pi}\left[\sum_{t=1}^{\infty} \gamma^{t-1} \boldsymbol{\xi}_{s_t,a_t}^\top \boldsymbol{w} \middle| s_1 = s, a_1 = a \right] && \text{(by (2.4))} \quad (2.6)\\
&= \underbrace{\mathbb{E}_{p,\pi}\left[\sum_{t=1}^{\infty} \gamma^{t-1} \boldsymbol{\xi}_{s_t,a_t}^\top \middle| s_1 = s, a_1 = a \right]}_{= \left(\boldsymbol{\psi}_{\mathrm{SA}}^\pi(s_1,a_1)\right)^\top} \boldsymbol{w} && (2.7)\\
&= \left(\boldsymbol{\psi}_{\mathrm{SA}}^\pi(s_1,a_1)\right)^\top \boldsymbol{w}. && \text{(where } s_1 = s,\ a_1 = a) \quad (2.8)
\end{aligned}$$

In Equation (2.8), the SF vector $\boldsymbol{\psi}_{\mathrm{SA}}^\pi$ is defined as

$$\boldsymbol{\psi}_{\mathrm{SA}}^\pi(s,a) \overset{\text{def.}}{=} \mathbb{E}_{p,\pi}\left[\sum_{t=1}^{\infty} \gamma^{t-1} \boldsymbol{\xi}_{s_t,a_t} \middle| s_1 = s, a_1 = a \right]. \tag{2.9}$$

This SF definition is different from the definition presented in Equation (2.3) in that a state-action representation function ξ is used instead of a state representation function ϕ. By line (2.8), the state-action SF $\boldsymbol{\psi}_{\mathrm{SA}}^\pi$ is a basis function that allows accurate predictions of the Q-value function Q^π. Consequently, the representation $\boldsymbol{\psi}_{\mathrm{SA}}^\pi$ is a value-predictive state representation because it is designed to construct a latent feature space that supports accurate predictions of the Q-value function Q^π.

2.1 Connection to Linear Temporal Difference Learning

Algorithms that learn SFs can be derived similarly to linear TD-learning [105, Chapter 8.4]. In linear TD-learning, for example linear Q-learning or SARSA, Q-values are represented with

$$Q^{\pi}(s, a; \boldsymbol{\theta}) = \boldsymbol{\xi}_{s,a}^{\top} \boldsymbol{\theta}, \tag{2.10}$$

where $\boldsymbol{\theta}$ is a real-valued weight vector that does not depend on a state s or action a. Linear TD-learning obtains the parameter vector $\boldsymbol{\theta}$ by minimizing the mean squared value error

$$\mathrm{VE}(\boldsymbol{\theta}) = \sum_{s,a,r,s'} \mu(s, a, r, s') \left(Q_{\boldsymbol{\theta}}^{\pi}(s, a; \boldsymbol{\theta}) - y_{s,a,r,s'}\right)^2. \tag{2.11}$$

Equation (2.11) averages prediction errors with respect to some distribution μ with which transitions (s, a, r, s') are sampled. The prediction target

$$y_{s,a,r,s'} = r + \gamma \sum_{a'} b(s', a') Q^{\pi}(s', a'; \boldsymbol{\theta}) \tag{2.12}$$

varies by which real-valued function b is used. For example, to find the Q-values for the optimal policy linear Q-learning uses an indicator function $b(s, a) = \mathbf{1}[a = \arg\max_a Q^{\pi}(s, a; \boldsymbol{\theta})]$ so that $y_{s,a,r,s'} = r + \gamma \max_{a'} Q^{\pi}(s', a'; \boldsymbol{\theta})$ [115]. For Expected SARSA [107, Chapter 6.6], which evaluates a fixed policy π, the target can be constructed using $b(s, a) = \pi(s, a)$, where $\pi(s, a)$ specifies the probability with which a is selected at state s. When computing a gradient of $\mathrm{VE}(\boldsymbol{\theta})$ the *prediction target* $y_{s,a,r,s'}$ is considered a constant. For an observed transition (s, a, r, s'), the parameter vector is updated using the rule

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left(Q^{\pi}(s, a; \boldsymbol{\theta}_t) - y_{s,a,r,s'}\right) \boldsymbol{\xi}_{s,a}, \tag{2.13}$$

where α is a learning rate and the subscript t tracks the update iteration. Algorithm 1 outlines the linear Q-learning algorithm.

Similarly, a SF-learning algorithm can be derived by defining the mean squared SF error [64]

$$\mathrm{SFE}(\boldsymbol{\psi}_{\mathrm{SA}}^{\pi}) = \sum_{s,a,r,s'} \mu(s, a, r, s') ||\boldsymbol{\psi}_{\mathrm{SA}}^{\pi}(s, a) - \vec{\boldsymbol{y}}_{s,a,r,s'}||^2. \tag{2.14}$$

Algorithm 1 Linear Q-Learning [105, Chapter 8.4]

Input: A state representation function ξ, a control policy π, and a learning rate $\alpha \in (0, 1]$.
Initialize $\boldsymbol{\theta}$.
loop
  Receive current state s from MDP.
  Select action a using the control policy π.
  Observe transition (s, a, r, s').
  $y_{s,a,r,s'} \leftarrow r + \gamma \max_{a'} Q(s', a'; \boldsymbol{\theta})$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \left(Q(s, a; \boldsymbol{\theta}) - y_{s,a,r,s'}\right) \boldsymbol{\xi}_{s,a}$

Because the SF $\boldsymbol{\psi}^{\pi}_{\mathrm{SA}}(s, a)$ is a vector of dimension m, the target

$$\vec{\boldsymbol{y}}_{s,a,r,s'} = \boldsymbol{\xi}_{s,a} + \gamma \sum_{a'} b(s', a') \boldsymbol{\psi}^{\pi}_{\mathrm{SA}}(s', a') \tag{2.15}$$

is also a vector but can be constructed similarly to the usual value-prediction target $y_{s,a,r,s'}$. Assuming that SFs are approximated linearly using the basis function ξ,

$$\boldsymbol{\psi}^{\pi}_{\mathrm{SA}}(s, a; \boldsymbol{G}) = \boldsymbol{G}\boldsymbol{\xi}_{s,a}, \tag{2.16}$$

where $\boldsymbol{G}$ is a square matrix. Computing the gradient of $\mathrm{SFE}(\boldsymbol{\psi}^{\pi}_{\mathrm{SA}})$ with respect to $\boldsymbol{G}$ results in an update rule similar to linear TD-learning:

$$\boldsymbol{G}_{t+1} = \boldsymbol{G}_t + \alpha \left(\boldsymbol{\psi}^{\pi}_{\mathrm{SA}}(s, a; \boldsymbol{G}_t) - \vec{\boldsymbol{y}}_{s,a,r,s'}\right) \boldsymbol{\xi}^{\top}_{s,a}. \tag{2.17}$$

Assuming the reward condition in Equation (2.4) holds, both linear TD-learning and SF-learning produce the same value-function sequence.

Proposition 1 (Linear TD-learning and SF-learning Equivalence)**.** *Consider an MDP M and a basis function ξ such that $r(s, a) = \boldsymbol{\xi}^{\top}_{s,a}\boldsymbol{w}$ for all states s and actions a. Suppose both iterates in Equation* (2.13) *and in Equation* (2.17) *use the same function b to construct prediction targets and are applied for the same trajectory $(s_1, a_1, r_1, s_2, a_2, ...)$. If $\boldsymbol{\theta}_0 = \boldsymbol{G}_0\boldsymbol{w}$, then*

$$\forall t > 0,\ \boldsymbol{\theta}_t = \boldsymbol{G}_t\boldsymbol{w}. \tag{2.18}$$

Proposition 1 proves that both linear TD-learning and linear SF-learning generate identical

value-function estimates on the same trajectory. A formal proof of Proposition 1 is listed in Appendix A.1. Because linear TD-learning need not converge to an optimal solution, the SF iterate in Equation (2.17) also need not converge to an optimal solution. The tabular case, in which convergence can be guaranteed, is a sub-case of the presented analysis: For finite state and action spaces, a basis function ξ can be constructed that outputs a one-hot bit vector of dimension n, where n is the number of all state and action pairs. In this light, learning SFs is akin to learning a value function in model-free RL.

A linear SF-Learning algorithm can be implemented similar to Linear Q-learning but using the SF update rule stated in Eq. (2.17) together with an update rule to approximate the weight vector $\boldsymbol{w}$. Algorithm 2 outlines the linear SF-learning algorithm.

Algorithm 2 Linear SF-Learning [10]

Input: A state representation function ξ, a control policy π, and a learning rate $\alpha_{\text{SF}}, \alpha_{\text{r}} \in (0, 1]$.
Initialize matrix $\boldsymbol{G}$ and vector $\boldsymbol{w}$.
loop
  Receive current state s from MDP.
  Select action a using the control policy π.
  Observe transition (s, a, r, s').
  $a^* \leftarrow \arg\max_{a'}(\boldsymbol{\psi}_{\text{SA}}(s', a'; \boldsymbol{G}))^\top \boldsymbol{w}$ (because $Q(s, a) \approx \boldsymbol{\psi}_{\text{SA}}(s', a'; \boldsymbol{G}))^\top \boldsymbol{w}$)
  $\vec{\boldsymbol{y}}_{s,a,r,s'} \leftarrow \boldsymbol{\xi}_{s,a} + \gamma \boldsymbol{\psi}_{\text{SA}}(s', a^*; \boldsymbol{G})$
  $\boldsymbol{G} \leftarrow \boldsymbol{G} + \alpha_{\text{SF}} \left(\boldsymbol{\psi}^{\pi}_{\text{SA}}(s, a; \boldsymbol{G}) - \vec{\boldsymbol{y}}_{s,a,r,s'}\right) \boldsymbol{\xi}^\top_{s,a}$
  $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha_{\text{r}} \left(\boldsymbol{\xi}^\top_{s,a} \boldsymbol{w} - r\right) \boldsymbol{\xi}_{s,a}$

Note that Algorithm 2 need not exactly track Algorithm 1, because the linear SF-Learning algorithm has to also learn the reward weight vector $\boldsymbol{w}$ from transition samples using a particular learning rate. Consequently, the reward condition of Proposition 1 is violated and the result may not apply in this case. However, if linear SF-learning is initialized with a correct $\boldsymbol{w}$ weight vector and both linear Q-Learning and linear SF-learning observe the same transition sequence, then Proposition 1 does apply and both algorithms generate the same value function sequence.

2.2 Transferring Learned Solutions Between Tasks

The previous discussion demonstrates how SFs provide a framework for factorizing the Q-function into a reward weight vector $\boldsymbol{w}$ and a SF vector $\boldsymbol{\psi}^{\pi}_{\text{SA}}$. This property leads to the question if the SF-learning algorithm is suitable for generalization across tasks where only the reward weights $\boldsymbol{w}$ are changed. A change in the reward weight vector $\boldsymbol{w}$ models a change in a task's reward function. While

a change in the reward function may lead to a change in optimal policy and thus re-learning of the SFs, re-using previously learned SFs may still speed-up learning [11, 64]. To test this hypothesis, the SF-learning algorithm was tested on a sequence of grid world navigation tasks shown in Figure 2.1. We consider two scenarios: In Figure 2.1(a) the rewarding goal cell is moved between opposing corners of the grid, resulting in significant reward function change because the optimal action changes at all grid cells. In Figure 2.1(b) the rewarding goal cell is moved diagonally between neighbouring grid cells, resulting in slight reward function change because the optimal action is preserved at most grid cells. SF-learning algorithm was also compared against Q-learning to test if transferring SFs results in faster learning than transferring Q-values. In each experiment, the agent learns how to navigate from the start state to the goal state through repeated trial and error interactions. Once the agent reaches the goal state, the trial, called *episode*, is completed. Each algorithm is allowed to interact with each task for 100 episodes and if a goal state was not reached within 2000 time steps during an episode, the episode was aborted and no reward is given to the agent. In each test case, both algorithms used a state representation ξ which maps each state and action pair to a unique one-hot bit vector. Using such a state representation reduces the linear Q-learning (Algorithm 1) to the usual tabular Q-learning algorithm [115].

Because the conducted experiments test if re-using a previously learned solution can help with initializing learning in another task, two different initialization strategies were tested.

To simulate Q-learning or SF-learning on an MDP, each agent needs to explore the grid to build approximations of Q-values or SFs. We test two different exploration and initialization techniques to determine which technique leads to faster convergence.

One exploration technique is to mix uniform random action selection with greedy action selection. Here, greedy action selection means selecting the action with the highest estimated Q-value. This exploration technique is called ε-greedy, where the probability parameter ε determines that with probability ε actions are chosen greedily and with probability $1 - \varepsilon$ actions are chosen uniformly at random. This ε-greedy exploration strategy is combined with initializing all weight parameters, the Q-values, SF matrix $\boldsymbol{G}$, and reward weights $\boldsymbol{w}$, to zero.

Another exploration technique is to always select actions greedily but initialize all values such that Q-value predictions always over-estimate the expected return. This "optimistic" initialization [53] encourages the agent to seek reward at all states resulting in exploration of all states. As exploration progresses, the agent receives no or negative reward and will in the limit only focus on rewarding

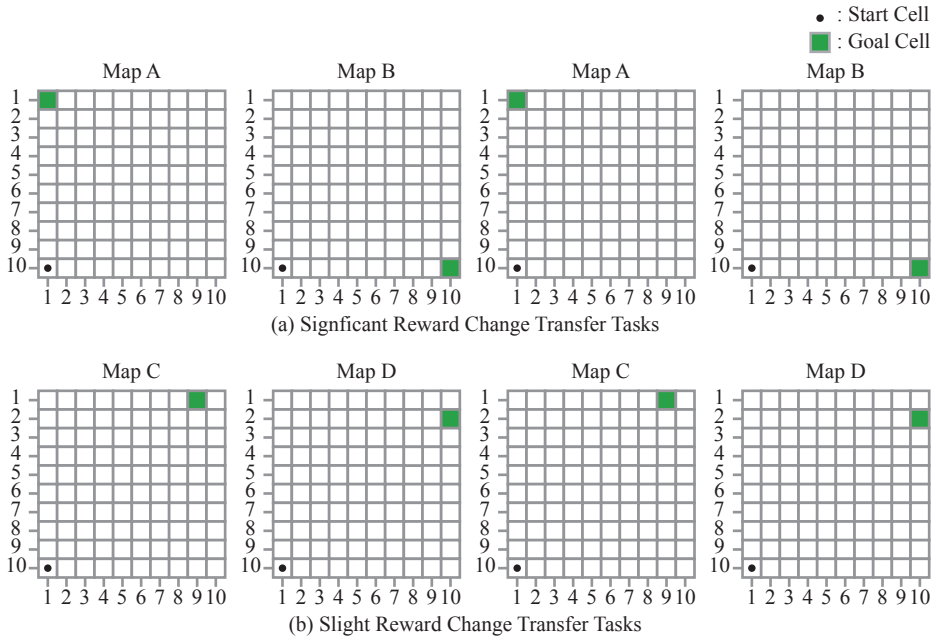


Figure 2.1: Transfer grid world sequences on which the SF-learning and Q-learning agents were tested. At each grid cell, the agent can move up, down, left, or right to move to an adjacent grid cell. Transitions are non-deterministic because with 5% chance selecting an action does not result in a move to the desired direction. 2.1(a): This task sequence simulates significant reward function changes where the goal location moves between two opposing corners of the grid. 2.1(b): This task sequence simulates slight reward function changes where the goal location moves between two diagonally adjacent grid cells.

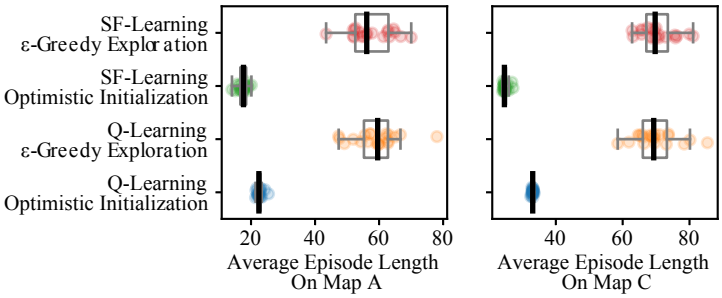


Figure 2.2: Average episode length on the first tasks of the two tested grid world sequences (Map A and Map C in Figure 2.1). Each algorithm was simulated for 200 episodes and the average episode length is plotted for 20 simulation repeats. On both tasks optimistic initialization significantly outperforms ε-greedy exploration. A Welch's t-test between the different exploration strategies in each panel results in p-values that lie below 10^{-14}. A goodness of fit test (Kolmogorov–Smirnov test) of the collected data with a normal distribution resulted in p-values of at least 52% and does not suggest that the data does not follow a normal distribution. Consequently, using the Welch's t-test is appropriate. Tables B.1 and B.2 list the tested and best performing learning.

states and converge to an optimal policy. This exploration strategy can be implemented in SF-learning by setting the matrix $\boldsymbol{G}$ to the identity matrix. If the maximum Q-value in a task is one, for example, setting each entry of the weight vector $\boldsymbol{w}$ to one then results in initializing all Q-values to one, because

$$Q(s,a) = (\boldsymbol{\psi}_{\mathrm{SA}}(s,a;\boldsymbol{G}))^\top \boldsymbol{w} = \boldsymbol{\xi}_{s,a}^\top \boldsymbol{G}^\top \boldsymbol{w} = \boldsymbol{\xi}_{s,a} \boldsymbol{I} \boldsymbol{w} = \boldsymbol{\xi}_{s,a}^\top \boldsymbol{w} = \boldsymbol{e}_i^\top \boldsymbol{w} = 1, \tag{2.19}$$

assuming that $\boldsymbol{\xi}_{s,a} = \boldsymbol{e}_i$ where $\boldsymbol{e}_i$ is a one-hot bit vector.[3]

Figure 2.2 compares the two exploration strategies on the first MDPs of the two tested grid world sequences (Map A and Map C in Figure 2.1). These plots illustrate that optimistic initialization leads to improved exploration in both tasks. The ε-greedy exploration technique explores by selecting actions uniformly at random. Optimistic initialization is more systematic, because after the first time a non-rewarding state is visited, its value is decreased and is then (often) lower than the value of other less explored states. Consequently, non-rewarding states are less likely to be visited multiple times. This property gives optimistic exploration an advantage over ε-greedy exploration in the tested grid world tasks.

Figure 2.3 illustrates transferring SFs across tasks leads to faster learning on the two grid world task sequences. Not transferring SFs or Q-values and re-learning an optimal policy using only the

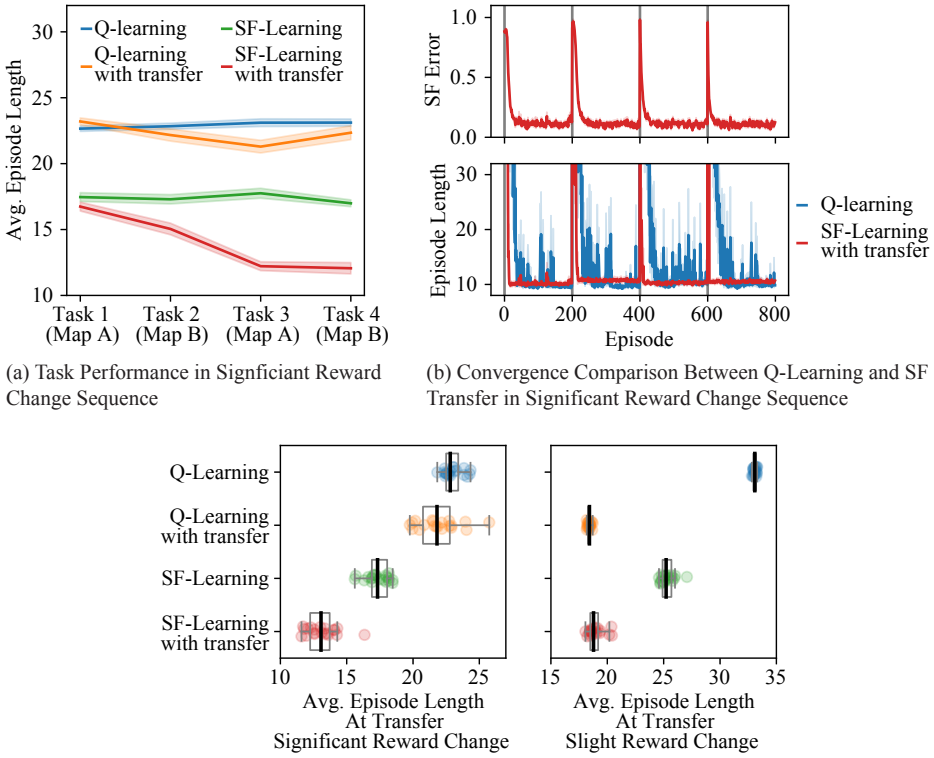
[3] A one-hot bit vector is a vector where the ith entry is set to one and all other entries are zero.

provided initialization leads to constant performance throughout the task curriculum (blue and green curves in Figure 2.3(a)). Transferring previously learned SFs leads to a performance improvement and results in accelerated learning and shorter episode lengths in sub-sequent tasks (magenta curve in Figure 2.3(a)). Yet, the benefit of transferring previously learned Q-values is less significant (orange curve in Figure 2.3(a)). Figure 2.3(b) plots the episode length for each simulated episode. Each agent is simulated for 200 episodes in each task, resulting in a change in reward function every 200 episodes, as indicated by the grey vertical lines. The SF-learning algorithm converges faster than Q-learning to optimal policy, as illustrated in the bottom panel of Figure 2.3(b). Because the changes in rewards leads to a change in optimal policy, SFs, which depend on this optimal policy, have to be re-learned and adjusted to each task. This re-learning is illustrated in the top panel of Figure 2.3(b), where the SF-error spikes up with each reward function change. Nevertheless, transferring SFs leads to a clear performance improvement over not transferring SFs, despite the need of adjusting SFs to each task to obtain an optimal policy. These performance differences are illustrated in Figure 2.3(c), where re-using a previously learned solution leads to significantly faster learning at transfer. The right panel in Figure 2.3(c) suggests a greater performance benefit in the slight reward change tasks sequence. This difference can be attributed to the fact that in the slight reward change tasks sequence the optimal policy changes only in a small number of states when the rewards are changed. Consequently, a previously learned solution can be adapted faster.

2.3 Policy Dependence Limits Transfer Across Tasks

The goal of re-using SFs across tasks is to capture a feature set common to a set of MDPs and this idea seems to perform well for transfer between these MDPs. Interestingly, the results presented in Figure 2.3 indicate that SFs, like Q-values, have to be re-learned when they are re-used across tasks.

Figure 2.4 presents a counter example illustrating when SFs have to be re-learned to find the optimal policy on a different task. In this example, the two MDPs have two actions and deterministic transitions indicated by arrows. Rewards are indicated by the arrow labels and the two MDPs only differ in rewards for two specific transitions. This difference in reward causes the optimal policy for each MDP to be different: The policy π_{aa}, which only selects action a, is optimal in the first MDP; the policy π_{ab}, which selects action b at state s_2 and action a elsewhere, is optimal in the second MDP. The left side of Figure 2.4 shows the SFs for both optimal policies, which is different for



(a) Task Performance in Signficiant Reward Change Sequence

(b) Convergence Comparison Between Q-Learning and SF Transfer in Significant Reward Change Sequence

(c) Performance at Transfer

Figure 2.3: Performance comparison of the tested agents on the significant and slight reward change sequences. 2.3(a): Each curve plots the average episode length across 20 repeats in each task or the significant reward change sequence. The shaded area indicates the standard error of measure. 2.3(b): The bottom panel plots the average episode length for each simulated episode across 20 simulation repeats. The shaded area indicates the standard error of measure. After 200 episodes the reward function was changed (as indicated by the grey lines) and the agent was placed into the next grid world map. When SFs were transferred, the SF-learning algorithms reward-weight vector $\boldsymbol{w}$ was reset to its initialization after each task change. The top panel plots the SF error averaged for each episode. 2.3(c): Average episode length on the transfer tasks two through three for the significant (left panel) and slight (right panel) reward change sequence. A Welch's t-test between the different agent configurations in each panel results in p-values that lie below 1%. Consequently, the performance improvement obtained by transferring SFs is significant. A goodness of fit test (Kolmogorov–Smirnov test) of the collected data with a normal distribution resulted in p-values of at least 50% and does not suggest that the data does not follow a normal distribution. Therefore, using the Welch's t-test is appropriate. In the slight reward change sequence, the Q-learning agent with Q-value transfer outperformed the other agent configurations. Tables B.1 and B.2 list the tested and best performing learning.

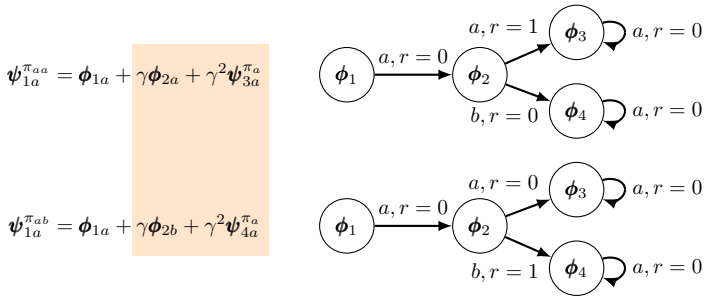


Figure 2.4: Successor Feature Transfer Counter Example. The change in optimal action at state $\boldsymbol{\phi}_2$ causes the SF at state $\boldsymbol{\phi}_1$ to change.

the two MDPs. This difference is caused because SFs are constrained to be similar to features the agent sees in the future. However, which features are seen is governed by the (optimal) policy. This highlights a key limitation of using SFs for transfer: the learned representation is not transferrable between optimal policies. When solving a previously unseen MDP, a learned SF representation can only be used to initialize the search for an optimal policy and the agent still has to adjust the SF representation to the policy that is only optimal in the current MDP.

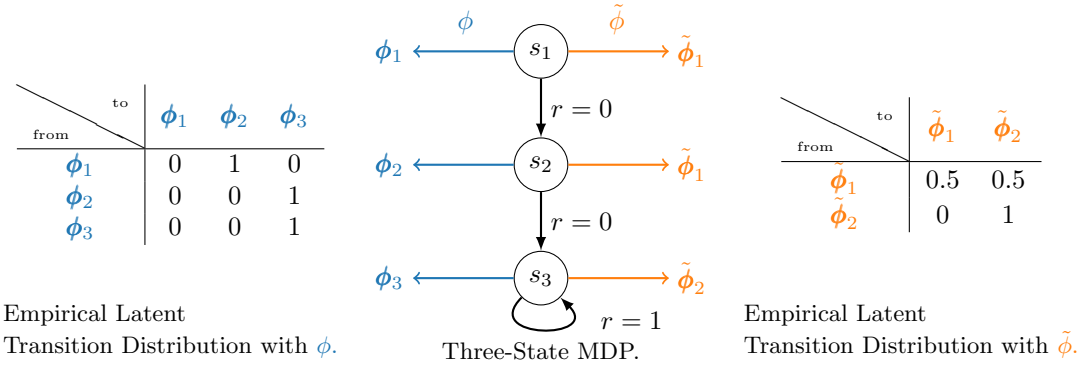
The presented results demonstrate advantages and dis-advantages of transferring SFs between MDPs that only differ in reward function. We argued that learning SFs is a form of TD-learning and under what conditions the SF-learning algorithm generates the same Q-value estimate sequence as Q-learning. While we were able to show a significant performance boost by using this approach, we also highlighted that the learned feature representation is dependent on the policy they are learned for. Hence, SF representations are an unsuitable choice in this context because one is typically interested in transferring knowledge between tasks with different optimal policies. The following chapters present a different SF model that removes these restrictions and learns a state representation that can be preserved across changes in rewards, transitions, and optimal policy.

Chapter 3

Reward-Predictive State Representations

In this chapter, we introduce Linear Successor Feature Models, a model that ties learning successor features to learning reward-predictive representations. Reward-predictive representations construct a latent state space detailed enough for an RL system to predict reward sequences for arbitrary decision sequences. This property of reward-predictive representations ties them to model-based RL, an approach where RL systems use predictions of reward sequences to plan their future decisions. Because LSFMs learn reward-predictive representations through TD-learning, these models provide a novel link between model-based RL and TD-learning. Reward-predictive representations only provide information about how to generalize across inputs and do not contain any specifics about transitions or rewards. Consequently, reward-predictive representations are robust to transfer across tasks where these specifics in transitions and rewards change.

Reward-predictive representations are constructed such that the empirical transition probabilities between latent states mimic transitions in the original task. Figure 3.1 presents a reward-prediction example where only one action is available to the agent. In this task, the goal is to predict that a positive reward is obtained in three time steps if the agent starts at state s_1. This example compares two different state representations, the representation ϕ, which does not compress the state space, and $\tilde{\phi}$, which merges the first two states into one latent state. These two state representations lead to different empirical latent transition probabilities. While the first representation preserves the deterministic transitions of the task, the second representation does not. If states s_1 and s_2 are mapped to the same latent state $\tilde{\boldsymbol{\phi}}_1$, then a transition from state s_1 to s_2 appears as a self-loop from latent state $\tilde{\boldsymbol{\phi}}_1$ to itself and a transition from s_2 to s_3 appears as a transition from $\tilde{\boldsymbol{\phi}}_1$ to $\tilde{\boldsymbol{\phi}}_2$. Because the state representation ϕ constructs a latent state space with empirical latent transition



Empirical Latent Transition Distribution with ϕ.

Three-State MDP.

Empirical Latent Transition Distribution with $\tilde{\phi}$.

Figure 3.1: Three-State MDP Example. The centre schematic shows a single action three-state MDP with deterministic transitions (black arrows). Only the self-looping transition at state s_3 is rewarded. The two state representations ϕ and $\tilde{\phi}$ map the three states to different feature vectors, resulting in different empirical feature-transition probabilities. These probabilities are computed from observed trajectories that start at state s_1.

probabilities that match the transition probabilities of the original task, this state representation is reward predictive.

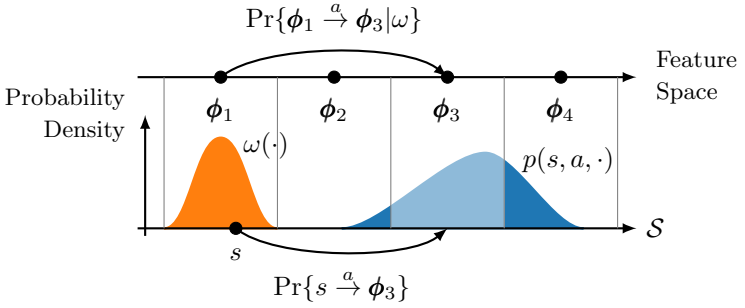
A reward-predictive state representation can be used in conjunction with a Linear Action Model [108, 121] to compute expected future reward outcomes.

Definition 2 (Linear Action Model (LAM))**.** *Given an MDP M and a state representation $\phi : \mathcal{S} \to \mathbb{R}^n$, a LAM consists of a set of matrices and vectors $\{\boldsymbol{M}_a, \boldsymbol{w}_a\}_{a \in \mathcal{A}}$, where $\boldsymbol{M}_a$ is of size $n \times n$ and the column vector $\boldsymbol{w}_a$ is of dimension n.*

Given a fixed state representation, the transition matrices of a LAM $\{\boldsymbol{M}_a\}_{a \in \mathcal{A}}$ model the empirical latent transition probabilities and the vectors $\{\boldsymbol{w}_a\}_{a \in \mathcal{A}}$ model a linear map from latent states to expected one-step reward outcomes. The expected reward outcome after following the action sequence $a_1, ..., a_t$ starting at state s can then be approximated with

$$\mathbb{E}_p \left[R_t | s, a_1, ..., a_t \right] \approx \boldsymbol{\phi}_s^\top \boldsymbol{M}_{a_1} \cdots \boldsymbol{M}_{a_{t-1}} \boldsymbol{w}_{a_t}. \tag{3.1}$$

The following sections will address how a state representation ϕ and a LAM can be found to predict expected future reward outcomes as accurately as possible. Because this chapter's goal is to establish different connections between learning successor features and model-based RL and to demonstrate that the learned reward-predictive state representations are suitable for transfer across variations in transitions and rewards, an extension of these model to non-linear latent transition and reward



$\Pr\{s \xrightarrow{a} \boldsymbol{\phi}_3\}$: The probability of transitioning from s to any state mapped to $\boldsymbol{\phi}_3$ by selecting action a.

$\Pr\{\boldsymbol{\phi}_1 \xrightarrow{a} \boldsymbol{\phi}_3|\omega\}$: Empirical probability of transitioning from latent state $\boldsymbol{\phi}_1$ to latent state $\boldsymbol{\phi}_3$ by selecting action a.

Figure 3.2: Empirical Latent Transition Probabilities Depend on State-Visitation Frequencies. This example illustrates how empirical latent transition probabilities depend on state-visitation frequencies. In this example, the state space $\mathcal{S}$ is a bounded interval in $\mathbb{R}$ that is clustered into one of four latent states: $\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \boldsymbol{\phi}_3$, or $\boldsymbol{\phi}_4$. State-visitation frequencies are modelled for each partition independently using the density function ω. The schematic plots the density function p over states of selecting action a at state s (blue area) and the density function ω over the state partition $\boldsymbol{\phi}_1$ (orange area). The probability $\Pr\{s \xrightarrow{a} \boldsymbol{\phi}_3\}$ of transitioning into the partition $\boldsymbol{\phi}_3$ is the blue shaded area. The probability $\Pr\{\boldsymbol{\phi}_1 \xrightarrow{a} \boldsymbol{\phi}_3|\omega\}$ of a transition from $\boldsymbol{\phi}_1$ to $\boldsymbol{\phi}_3$ occurring is the marginal of $\Pr\{s \xrightarrow{a} \boldsymbol{\phi}_3\}$ over all states s mapping to $\boldsymbol{\phi}_1$, weighted by ω.

functions is left to future work.

To tie SFs to reward-predictive state representations, we first introduce a set of square real-valued matrices $\{\boldsymbol{F}_a\}_{a\in\mathcal{A}}$ such that, for every state s and action a,

$$\boldsymbol{\phi}_s^\top \boldsymbol{F}_a \approx \boldsymbol{\psi}^\pi(s,a) \tag{3.2}$$

$$= \mathbb{E}_{p,\pi}\left[\sum_{t=1}^{\infty} \gamma^{t-1} \boldsymbol{\phi}_{s_t} \middle| s_1 = s, a_1 = a\right] \tag{3.3}$$

where the policy π is defined on the latent state space. A Linear Successor Feature Model (LSFM) is then defined using the matrices $\{\boldsymbol{F}_a\}_{a\in\mathcal{A}}$:

Definition 3 (Linear Successor Feature Model (LSFM))**.** *Given an MDP, a policy π, and a state representation $\phi : \mathcal{S} \to \mathbb{R}^n$, an LSFM consists of a set of matrices and vectors $\{\boldsymbol{F}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$, where $\boldsymbol{F}_a$ is of size $n \times n$ and the column vector $\boldsymbol{w}_a$ is of dimension n. The matrices $\{\boldsymbol{F}_a\}_{a\in\mathcal{A}}$ are used to model a linear map from latent state features to SFs as described in Equation* (3.2)*.*

LSFMs require the state representation ϕ to (linearly) approximate the SF $\boldsymbol{\psi}^\pi(s,a)$ using the matrices $\{\boldsymbol{F}_a\}_{a\in\mathcal{A}}$ (Equation (3.2)). Previously presented SF frameworks [11, 12] do not use the state representation ϕ to approximate SF vectors $\boldsymbol{\psi}^\pi$ as described in Equation (3.2) and construct the state-representation function ϕ differently, for example, by setting the output of ϕ to be one-step

rewards [11]. In contrast, LSFMs are used to learn the state-representation function ϕ that satisfies Equation (3.2). Because LSFMs distinctly incorporate this approximative property, SFs can be connected to model-based RL.

An intelligent agent that uses a state representation ϕ operates directly on the constructed latent state space and is constrained to only search the space of policies that are defined on its latent state space. These policies are called *abstract policies*.

Definition 4 (Abstract Policies)**.** *An abstract policy π_ϕ is a function mapping latent state and action pairs to probability values:*

$$\forall s \in \mathcal{S}, a \in \mathcal{A}, \pi_\phi(\boldsymbol{\phi}_s, a) \in [0,1] \text{ and } \sum_a \pi_\phi(\boldsymbol{\phi}_s, a) = 1.$$

For a fixed state representation ϕ, the set of all abstract policies is denoted with Π_ϕ.

The following sections first tie learning LAMs to reward-predictive state representations. We then show that learning LSFMs is equivalent to learning LAMs tying LSFMs to reward-predictive state representations.

3.1 Encoding Bisimulation Relations

To ensure accurate predictions of future reward outcomes, the previous discussion suggests that the empirical latent transition probabilities have to match the transition probabilities in the original task. Figure 3.2 presents a schematic explaining these dependencies further. In this example, the state space is a bounded interval in $\mathbb{R}$ that is mapped to four different latent states, $\boldsymbol{\phi}_1$, $\boldsymbol{\phi}_2$, $\boldsymbol{\phi}_3$, or $\boldsymbol{\phi}_4$. The probability of transitioning from the state s to any state that is mapped to $\boldsymbol{\phi}_3$ is denoted with $\Pr\{s \xrightarrow{a} \boldsymbol{\phi}_3\}$. This probability $\Pr\{s \xrightarrow{a} \boldsymbol{\phi}_3\}$ is the marginal over all states s' that are mapped to the latent state $\boldsymbol{\phi}_3$. Assume that ω is a density function over all states that are mapped to the latent state $\boldsymbol{\phi}_1$. This density function could model the visitation frequencies of different states as an intelligent agent interacts with the MDP. The empirical probability of transitioning from latent state $\boldsymbol{\phi}_1$ to $\boldsymbol{\phi}_3$ is then the marginal over all states mapping to $\boldsymbol{\phi}_1$ and

$$\Pr\left\{\boldsymbol{\phi}_1 \xrightarrow{a} \boldsymbol{\phi}_3 \middle| \omega\right\} = \int_{s:\phi(s)=\boldsymbol{\phi}_1} \omega(s) \Pr\{s \xrightarrow{a} \boldsymbol{\phi}_3\} ds = \mathbb{E}_\omega \left[\Pr\{s \xrightarrow{a} \boldsymbol{\phi}_3\} \middle| \phi(s) = \boldsymbol{\phi}_1\right]. \tag{3.4}$$

The expectation in Equation (3.4) is computed with respect to ω over all states s that map to the latent state $\boldsymbol{\phi}_1$. As Equation (3.4) outlines, the empirical transition probability $\Pr\{\boldsymbol{\phi}_1 \xrightarrow{a} \boldsymbol{\phi}_3|\omega\}$ depends on the visitation frequencies ω. The probability $\Pr\{s \xrightarrow{a} \boldsymbol{\phi}_3\}$ of transitioning from a state s into a partition only depends on the transition function p itself.

Consider two different states s and $\tilde{s}$ that map to the same latent state and $\phi(s) = \phi(\tilde{s})$. If the state representation is constructed such that

$$\forall a, \forall \boldsymbol{\phi}_i,\ \Pr\{s \xrightarrow{a} \boldsymbol{\phi}_i\} = \Pr\{\tilde{s} \xrightarrow{a} \boldsymbol{\phi}_i\} \text{ and } \mathbb{E}_p\left[r(s,a,s')|s,a\right] = \mathbb{E}_p\left[r(\tilde{s},a,s')|\tilde{s},a\right], \tag{3.5}$$

then the empirical latent state transition probabilities would become independent of ω because the integrand in Equation (3.4) is constant and

$$\Pr\left\{\boldsymbol{\phi}_1 \xrightarrow{a} \boldsymbol{\phi}_3 \middle| \omega\right\} = \int_{s:\phi(s)=\boldsymbol{\phi}_1} \omega(s) \underbrace{\Pr\{s \xrightarrow{a} \boldsymbol{\phi}_3\}}_{\text{constant}} ds = \Pr\{s \xrightarrow{a} \boldsymbol{\phi}_3\}. \tag{3.6}$$

Equation (3.6) follows directly from the transition condition in line (3.5), because the probability $\Pr\{s \xrightarrow{a} \boldsymbol{\phi}_3\}$ is constant for all states s that are mapped to the latent state vector $\boldsymbol{\phi}_1$. If the two identities in line (3.5) hold, then the resulting latent state space constructs latent transition probabilities that correspond to the transition probabilities in the original task. Equation (3.5) describes an informal definition of bisimulation [46]. Definition 5 listed in Appendix A.2 presents a formal measure theoretic definition of bisimulation on arbitrary (measurable) state spaces. This definition is used to prove the theorems stated in this section. To prove that LAMs encode state representations that generalize only across bisimilar states, two assumptions are made.

Assumption 1. *The state space $\mathcal{S}$ of an MDP can be partitioned into at most n different partitions of bisimilar states, where n is a natural number.*

Assumption 2. *A state representation $\phi : \mathcal{S} \to \{\boldsymbol{e}_1, ..., \boldsymbol{e}_n\}$ is assumed to have a range that consists of all n one-hot bit vectors. For each i, there exists a state s such that $\phi(s) = \boldsymbol{e}_i$.*

Assumption 1 is not particularly restrictive in a learning context: If an agent has observed n distinct states during training, then a state representation assigning each state to one of n different one-hot bit vectors can always be constructed. While doing so may not be useful to generalize across different states, this argument suggests that Assumption 1 is not restrictive in practice. Assumption 2 is relaxed in the following sections.

If action a is selected at state s, the expected next feature vector is

$$\mathbb{E}_p\left[\phi(s')|s,a\right] = \sum_{j=1}^{n} \Pr\{s \xrightarrow{a} \boldsymbol{e}_j\}\boldsymbol{e}_j = \left[\cdots, \Pr\{s \xrightarrow{a} \boldsymbol{e}_j\}, \cdots\right]^\top. \tag{3.7}$$

The expected value in Equation (3.7) is computed over all possible next states s' that can be reached from state s by selecting action a. In Equation (3.7), the next state s' is a random variable whose probability distribution or density function is described the by the MDP's transition function p. By Assumption 2, each state is mapped to some one-hot bit vector $\boldsymbol{e}_j$. Because there are only n different one-hot bit vectors of dimension n, the summation in Equation (3.7) is finite. Each entry of the resulting vector in Equation (3.7) stores the probability $\Pr\{s \xrightarrow{a} \boldsymbol{e}_j\}$ of observing the feature vector $\boldsymbol{e}_j$ after selecting action a at state s.

Because the expected next feature vector $\mathbb{E}_p\left[\phi(s')|s,a\right]$ is a probability vector, the transition matrices $\{\boldsymbol{M}_a\}_{a\in\mathcal{A}}$ of a LAM are stochastic: If $\phi(s) = \boldsymbol{e}_i$ and $\boldsymbol{e}_i^\top \boldsymbol{M}_a = \mathbb{E}_p\left[\boldsymbol{e}_j|s,a\right]$, then the ith row of the matrix $\boldsymbol{M}_a$ is equal to the probability vector shown in Equation (3.7). If $\boldsymbol{e}_i^\top \boldsymbol{w}_a = \mathbb{E}_p\left[r(s,a,s')|s,a\right]$, then the weight vectors of a LAM $\{\boldsymbol{w}_a\}_{a\in\mathcal{A}}$ encode a reward table. These observations lead to the first theorem.[1]

Theorem 1. *For an MDP $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma\rangle$, let $\phi : \mathcal{S} \to \{\boldsymbol{e}_1, ..., \boldsymbol{e}_n\}$ be a state representation and $\{\boldsymbol{M}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$ a LAM. Assume that $\mathcal{S}$ can be partitioned into at most n partitions of bisimilar states. If the state representation ϕ satisfies*

$$\forall s \in \mathcal{S}, \forall a \in \mathcal{A},\ \boldsymbol{\phi}_s^\top \boldsymbol{w}_a = \mathbb{E}_p\left[r(s,a,s')|s,a\right] \ \text{ and } \ \boldsymbol{\phi}_s^\top \boldsymbol{M}_a = \mathbb{E}_p\left[\boldsymbol{\phi}_{s'}|s,a\right], \tag{3.8}$$

then ϕ generalizes across bisimilar states and any two states s and $\tilde{s}$ are bisimilar if $\boldsymbol{\phi}_s = \boldsymbol{\phi}_{\tilde{s}}$.

The proof of Theorem 1 uses the fact that the expected value of one-hot bit vectors encode exact probability values. A similar observation can be made about the SFs for a one-hot bit-vector state representation. In this case, the $(1-\gamma)$ rescaled SF contains the marginal of reaching a state partition across time steps:

$$(1-\gamma)\mathbb{E}_{p,\pi}\left[\sum_{t=1}^{\infty} \gamma^{t-1}\boldsymbol{e}_t \middle| s, a_1\right] = \left[..., \sum_{t=1}^{\infty}(1-\gamma)\gamma^{t-1}\mathbb{E}_{p,\pi}\left[\Pr\left\{s \xrightarrow{a_1\cdots a_t} \boldsymbol{e}_i\right\}\middle| s, a_1\right], ...\right]^\top, \tag{3.9}$$

[1] Appendix A.2 presents formal proofs for all presented theorems.

where the expectation in Equation (3.9) is computed over infinite length trajectories starting at state s with action a. This observation leads to the following theorem stating that LSFMs can be used to identify a one-hot state representation that generalizes across bisimilar states.

Theorem 2. *For an MDP $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$, let $\phi : \mathcal{S} \to \{\boldsymbol{e}_1, ..., \boldsymbol{e}_n\}$ be a state representation and $\{\boldsymbol{F}_a, \boldsymbol{w}_a\}_{a \in \mathcal{A}}$ an LSFM. If, for one policy $\pi \in \Pi_\phi$, the representation ϕ satisfies*

$$\forall s \in \mathcal{S}, \forall a \in \mathcal{A},\ \boldsymbol{\phi}_s^\top \boldsymbol{w}_a = \mathbb{E}_p \left[r(s, a, s') | s, a \right] \ \text{ and } \boldsymbol{\phi}_s^\top \boldsymbol{F}_a = \boldsymbol{\phi}_s^\top + \gamma \mathbb{E}_{p,\pi} \left[\boldsymbol{\phi}_{s'} \boldsymbol{F}_{a'} | s, a \right], \tag{3.10}$$

then ϕ generalizes across bisimilar states and any two states s and $\tilde{s}$ are bisimilar if $\boldsymbol{\phi}_s = \boldsymbol{\phi}_{\tilde{s}}$. If Equation (3.10) holds for one policy $\pi \in \Pi_\phi$, then Equation (3.10) also holds every other policy $\tilde{\pi} \in \Pi_\phi$ as well.

Equation (3.10) describes a fixed-point equation similar to the Bellman fixed-point equation:

$$\begin{aligned}
\boldsymbol{e}_s^\top \boldsymbol{F}_a &= \left(\boldsymbol{\psi}^\pi(s, a) \right)^\top && (3.11) \\
&= \mathbb{E}_{p,\pi} \left[\sum_{t=1}^{\infty} \gamma^{t-1} \boldsymbol{e}_{s_t}^\top \middle| s, a \right] && (3.12) \\
&= \boldsymbol{e}_s^\top + \gamma \mathbb{E}_p \left[\mathbb{E}_{p,\pi} \left[\sum_{t=1}^{\infty} \gamma^{t-1} \boldsymbol{e}_{s_t}^\top \middle| s', a' \right] \middle| s, a \right] && (3.13) \\
&= \boldsymbol{e}_s^\top + \gamma \mathbb{E}_p \left[\boldsymbol{e}_{s'}^\top \boldsymbol{F}_{a'} \middle| s, a \right]. && (3.14)
\end{aligned}$$

Finding a policy $\pi \in \Pi_\phi$ to test if Equation (3.10) holds for a state representation ϕ is trivial, because it is sufficient to test the state representation for any single policy. Theorems 1 and 2 show that LAMs and LSFMs can be used to identify one-hot reward-predictive state representations. To arrive at an algorithm that can learn reward-predictive state representations, the following sections convert the conditions outlined in Theorems 1 and 2 into learning objectives. The next section presents an analysis showing how violating these conditions by some margin results in increased reward-sequence prediction errors. We refer to state representations that can only approximately predict expected reward sequences as an *approximate reward-predictive state representation.*

3.2 Approximate Reward-Predictive Representations

In this section, we analyze to what extent a state representation is reward predictive if it only approximately satisfies the conditions outlined in Theorems 1 and 2. In addition, we will also generalize beyond one-hot representations and relax Assumption 2 by considering state representations that map the state space into $\mathbb{R}^n$. The latent feature's dimension n is considered a fixed hyper-parameter.

Because LAMs only model one-step transitions but are used to predict entire reward sequences, the scale and expansion properties of the constructed latent state space influences how prediction errors scale and compound [110, 6]. Define the following variables:[2]

$$W = \max_{a \in \mathcal{A}} ||\boldsymbol{w}_a||, \; M = \max_{a \in \mathcal{A}} ||\boldsymbol{M}_a||, \; N = \sup_{s \in \mathcal{S}} ||\boldsymbol{\phi}_s||. \tag{3.15}$$

To identify approximate reward-predictive state representations, a state representation ϕ is analyzed by its one-step reward-prediction error and one-step expected transition error. These quantities are computed using a LAM $\{\boldsymbol{M}_a, \boldsymbol{w}_a\}_{a \in \mathcal{A}}$ and are defined as

$$\varepsilon_r = \sup_{s,a} \left| r(s,a) - \boldsymbol{\phi}_s^\top \boldsymbol{w}_a \right| \text{ and} \tag{3.16}$$

$$\varepsilon_p = \sup_{s,a} \left|\left| \mathbb{E}_p \left[\boldsymbol{\phi}_{s'}^\top \middle| s, a \right] - \boldsymbol{\phi}_s^\top \boldsymbol{M}_a \right|\right|. \tag{3.17}$$

Equivalently, a state representation is also analyzed using an LSFM that predicts the SF for a policy that selects actions uniformly at random. For an LSFM $\{\boldsymbol{F}_a, \boldsymbol{w}_a\}_{a \in \mathcal{A}}$, define

$$\overline{\boldsymbol{F}} = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \boldsymbol{F}_a. \tag{3.18}$$

For such an LSFM, the linear SF prediction error is defined as

$$\varepsilon_\psi = \sup_{s,a} \left|\left| \boldsymbol{\phi}_s^\top + \gamma \mathbb{E}_p \left[\boldsymbol{\phi}_{s'}^\top \overline{\boldsymbol{F}} \middle| s, a \right] - \boldsymbol{\phi}_s^\top \boldsymbol{F}_a \right|\right|. \tag{3.19}$$

Because the matrix $\overline{\boldsymbol{F}}$ averages across all actions, the LSFM computes SFs for a policy that selects actions uniformly at random. Here, we focus on uniform random action selection to simplify the analysis. While the matrix $\overline{\boldsymbol{F}}$ could be constructed differently, the proofs of the theoretical results

[2] All norms are assumed to be L_2. The Euclidean norm is used for vectors. The norm of a matrix $\boldsymbol{M}$ is computed with $||\boldsymbol{M}|| = \sqrt{\sum_{i,j} \boldsymbol{M}(i,j)^2}$, where the summation ranges over all matrix entries $\boldsymbol{M}(i,j)$.

presented in this section assume that $\overline{\boldsymbol{F}}$ can only depend on the matrices $\{\boldsymbol{F}_a\}_{a\in\mathcal{A}}$ and is not a function of the state s.

Similar to the previous discussion, LSFMs are closely related to LAMs and the one-step transition error ε_p can be upper bounded by the linear SF error ε_ψ. We define the quantities ε_r, ε_t, and ε_ψ to upper bound the magnitude with which the identities provided in Theorems 1 and 2 are violated. The following analysis generalizes the previously presented results by showing that, if a state representation approximately satisfies the requirements of Theorems 1 and 2, then this state representation is approximately reward predictive.

Lemma 1. *For an MDP, a state representation ϕ, an LSFM and a LAM,*

$$\varepsilon_p \leq \varepsilon_\psi \frac{1+\gamma M}{\gamma} + C_{\gamma,M,N}\Delta, \tag{3.20}$$

where $C_{\gamma,M,N} = (1+\gamma)(1+\gamma M)N/(\gamma(1-\gamma M))$ *and* $\Delta = \max_a ||\boldsymbol{I} + \gamma \boldsymbol{M}_a \overline{\boldsymbol{F}} - \boldsymbol{F}_a||$.

Lemma 1 presents a bound stating that if an LSFM has low linear SF prediction errors, then a corresponding LAM can be constructed with low one-step transition error ε_p, assuming that Δ is close to zero. In Chapter 2, Equation (2.2), the action-conditional SR matrix is defined as $\boldsymbol{\Psi}_a^\pi = \boldsymbol{I} + \gamma \boldsymbol{P}_a \boldsymbol{\Psi}^\pi$, where $\boldsymbol{P}_a$ is a stochastic transition matrix for a finite MDP. If $\Delta = 0$, then the matrices $\{\boldsymbol{F}_a\}_{a\in\mathcal{A}}$ can be thought of as action-conditional SR matrices for the transition matrices $\{\boldsymbol{M}_a\}_{a\in\mathcal{A}}$ and

$$\boldsymbol{F}_a = \boldsymbol{I} + \gamma \boldsymbol{M}_a \overline{\boldsymbol{F}}. \tag{3.21}$$

In Chapter 2 we shows that there exists a bijection between the transition matrices $\{\boldsymbol{P}_a\}_{a\in\mathcal{A}}$ and the action-conditional SR matrices $\{\boldsymbol{\Psi}_a^\pi\}_{a\in\mathcal{A}}$. The proof of Lemma 1 repeats the same argument for Equation (3.21) and shows that there exists a bijection between the matrices $\{\boldsymbol{F}_a\}_{a\in\mathcal{A}}$ and $\{\boldsymbol{M}_a\}_{a\in\mathcal{A}}$. For arbitrary state representations and LSFMs, Equation (3.21) may not hold if $\Delta > 0$.

The following theorem presents a bound stating that low one-step reward and one-step transition errors lead to state representations that support accurate predictions of future expected reward outcomes. By Lemma 1, the following results also apply to LSFMs because low linear SF prediction errors lead to low one-step expected transition errors.

Theorem 3. *For an MDP, state representation $\phi \to \mathbb{R}^n$, and for all $T \geq 1, s, a_1, ..., a_T$,*

$$\left|\boldsymbol{\phi}_s^\top \boldsymbol{M}_{a_1} \cdots \boldsymbol{M}_{a_{T-1}} \boldsymbol{w}_{a_T} - \mathbb{E}_p\left[r_T | s, a_1, ..., a_T\right]\right| \leq \varepsilon_p \sum_{t=1}^{T-1} M^t W + \varepsilon_r. \tag{3.22}$$

Theorem 3 shows that prediction errors of expected rollouts are bounded linearly in ε_r and ε_p and prediction errors tend to zero as ε_r and ε_p tend to zero. Because LAMs are one-step models, prediction errors increase linearly with T if $M \leq 1$ as the model is used to generalize over multiple time steps. Prediction errors may increase exponentially if the transition matrices are expansions and $M > 1$, similar to previously presented bounds [6].

The following theorem bounds the prediction error of finding a linear approximation of the Q-function $Q^\pi(s, a) \approx \boldsymbol{\phi}_s^\top \boldsymbol{q}_a$ using a state representation ϕ and a real valued vector $\boldsymbol{q}_a$.

Theorem 4. *For an MDP, state representation $\phi : \mathcal{S} \to \mathbb{R}^n$, any arbitrary abstract policy $\pi \in \Pi_\phi$, and LAM $\{\boldsymbol{M}_a, \boldsymbol{w}_a\}_{a \in \mathcal{A}}$, there exists vectors $\boldsymbol{v}^\pi$ and $\{\boldsymbol{q}_a = \boldsymbol{w}_a + \gamma \boldsymbol{M}_a \boldsymbol{v}^\pi\}_{a \in \mathcal{A}}$ such that, for all states s and actions a,*

$$\left|V^\pi(s) - \boldsymbol{\phi}_s^\top \boldsymbol{v}^\pi\right| \leq \frac{\varepsilon_r + \gamma \varepsilon_p \left|\left|\boldsymbol{v}^\pi\right|\right|}{1 - \gamma} \text{ and } \left|Q^\pi(s, a) - \boldsymbol{\phi}_s^\top \boldsymbol{q}_a\right| \leq \frac{\varepsilon_r + \gamma \varepsilon_p \left|\left|\boldsymbol{v}^\pi\right|\right|}{1 - \gamma}. \tag{3.23}$$

By Theorem 4, an (approximate) reward-predictive state representation (approximately) generalizes across all abstract policies, because the same state representation can be used to predict the value of every possible abstract policy $\pi \in \Pi_\phi$. Prediction errors tend to zero as ε_r and ε_p tend to zero. The value prediction error bounds stated in Theorem 4 are similar to bounds presented by [19] on approximate (linear) policy iteration, because the presented results also approximate value functions using a function that is linear in some basis function ϕ. Conforming to these previously presented results on linear value function approximation, prediction errors scale linearly in one-step prediction errors ε_ψ and ε_p. Theorems 4 and 3 show that, by learning a state representation that predicts SFs for policies that select actions uniformly at random, an approximate reward-predictive state representation is obtained. This state representation generalizes across the entire space of abstract policies, because accurate predictions of each policy's value function are possible if prediction errors are low enough. Appendix A.2.1 presents formal proofs of Theorems 3 and 4.

Figure 3.3 presents an example highlighting that reward-predictive state representations do not necessarily encode bisimulation relations. In this example, states A and B are not bisimilar, because

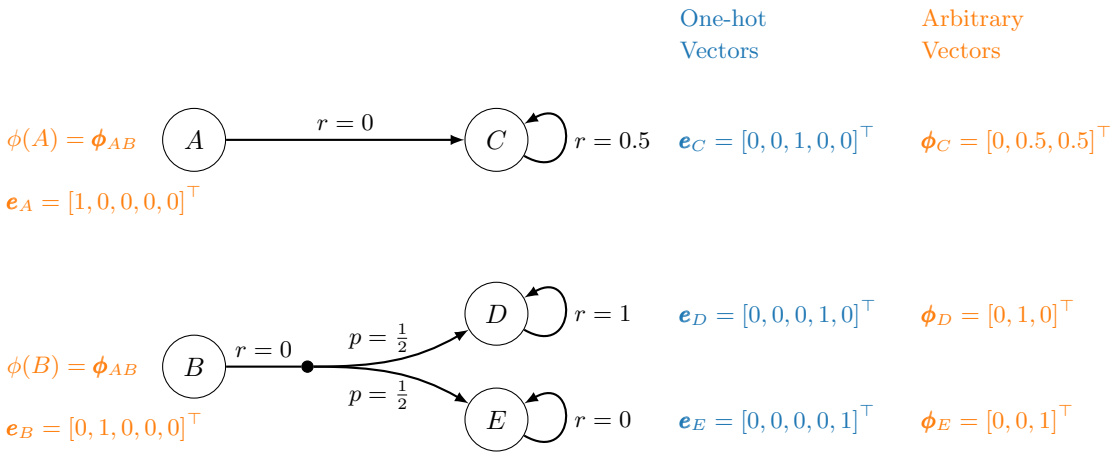
the probabilities with which they transition to C, D, or E are different. The state representation ϕ generalizes across these two states and $\varepsilon_r = \varepsilon_p = \varepsilon_\psi = 0$. The expected reward sequence for transitioning out of A or B is always $0, 0.5, 0.5, ...$, so both states have equal expected future reward outcomes and the state representation ϕ is reward predictive. However, the state representation is not predictive of the probability with which a particular reward sequence is observed. For example, the latent state space constructed by ϕ would have to make a distinction between state A and B to support predictions stating that a reward sequence of $0, 1, 1, ...$ can be obtained from state B with probability 0.5. The example in Figure 3.3 highlights the difference between the analysis presented in this section and the previous section: By relaxing Assumption 2, one may still obtain a reward-predictive state representation, but this representation may not necessarily encode a bisimulation relation.

Figure 3.4 illustrates on an example that reward-predictive representation generalize across different states differently than the value-predictive state representations discussed in Chapter 2. In this example it is not always possible to construct an optimal policy using a value-predictive state representation. If a sub-optimal policy is used, value-predictive state representations may alias states that have different optimal actions prohibiting an intelligent agent from finding an optimal policy if such a value-predictive state representation is used. In contrast, reward-predictive state representations generalize across the entire policy space (Theorem 4) and allow an agent to find either an optimal policy or close to optimal policy in the presence of approximation errors.

Note that an (approximate) reward-predictive state representation $\phi : \mathcal{S} \to \mathbb{R}^n$ could in principle map each state to a distinct latent state vector. As demonstrated by the following simulations, the idea behind generalizing across states is that the dimension n of the constructed latent space is sufficiently small to constrain the LSFM learning algorithm and assign approximately the same latent state vector to different states. The following section illustrates how (approximate) reward-predictive state representations model generalization across different states.

3.3 Learning Reward-Predictive Representations

Using the previously presented theoretical results, this section designs a loss objective to learn approximate reward-predictive state representations. By optimizing a loss function, a randomly chosen state-representation function $\phi : \mathcal{S} \to \mathbb{R}^n$ is iteratively improved until the function ϕ can be



(a) Reward-predictive representations can be encoded using different state features.

Model	Prediction Target	with one-hot	with arbitrary
LAM	$\mathbb{E}_p[\boldsymbol{\phi}\|A]$	$= [0, 0, 1, 0, 0]^\top$	$= [0, 0.5, 0.5]^\top$
	$\mathbb{E}_p[\boldsymbol{\phi}\|B]$	$= [0, 0, 0, 0.5, 0.5]^\top$	$= [0, 0.5, 0.5]^\top$
LSFM	$\mathbb{E}_p[\sum_{t=1}^{\infty} \gamma^{t-1} \boldsymbol{\phi}_t \| A]$	$= [1, 0, 9, 0, 0]^\top$	$= \boldsymbol{\phi}_{AB} + [0, 3.5, 3.5]^\top$
	$\mathbb{E}_p[\sum_{t=1}^{\infty} \gamma^{t-1} \boldsymbol{\phi}_t \| B]$	$= [0, 1, 0, 3.5, 3.5]^\top$	$= \boldsymbol{\phi}_{AB} + [0, 3.5, 3.5]^\top$

(b) Prediction targets of a LAM and LSFM for both state representations.

Figure 3.3: Real-valued reward-predictive state representations may not encode bisimulations, but support predictions of future expected reward outcomes. 3.3(a): In this five-state, example no states are bisimilar. Each edge is labelled with the reward given to the agent for a particular transition. The transition departing state B is probabilistic and leads to state D or E with equal probability. All other transitions are deterministic. Two different state representations are considered. One representation maps states to one-hot bit vectors and the other representation maps states to real-valued vectors. 3.3(b): Prediction targets for both LAM and LSFM depend on what state representation is used. For a one-hot state representation, the LAM and LSFM have different prediction targets for states A and B, because a one-hot bit-vector state representation can be used to detect that transition probabilities are different between A and B. In contrast, real valued state representations may lead to equal prediction targets for both LAM and LSFM, because the state features $\boldsymbol{\phi}_C$, $\boldsymbol{\phi}_D$, and $\boldsymbol{\phi}_E$ can hide different transition probabilities. The state representation ϕ is reward predictive and $\varepsilon_r = \varepsilon_p = \varepsilon_\psi = 0$.

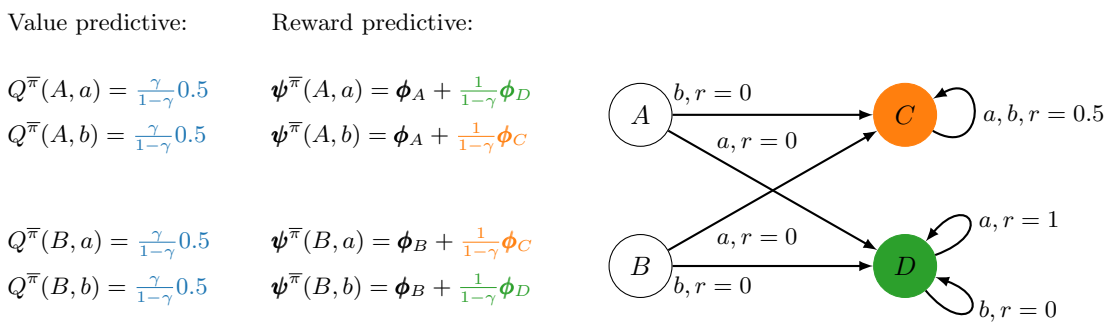


Figure 3.4: Value-predictive state representations may prohibit an agent from learning an optimal policy. In this MDP, the agent can choose between action a and action b. All transitions are deterministic and each edge is labelled with the reward given to the agent. If a uniform-random action-selection policy is used to construct a value-predictive state representation, then both states A and B will have equal Q-values. A reward-predictive state representation would always distinguish between A and B, because at state A the action sequence $b, a, a...$ leads to a reward sequence of $0, 0.5, 0.5, ...$ while at state B the action sequence $b, a, a, ...$ leads to a reward sequence of $0, 1, 1, ...$ An LSFM detects that states A and B should not be merged into the same latent state, because the states have different SFs. The optimal policy is to select action a at state A, and action b at state B and then collect a reward of one at state D by repeating action a. If an agent uses a reward-predictive state representation, then the optimal policy could be recovered. If an agent uses a value-predictive state representation, the agent would be constrained to not distinguish between states A and B and cannot recover an optimal policy.

used to accurately predict reward sequences. The cluster plots in Figure 3.5 illustrate this process: Starting with a random assignment of feature vectors to different grid states, a state representation is iteratively improved until all states of the same column are clustered approximately into the same latent state. These latent state vectors were only clustered because the loss function assesses whether a state representation is reward predictive. The fact that states of the same column are assigned approximately to the same latent state is an artifact of this optimization process. The hyper-parameter n can be understood as the size of the constructed latent space and a bound on the degree of compression of the state space. For example, if the state space consists of nine different states, setting $n = 9$ could result in not compressing the state space and mapping nine states onto nine distinct one-hot bit vectors. The following experiments explore how choosing a low enough feature dimension leads to compression and generalization across states.

The previous sections present bounds on prediction errors that are parameterized in the worst-case one-step reward-prediction error ε_r and worst-case linear SF prediction error ε_ψ. Given only a finite data set of transitions $\mathcal{D} = \{(s_i, a_i, r_i, s'_i)\}_{i=1}^{D}$, it may not be possible to compute estimates for ε_r and ε_ψ without making further assumptions on the MDP at hand, such as a finite state

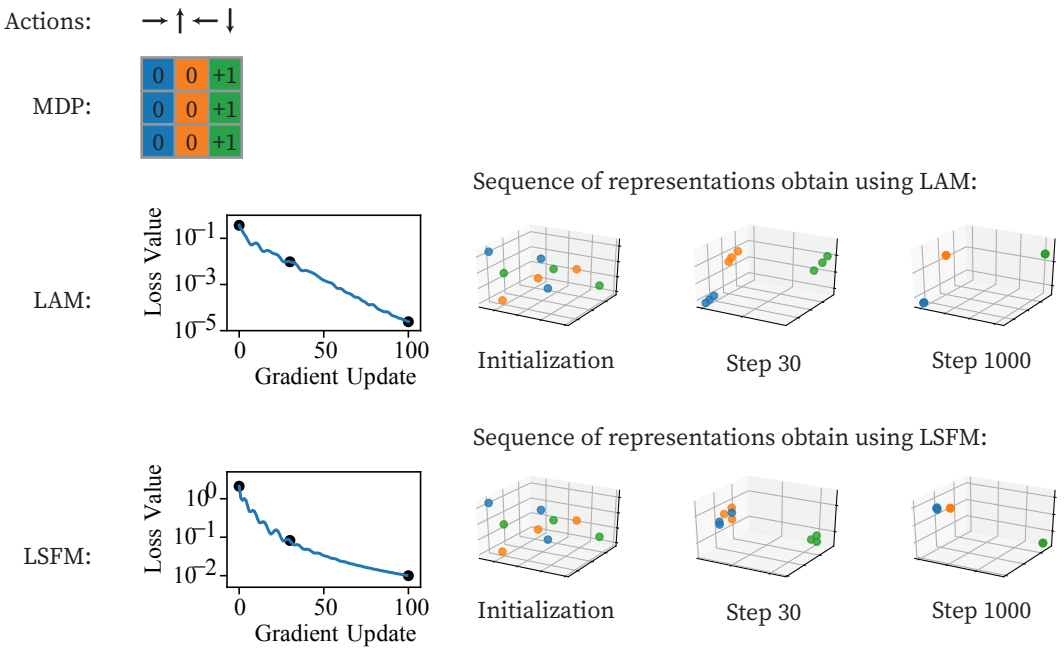


Figure 3.5: In the column world task, learning reward-predictive state representations leads to clustering grid cells by each column. The top row illustrates a map of the column world task and a colouring of each column. The middle row presents an experiment that optimizes across different state representations to find a LAM that can be used for accurate one-step reward and one-step expected transition predictions. Each latent state is plotted as a dot in 3D-space and dots are coloured by the column they correspond to. At the end of the optimization process, three clusters of the same colour are formed showing that approximately the same latent state is assigned to states of the same column. The third row repeats the same experiment using an LSFM, which assesses whether the constructed latent state space can be used for accurate one-step reward predictions and SF predictions. Appendix B.2.1 describes this experiment in detail.

space or a bounded Rademacher complexity of the transition and reward functions to obtain robust performance on a test data set [74]. Because the goal of this project is to study the connections between SFs and different models of generalization across different states, an analysis of how to find provably correct predictions of ε_r and ε_ψ given a finite data set $\mathcal{D}$ is beyond the scope of this dissertation. Instead, the conducted experiments collect a data set $\mathcal{D}$ by sampling trajectories using a policy that selects actions uniformly at random. The data set $\mathcal{D}$ is generated to be large enough to cover all state and action pairs, ensuring that all possible transitions and rewards are implicitly represented in this data set. If the data set does not cover all states and actions, then the resulting reward-predictive state representation may only produce accurate predictions for some reward sequences because the data set does not communicate all aspects of the MDP at hand. A study of this interaction between a possibly limited training data set and the resulting model's ability to make accurate predictions is left for future work.

We design a loss objective $\mathcal{L}_{\text{LSFM}}$ that is the sum of three different terms: The first term $\mathcal{L}_r$ computes the one-step reward prediction error and is designed to minimize the reward error ε_r. The second term $\mathcal{L}_\psi$ computes the SF prediction error and is designed to minimize the SF error ε_ψ. The last term $\mathcal{L}_N$ is a regularizer constraining the gradient optimizer to find a state representation that outputs unit norm vectors. Empirically, we found that this regularizer encourages the optimizer to find a model with $M \approx 1$ and $W \approx 1$. (Reward-sequence prediction errors are lower for these values of M and W, as stated in Theorem 3.22.) Given a finite data set of transitions $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}_{i=1}^D$, the formal loss objective is

$$\mathcal{L}_{\text{LSFM}} = \underbrace{\sum_{i=1}^{D} \left(\boldsymbol{\phi}_{s_i}^\top \boldsymbol{w}_{a_i} - r_i\right)^2}_{=\mathcal{L}_r} + \alpha_\psi \underbrace{\sum_{i=1}^{D} \left|\left|\boldsymbol{\phi}_{s_i}^\top \boldsymbol{F}_a - \vec{\boldsymbol{y}}_{s_i,a_i,r_i,s_i'}\right|\right|_2^2}_{=\mathcal{L}_\psi} + \alpha_N \underbrace{\sum_{i=1}^{D} \left(\left|\left|\boldsymbol{\phi}_{s_i}\right|\right|_2^2 - 1\right)^2}_{=\mathcal{L}_N}, \tag{3.24}$$

for a finite data set of transitions $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}_{i=1}^D$. In Equation (3.24), the prediction target

$$\vec{\boldsymbol{y}}_{s,a,r,s'} = \boldsymbol{\phi}_s^\top + \gamma \boldsymbol{\phi}_{s'}^\top \overline{\boldsymbol{F}}$$

and $\alpha_\psi, \alpha_{\text{N}} > 0$ are hyper-parameters. These hyper-parameters weigh the contribution of each error term to the overall loss objective. If α_ψ is set to too small a value, then the resulting state representation may only produce accurate one-step reward predictions, but not accurate predictions

of longer reward sequences. This chapter presents simulations on finite state spaces and represents a state representation as a function $s \mapsto \boldsymbol{e}_s^\top \boldsymbol{\Phi}$ where $\boldsymbol{\Phi}$ is a weight matrix of size $|\mathcal{S}| \times n$. An approximation of a reward-predictive state representation is obtained by performing gradient descent on the loss objective $\mathcal{L}_{\text{LSFM}}$ with respect to the free parameters $\{\boldsymbol{F}_a, \boldsymbol{w}_a\}_{a \in \mathcal{A}}$ and $\boldsymbol{\Phi}$. For each gradient update, the target $\vec{\boldsymbol{y}}_{s,a,r,s'}$ is considered a constant. The previously presented bounds show that prediction errors also increase with $\Delta = \max_a ||\boldsymbol{I} + \gamma \boldsymbol{M}_a \overline{\boldsymbol{F}} - \boldsymbol{F}_a||$. Minimizing $\mathcal{L}_\psi$ for a fixed state representation ϕ minimizes Δ, because

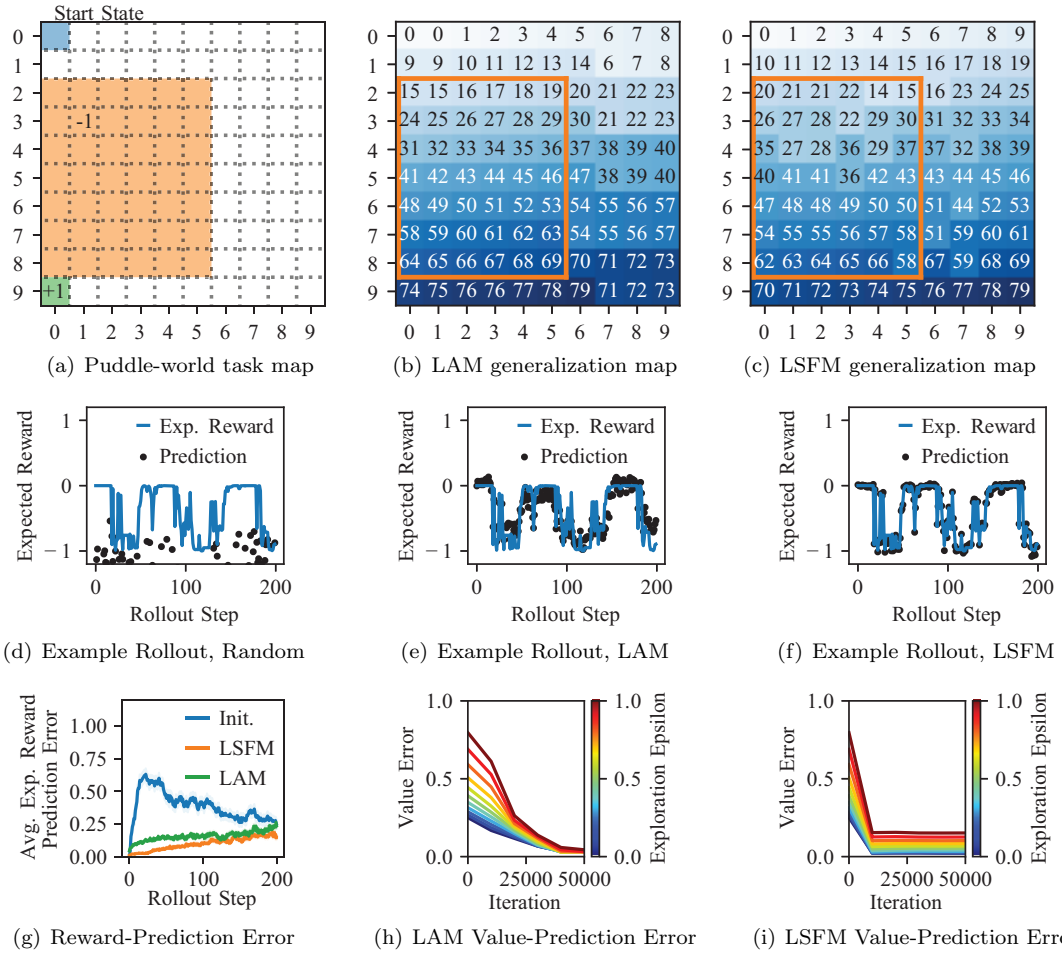
$$\Delta \leq c_\phi \mathcal{L}_\psi, \tag{3.25}$$

where c_ϕ is a non-negative constant. Appendix A.2.2 presents a formal proof for Equation (3.25).

To assess whether minimizing the loss $\mathcal{L}_{\text{LSFM}}$ leads to approximating reward-predictive state representations, a transition data set was collected from the puddle-world task [22]. Conforming to the previous analysis, the LSFM is compared to a LAM that is trained using a similar loss function, described in Appendix B.2.2.

Figure 3.6 presents the puddle-world experiments and the results. In puddle-world (Figure 3.6(a)), the agent has to navigate from a start state to a goal to obtain a reward of one while avoiding a puddle. Entering the puddle is penalized with a reward of minus one. To predict future reward outcomes accurately, a state representation has to preserve the grid position as accurately as possible to predict the location of the different reward cells.

By constraining the latent state space to 80 dimensions, the optimization process is forced to find a compression of all 100 grid cells. To analyze across which states the learned reward-predictive state representation generalizes, all feature vectors were clustered using agglomerative clustering. Two different states that are associated with feature vectors $\boldsymbol{\phi}_s$ and $\boldsymbol{\phi}_{\tilde{s}}$ are merged into the same cluster if their Euclidean distance $||\boldsymbol{\phi}_s - \boldsymbol{\phi}_{\tilde{s}}||_2$ is low. For example, a randomly chosen representation would randomly assign states to different latent states and the partition map could assign the grid cell at $(0, 0)$ and $(9, 9)$ to the same latent state. Figures 3.6(b) and 3.6(c) plot the obtained clustering as a partition map. Grid cells are labelled with the same partition index if they belong to the same cluster and colours correspond to the partition index. To predict reward outcomes accurately, the position in the grid needs to be roughly retained in the constructed latent state space. The partition maps in Figures 3.6(b) and 3.6(c) suggest that the learned state representation extracts this property from



(a) Puddle-world task map (b) LAM generalization map (c) LSFM generalization map

(d) Example Rollout, Random (e) Example Rollout, LAM (f) Example Rollout, LSFM

(g) Reward-Prediction Error (h) LAM Value-Prediction Error (i) LSFM Value-Prediction Error

Figure 3.6: Puddle-World Experiment. 3.6(a): Map of the puddle-world task in which the agent can move up, down, left, or right to transition to adjacent grid cells. The agent always starts at the blue start cell and once the green reward cell is reached a reward of +1 is given and the interaction sequence is terminated. For each transition that enters the orange puddle, a reward of −1 is received. 3.6(b), 3.6(c): Partitioning obtained by merging latent states into clusters by Euclidean distance. 3.6(d), 3.6(e), 3.6(f): Expected reward and predictions for a randomly chosen 200-step action sequence using a randomly chosen representation, a representation learned with a LAM, and a representation learned with an LSFM. 3.6(g): Average expected reward-prediction errors with standard error for each representation. 3.6(h), 3.6(i): Optimizing the loss objective results in a sequence of state representations suitable for finding linear approximations of the value functions for a range of different ε-greedy policies. Appendix B.2.2 presents more details.

a transition data set, by generalizing only across neighbouring grid cells and tiling the grid space. Only a transition data set $\mathcal{D}$ was given as input to the optimization algorithm and the algorithm was not informed about the grid-world topology of the task in any other way.

Figures 3.6(d), 3.6(e), 3.6(f) plot an expected reward rollout and the predictions presented by a random initialization (3.6(d)), the learned representation using a LAM (3.6(e)), and the learned representation using a LSFM (3.6(f)). The blue curve plots the expected reward outcome $\mathbb{E}_p[r_t|s, a_1, ..., a_t]$ as a function of t for a randomly selected action sequence. Because transitions are probabilistic, the (blue) expected reward curve is smoothed and does not assume exact values of -1 or $+1$. While a randomly initialized state representation produces poor predictions of future reward outcomes (Figures 3.6(d)), the learned representations produce relatively accurate predictions and follow the expected reward curve (Figures 3.6(e) and 3.6(f)). Because the optimization process was forced to compress 100 grid cells into a 80-dimensional latent state space, the latent state space cannot preserve the exact grid cell position and thus approximation errors occur.

Figure 3.6(g) averages the expected reward-prediction errors across 100 randomly selected action sequences. While a randomly chosen initialization produces high prediction errors, the learned state representations produce relatively low prediction errors of future reward outcomes. If expected reward-prediction errors are random after 200 time steps, then the γ-discounted return can be off by at most $0.9^{200} \cdot 1/(1-0.9) \approx 1.4 \cdot 10^{-9}$ after 200 time steps for $\gamma = 0.9$ and a reward range of $[-1, 1]$. Hence, planning over a horizon of more than 200 time steps will impact a policy's value estimate insignificantly. Reward-prediction errors decrease for a randomly chosen state representation (blue curve in Figure 3.6(g)) because the stochasticity of the task's transitions smooths future expected reward outcomes as the number of steps increases.

While the plots in Figure 3.6 suggest that both LSFMs and LAMs can be used to learn approximate reward-predictive state representations, the LSFM produces lower prediction errors for expected reward outcomes than the LAM and the LAM produces lower value-prediction errors. Because both models optimize different non-linear and non-convex loss functions, the optimization process leads to different local optima, leading to different performance on the puddle-world task. While prediction errors are present, Figure 3.6 suggests that both LSFM and LAM learn an approximate reward-predictive state representation and empirically the differences between each model are not significant.

3.4 Connection to Model-based Reinforcement Learning

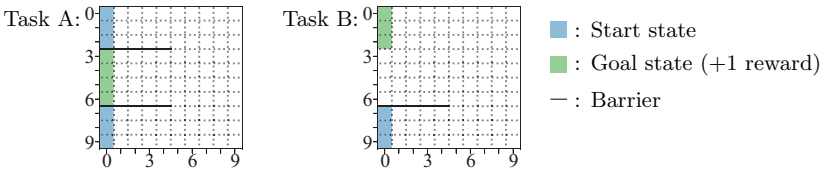
The key characteristic of a model-based RL agent is to build an internal model of a task that supports predictions of future reward outcomes for any arbitrary decision sequence. Because reward-predictive state representations construct a latent state space suitable for predicting reward sequences for any arbitrary decision sequence, learning reward-predictive state representations can be understood as form of model-based RL. LSFMs tie SFs to reward-predictive state representations, which support predictions of future reward outcomes for any arbitrary decision sequence. The presented analysis describes how learning SFs is akin to learning a transition and reward model in model-based RL.

3.5 Generalization Across Transition and Reward Functions

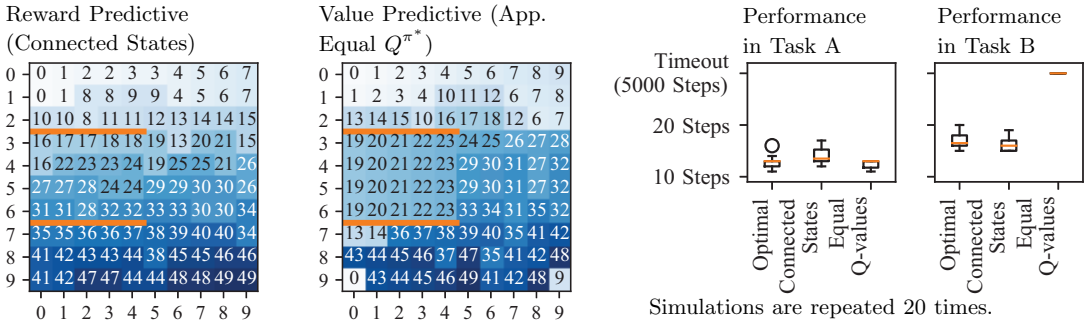
One key distinction between value- and reward-predictive state representations is their ability to generalize across different transition and reward functions. While prior work on SFs [10] and adversarial IRL [42] separately model the reward function from the transition function and observed policy, reward-predictive state representations only model equivalence relations between states separately from the transition and reward model. Consequently, reward-predictive state representations extract equivalences between different state's transitions and one-step rewards, reward-predictive state representations can be reused across tasks that vary in their transition and reward functions [65]. Figure 3.7 presents a transfer experiment highlighting that reusing a previously learned reward-predictive state representation allows an intelligent agent to learn an optimal policy using less data.

This experiment uses two grid-world tasks (Figure 3.7(a)): For Task A, a transition data set $\mathcal{D}_A$ is collected. A reward-predictive state representation is learned using an LSFM and a value-predictive state representation is learned using a form of Fitted Q-iteration [87]. These state representations are then reused without modification to learn an optimal policy for Task B given a data set $\mathcal{D}_B$ collected from Task B. Both data sets are generated by performing a random walk from a uniformly sampled start state to one of the rewarding goal states. In both tasks, the agent can transition between adjacent grid cells by moving up, down, left, or right, but cannot transition across a barrier. Transitions are probabilistic, because, with a 5% chance, the agent does not move after selecting any action.

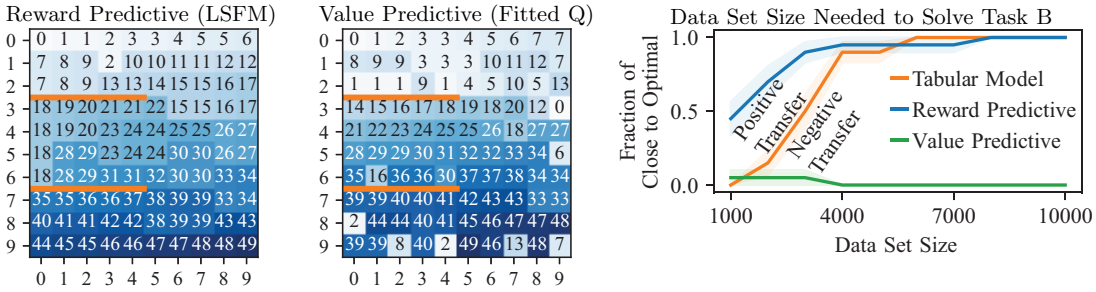
Figure 3.7(b) presents two heuristics for clustering all 100 states into 50 latent states. The first



(a) Maps of Transfer Grid Worlds



(b) State representations obtained using a clustering heuristic or optimal Q-values of Task A



(c) State partitions obtained through learning on a transition data set $\mathcal{D}$.

Figure 3.7: Reward-predictive representations generalize across variations in transitions and rewards. 3.7(b): The left panels plot state partitions obtained by clustering connected states or states with equal optimal Q-values in Task A (3.7(a)). The right panels plot the number of times steps a policy, which uses each representation, needs to complete Task B (3.7(a)). 3.7(c): The left panels plot partitions obtained by clustering latent states of a reward-predictive and value-predictive representation. The right panel plots how often one out of 20 transition data sets can be used to find an optimal policy as a function of the data set size. By reusing the learned reward-predictive representation, an agent can generalize across states and compute an optimal policy using less data than a tabular model-based RL agent. Reusing a value-predictive representation leads to poor performance, because this representation is only predictive of Task A's optimal policy.

heuristic constructs a reward-predictive state representation by joining states into the same latent state partition if they are directly connected to another. Because both tasks are navigation tasks, partitioning the state space in this way leads to approximately preserving the agent's location in the grid. The second heuristic constructs a value-predictive state representation by joining states that have approximately the same optimal Q-values. Because Q-values are discounted sums of rewards, Q-values decay as one moves further away from a goal cell. This situation leads to different corners being merged into the same state partition (for example grid cell $(0, 0)$ is merged with $(0, 9)$) and an overall more fragmented partitioning that does not preserve the agent's location. Because both state representations are computed for Task A, both state representations can be used to compute an optimal policy for Task A. For Task B, an optimal policy cannot be obtained using the value-predictive state representation. For example, both grid cells at $(0, 0)$ and $(0, 9)$ have different optimal actions in Task B but are mapped to the same latent state. Consequently, an optimal action cannot be computed using the previously learned value-predictive state representation. Because each grid cell has a different optimal action, an optimal abstract policy mapping each latent state to an optimal action cannot be found and the navigation Task B cannot be completed within 5000 time steps if the value-predictive state representation is used (Figure 3.7(b), right panel). In contrast, the reward-predictive state representation can be used, because it approximately models the grid-world topology. For Task B, each latent state has to be associated with different one-step rewards and latent transitions, but it is still possible to obtain an optimal policy using this state representation and complete the navigation task quickly.

Figure 3.7(c) repeats a similar experiment, but learns a state representation using either an LSFM to find a reward-predictive state representation or a modification of Fitted Q-iteration to find a value-predictive state representation. The two left panels in Figure 3.7(c) plot a partitioning of the state space that was obtained by clustering all latent state feature vectors using agglomerative clustering. In this experiment, the latent feature space was set to have 50 dimensions. One can observe that the state representation learned using an LSFM qualitatively corresponds to clustering connected grid cells. The learned value-predictive state representation qualitatively resembles a clustering of states by their optimal Q-values, because Fitted Q-iteration optimizes this state representation to predict the optimal value function as accurately as possible. Both state representations are tested on Task B using the following procedure: First, a data set $\mathcal{D}_B$ was collected of a fixed size. Then, Fitted Q-iteration was used to compute the optimal policy for Task B using the previously learned

state representation as a basis function such that $Q^{\pi^*}(s,a) \approx \boldsymbol{\phi}_s^\top \boldsymbol{q}_a$ where $\boldsymbol{q}_a$ is a weight vector and $\phi(s) = \boldsymbol{\phi}_s$. The state representation ϕ trained on Task A is not modified to obtain an optimal policy in Task B. If the training data set $\mathcal{D}_B$ obtained from Task B is too small, then the data set may not provide enough information to find an optimal policy. In this case, Fitted Q-iteration converged to a sub-optimal policy. Because the data sets $\mathcal{D}_B$ are generated at random, one may find that sampling a data set of 2000 transitions may lead to an optimal solution often, but not all the time.

The right panel in Figure 3.7(c) plots the dependency of being able to find an optimal policy as a function of the transition data set. For each data set size, twenty different data sets were sampled and the y-axis plots the fraction (with standard error of measure) of how often using this data set leads to a close-to-optimal policy. A close-to-optimal policy solves the navigation task in at most 22 time steps. The orange curve is computed using a tabular model-based agent, which constructs a transition and reward table using the sampled data set and solves for an optimal policy using value iteration. Reusing a reward-predictive state representation in Task B often leads to finding an optimal policy for small data set sizes (the blue curve in Figure 3.7(c), right panel). Because the training data set $\mathcal{D}_B$ is only used to inform different latent transitions and rewards, this agent can generalize across different states and reuse what it has learned without having to observe every possible transition. This behavior leads to better performance than the tabular model-based baseline algorithm, which does not generalize across different states and constructs a transition table for Task B and computes an optimal policy using value iteration [107, Chapter 4.4]. Reusing the learned value-predictive state representation leads to finding a sub-optimal policy in almost all cases (green curve in Figure 3.7(c), right panel). The value-predictive state representation is optimized to predict the Q-value function of the optimal policy in Task A. Because Task B has a different optimal policy, reusing this representation does not lead to an optimal policy in Task B, because the previously learned representation is explicitly tied to Task A's optimal policy. Note that any trial that did not find a close-to-optimal policy that completes the task in 22 time steps did also not finish the task and hit the timeout threshold of 5000 time steps. Appendix B.2.3 presents additional details on the experiments conducted in Figure 3.7.

3.5.1 Reward-Predictive Representations Encode Task Relevant State Information

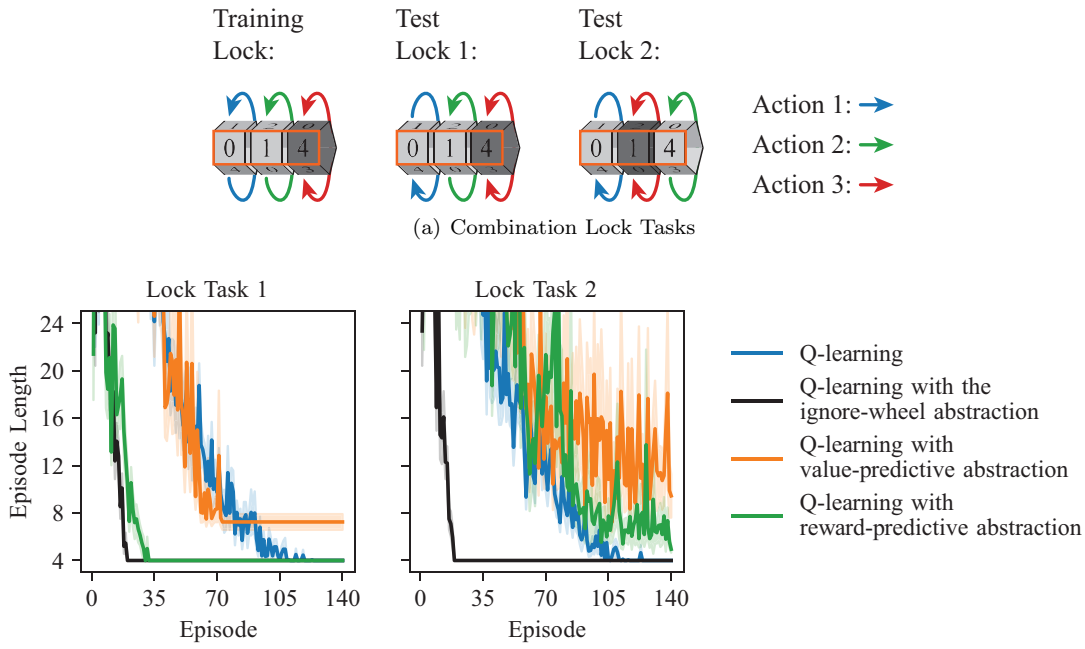
Lastly, we present a simulation result illustrating which aspect of an MDP reward-predictive state representations encode. Figure 3.8(a) illustrates the combination lock task, where an agent rotates three different numerical dials to obtain a rewarding number combination. In this task, the state is defined as a number triple and each dial has five different positions labelled with the digits zero through four. For example, if Action 1 is chosen at state $(0, 1, 4)$, then the left dial rotates by one step and the state changes to $(1, 1, 4)$. A dial that is set to four will rotate to zero. For example, selecting Action 2 at state $(0, 4, 4)$ will result in a transition to state $(0, 0, 4)$.

In the Training Lock task (Figure 3.8(a), left schematic), the right dial is "broken" and spins randomly after each time step. Consequently, Action 3 (red arrow) only causes a random change in the right dial and the agent can only use Action 1 (blue arrow) or Action 2 (green arrow) to manipulate the state of the lock. When the agent enters a combination where the left and middle dials are set to four, a reward of $+1$ is given, otherwise the agent receives no reward.[3] While the combination lock task has $5 \cdot 5 \cdot 5 = 125$ different states, the state space of the Training Lock can be compressed to $5 \cdot 5 = 25$ latent states by ignoring the position of the randomly changing right dial, because the right dial is neither relevant for maximizing reward nor predicting reward sequences.

The Test Lock 1 and Test Lock 2 tasks (Figure 3.8(a), center and right schematics) differ from the training task in that the rewarding combination is changed and the rotation direction of the left dial is reversed (Action 1, blue arrow), resembling a change in both transition and reward functions. While in Test Lock 1, the right dial still spins at random, in Test Lock 2 the middle dial rotates at random instead and the right dial becomes relevant for maximizing reward. Both test tasks can also be compressed into 25 latent states by ignoring the position of the randomly rotating dial, but in Test Lock 2 this compression would be constructed differently than in Test Task 1 or in the Training Lock.

To determine if a reward- or value-predictive state representation can be re-used to accelerate learning in a previously unseen task, we compute a reward- and a value-predictive state representation for the Training Lock MDP. Subsequently, both representations are used to compress the state space of a Q-learning agent [115] to learn an optimal policy in Test Lock 1 and Test Lock 2.

[3] Specifically, in the Training Lock task, the rewarding states are $(4, 4, 0), (4, 4, 1), \ldots (4, 4, 4)$.



(a) Combination Lock Tasks

(b) Performance of Each Model at Transfer

Figure 3.8: Combination Lock Transfer Experiment. 3.8(a): In the combination lock tasks, the agent decides between three different actions to rotate each dial by one digit. Each dial has five sides labelled with the digits zero through four. The dark gray dial is "broken" and spins at random at every time step. In the training task, any combination setting the left and middle dial to four are rewarding. In Test Lock 1, setting the left dial to two and the middle dial to three is rewarding and simulations were started by setting the left dial to two and the middle dial to four. In Test Lock 2, setting the left dial to two and the right dial to three is rewarding and simulations were started by setting the left dial to two and the right dial to four. 3.8(b): Each panel plots the episode length of the Q-learning algorithm on Lock Task 1 and Lock Task 2 averaged over 20 repeats. Note that Q-learning with the ignore-wheel abstraction uses a different abstraction in Test Lock 1 and Test Lock 2. In Test Lock 1, the ignore-wheel abstraction ignores the right dial. In Test Lock 2, the ignore-wheel abstraction ignores the middle dial. Please refer to Appendix B.2.4 for a detailed description of the experiment implementation.

Any resulting performance changes are then indicative of the state abstraction's ability to generalize from the Training Lock task to Test Lock 1 and Test Lock 2. If a state representation can generalize information from the training to the test task, then Q-learning with a state abstraction should converge to an optimal policy faster than Q-learning without a state abstraction. To combine the tabular Q-learning algorithm with a state abstraction, we assume in this simulation that a state abstraction function maps each state to a set of discrete latent states. Specifically, before updating the Q-learning agent with a transition (s, a, r, s'), this transition is mapped to a latent space transition $(\phi(s), a, r', \phi(s))$ using the respective state abstraction function ϕ. Because the latent state space is smaller than the task's original state space, one would expect that using a state abstraction function results in faster convergence, because any Q-value update is generalized across all states that are mapped to the same latent state.

The reward-predictive state representation is computed using the training task's transition and reward table with the same procedure used for the column-world task presented in Figure 3.5. (Please also refer to Appendix B.2.4 for implementation details.) To obtain a state-representation function mapping states to discrete latent states, the real-valued state representation function $\phi_{\text{real-valued}} : \mathcal{S} \to \mathbb{R}^n$ is then further refined by clustering the set of feature vectors $\{\phi_{\text{real-valued}}(s) | s \in \mathcal{S}\}$ into discrete clusters using agglomerative clustering. Each cluster then becomes a separate latent state in the resulting latent state space.

The value-predictive state representation is computed by associating two states with the same latent state if the optimal policy has equal Q-values at both states and for each action. This type of state abstraction has been previously introduced by [67] as a Q^*-irrelevance state abstraction and is predictive of Q-values because each latent state is associated with a different set of Q-values.

Figure 3.8(b) plots the episode length of four different Q-learning agent configurations. The first configuration (blue curves in Figure 3.8(b)) forms a baseline and simulates the Q-learning algorithm on both test tasks, without using any state abstraction. The second configuration, called "Q-learning with ignore dial abstraction" (black curves in Figure 3.8(b)) simulates the Q-learning algorithm with a manually coded state abstraction that ignores the right dial in Test Lock 1 and the middle dial in Test Lock 2. Because this state abstraction compresses the 125 task states into 25 latent states by removing the digit from the state triplet not relevant for maximizing reward, this variation converges significantly faster than the baseline algorithm. The third variation, called "Q-learning with value-predictive abstraction" (orange curves in Figure 3.8(b)) simulates Q-learning with the

value-predictive state abstraction. In both Test Lock 1 and Test Lock 2, this variation converges more slowly than using Q-learning without any state abstraction. As discussed in Section 1.2, a value-predictive state abstraction is constructed using the Q-values of the policy that is optimal in the Training Lock. Because each combination lock MDP has a different optimal policy, a value-predictive state abstraction cannot be transferred across any of the two tasks. Consequently, the "Q-learning with a value-predictive abstraction" agent does not converge more quickly than the baseline agent. In these simulations, the Q-learning algorithm is not capable of finding an optimal policy, as outlined previously in Figure 3.4. The green curves in Figure 3.8(b) plot the average episode length when Q-learning is combined with a reward-predictive state abstraction. In Lock Task 1, this agent converges almost as fast as the agent using the manually coded state abstraction (black curve, left panel). Only the position of the left and middle dials are relevant for predicting reward sequences $r_1, ..., r_t$ that are generated by following an arbitrary action sequence $a_1, ..., a_t$ from an arbitrary start state s in the Training Lock MDP and in the Test Lock 1 MDP. Consequently, a reward-predictive state abstraction can compress the 125 task states into 25 different latent states by ignoring the right dial, resulting in a significant performance improvement in Test Lock 1. Note that LSFMs only approximate reward-predictive state representations resulting in slightly slower convergence in comparison to the black curve in the left panel of Figure 3.8(b).

This behaviour changes in Test Lock 2, because in Test Lock 2 a different dial moves at random, changing how the state space should be compressed. Because the right dial is relevant for predicting expected reward sequences in Test Lock 2, the reward-predictive state representation learned in the Training Lock task is no longer reward-predictive in Test Lock 2 and cannot be re-used without modifications. Consequently, the "Q-learning with reward-predictive abstraction" agent exhibits worse performance (Figure 3.8(b), right panel, green curve) than using Q-learning without any state abstraction (Figure 3.8(b), right panel, blue curve).

The results presented in Figure 3.8 demonstrate that reward-predictive state representations encode which state information is relevant for predicting reward sequences and maximizing reward in a task. Because reward-predictive state representations only model this aspect of an MDP, this type of state representation generalizes across tasks that preserve these state equivalences but differ in their transitions and rewards otherwise. In contrast, value-predictive state representations "overfit" to a specific MDP and the MDP's optimal policy, resulting in negative transfer and possibly prohibiting an agent from finding an optimal policy at transfer.

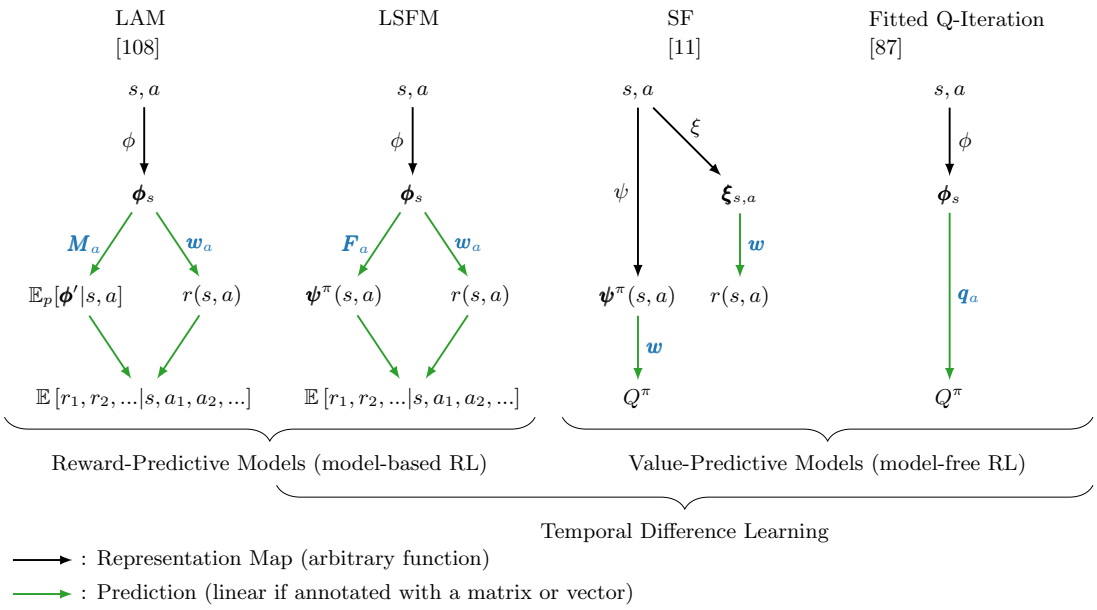


Figure 3.9: Comparison of Presented State-Representation Models.

3.6 Discussion

This chapter presents a study of how successor features are tied to reward-predictive representation learning and model-based RL. Connections are drawn by analyzing which properties different latent state spaces are predictive of. The schematic in Figure 3.9 illustrates the differences between the presented models. By introducing LSFMs, SFs are tied to learning state representations that are predictive of future expected reward outcomes. This model ties successor features to model-based reinforcement learning, because an agent that has learned an LSFM can predict expected future reward outcomes for any arbitrary action sequence. Because SFs obey a fixed-point equation similar to the Bellman fixed-point equation, SFs can also be linked to temporal-difference learning. Similar to LAMs, LSFMs are a "strict" model-based architecture and are distinct from model-based and model-free hybrid architectures that iteratively search for an optimal policy and adjust their internal representation [80, 116, 38, 44]. LSFMs only evaluate SFs for a fixed target policy that selects actions uniformly at random. The learned model can then be used to predict the value function of any arbitrary policy, including the optimal policy. In contrast to model-based and model-free hybrid architectures, the learned state representation does not have to be adopted to predict an optimal policy and generalizes across all latent policies. How to learn neural networks mapping inputs to latent feature spaces that are predictive of future reward outcomes is beyond the scope of

this dissertation and is left for future work.

Similar to reward-predictive state representations, the Predictron architecture [97] and the MuZero algorithm [92] use or construct a state representation to predict reward sequences. In contrast to reward-predictive state representations, these other architectures predict reward sequences for k time steps and then use the value function for one (or multiple) policies to predict the return obtained after k time steps. This distinction is key in learning reward-predictive state representations with LSFMs, which do not include a value prediction module, because if a state representation is designed to predict the value function of a policy, then the resulting state representation would be (to some extent) value predictive. As outlined in Section 3.5, this change would compromise the resulting state abstraction's ability to generalize across different transition and reward functions.

In contrast to the SF framework introduced by [11], the connection between LSFMs and model-based RL is possible because the same state representation ϕ is used to predict its own SF (Figure 3.9 center column). While the deep learning models presented by [61] and [123] also use one state representation to predict SFs and one-step rewards, these models are also constrained to predict image frames. LSFMs do not use the state representation to reconstruct actual states. Instead, the state space is explicitly compressed, allowing the agent to generalize across distinct states.

Table 3.1 summarizes different properties of the presented state representations. Bisimulation relations [46] preserve most structure of the original MDP and latent transition probabilities match with transition probabilities in the original MDP (Section 3.1). Reward-predictive state representations do not preserve the transition probabilities of the original task (Figure 3.3) but construct a latent state space that is predictive of expected future reward outcomes. These two representations generalize across all abstract policies, because they can predict reward outcomes for arbitrary action sequences. Successor features are equivalent to value-predictive state representations, which construct a latent state space that is predictive of a policy's value function (Chapter 2). Because the value function can be factored into SFs and a reward model (Equation (2.8)), SFs are robust to variations in reward functions. Reward-predictive state representations remove previous limitations of SFs and generalize across variations in transition functions. This property stands in contrast to previous work on SFs, which demonstrate robustness against changes in the reward function only [11, 12, 61, 123, 100, 75, 89]. In all cases, including reward-predictive state representations, the learned models can only generalize to changes that approximately preserve the latent state space

Model	Predicts Trained Policy	Generalizes to Variations in Rewards	Generalizes to Variations in Transitions	Generalizes Across All Policies	Predicts Transition Probabilities
Bisimulation [46]	yes	yes	yes	yes	yes
Reward-Predictive (LSFM or LAM)	yes	yes	yes	yes	no
Successor Features [11]	yes	yes	no	no	no
Value-Predictive (Fitted Q-iteration)	yes	no	no	no	no

Table 3.1: Summary of Generalization Properties of Presented State Representations

structure. For example, in Figure 3.7 positive transfer is possible because both tasks are grid worlds and only the locations of rewards and barriers is changed. If the same representation is used on a completely different randomly generated finite MDP, then positive transfer may not be possible because both tasks do not have a latent state structure in common.

In comparison to previous work on state abstractions [33, 67, 2, 3, 4], this chapter does not consider state representations that compress the state space as much as possible. Instead, the degree of compression is set through the hyper-parameter that controls the dimension of the constructed latent state space. The presented experiments demonstrate that these state representations compress the state space and implicitly convey information useful for transfer. This formulation of state representations connects ideas from state abstractions to models that analyze linear basis functions [82, 104, 59] or learn linear representations of the transition and reward functions [98]. Recently, [88] presented algorithms to cluster approximately bisimilar states. Their method relies on bisimulation metrics [34], which use the Wasserstein metric to assess if two state transitions have the same distribution over next state clusters. In contrast to their approach, we phrase learning reward-predictive state representations as an energy-minimization problem and remove the requirement of computing the Wasserstein metric.

The presented experiments learn state representations by generating a transition data set covering all states of an MDP. Complete coverage is obtained on the grid world tasks by generating a large enough transition data set. Because this article focuses on drawing connections between different models of generalization across states, combining the presented algorithms with efficient exploration algorithms [52] or obtaining sample complexity or regret bounds similar to prior work [51, 8, 81] is left to future studies.

3.7 Conclusions

This chapter presents an analysis of which latent representations an intelligent agent can construct to support different predictions, leading to new connections between model-based and model-free RL. By introducing LSFMs, the presented analysis links learning successor features to model-based RL and demonstrates that the learned reward-predictive state representations are suitable for transfer across variations in transitions and rewards. The presented results outline how different models of generalization are related to another and proposes a model for phrasing model-based learning as a representation-learning problem. These results motivate the design and further investigation of new approximate model-based RL algorithms that learn state representations instead of one-step reward and transition models.

Chapter 4

Reward-Predictive Representations Generalize Across Tasks

In this chapter, we support the thesis stated in Chapter 1 and present a sequence of transfer simulations suggesting that reward-predictive representations are robust to transfer across tasks with different rewards and transitions. As discussed in the previous chapters, 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. In this chapter we discuss the dichotomy between 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 sequences.

While different RL transfer algorithms have been proposed (see [111] for a survey), we demonstrate 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 [63] shows that reward-predictive state abstractions can be extracted from the successor representation (SR) [29], 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 [75, 11, 12], 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, as discussed in Chapter 2. In contrast, in this chapter we show 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 behavioural and neural signatures of human structure learning [26, 9, 41] but not afforded by alternative algorithms that compress the transition function itself directly [100, 89].

To unpack the relative advantages of distinct state abstraction algorithms for generalization, we proceed as follows. In the following section, we begin with a simple illustration of the state-abstraction framework and then present the conceptual utility of reward-predictive state abstractions. Next, we present our first result by examining 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 4.2.1). Subsequently, we consider 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 (Section 4.2.2). Extending our simulations to an online learning setting, 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 (Section 4.2.3). Finally, we demonstrate how this advantage can be leveraged in a guitar playing task, whereby an agent can reapply learned structure about the fret-board while learning a musical scale to quickly learn to play other scales that differ in transitions, rewards, and policy (Section 4.2.4).

4.1 Generalization Across States

In model-free RL, for example Q-learning [115], 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 RL [103, 23] 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 [103].

While several approaches exist for constructing a useful state abstraction ϕ for complex MDPs, this chapter investigates which state abstractions facilitate re-use across different tasks. Specifically, we consider the question of which algorithm should be used to learn a state abstraction ϕ from a hypothesis space of all possible state abstractions $\mathcal{H}$ to 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

$$\mathcal{H} = \{\phi : \mathcal{S} \rightarrow \mathcal{S}_\phi\} \tag{4.1}$$

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.

In this chapter, state abstractions are generated in one of two ways:

1. Enumerate all possible state abstractions using Algorithm U [58]. This method is used in Figures 4.2 and 4.5.

2. Learning a state-abstraction function from transition data. This method is used in Figures 4.6 and 4.7 .

Reward-predictive state abstractions can be learned using Linear Successor Feature Models (LSFMs) (see Chapter 3) as a means to learn reward-predictive state abstractions. Successor Features [10] are a generalization of the SR [29], and predict the expected visitation frequencies in some latent

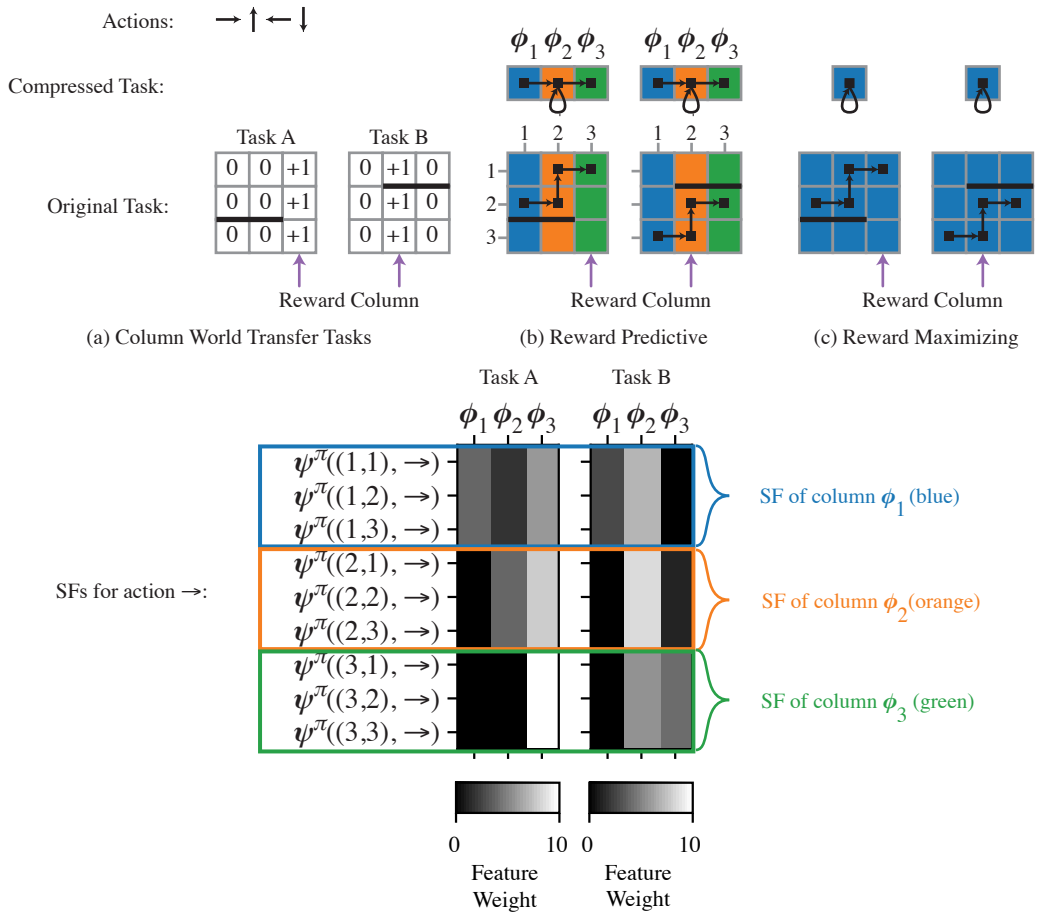
feature space:

$$\boldsymbol{\psi}^{\pi}(s,a) = \mathbb{E}\left[\phi(s_1) + \gamma\phi(s_2) + \gamma^2\phi(s_3) + \cdots \,\middle|\, s = s_1, a = a_1, \pi\right], \tag{4.2}$$

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 Equation (4.2). 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 $\phi(s_t)$ (Equation (4.2)) (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.

As discussed in Chapter 3, 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. In contrast to SFs, reward-predictive state abstractions are independent of a specific policy and can be used to generalize across all policies that are defined in terms of the latent states. More concretely, by Theorem 4 presented in Chapter 3 a reward-predictive state abstraction can be used to predict the value of any arbitrary abstract policy by first predicting which reward sequence a specific policy generates and then computing the discounted sum over this reward sequence. Figure 4.1 presents an intuitive transfer example and plots different SFs for each task. Consequently, an agent would have to adjust a previously learned SF.

An alternative to using LSFMs are Linear Action Models (LAM), which predict the expected next state instead of SFs. Because we found that LSFMs are easier to use than LAMs in practice, this article focuses on LSFMs.



(d) Reward-predictive state representations merge states with equal successore features into the same latent state.

Figure 4.1: State Abstraction Transfer. 4.1(a): Variations of the column world tasks presented in Figure 1.1(a). Both Task A and Task B differ in their rewards and transitions, because a different column is rewarded and the barrier (thick black line) is placed at different locations. 4.1(b): A reward-predictive state representation generalizes across different columns, similar to Figure 1.1. 4.1(d): Each row in the shown matrix plots visualizes the entries of a three dimensional SF vector. These matrix plots illustrate that SFs, which are computed for each task's optimal policy, are different in each task and cannot be immediately reused in this example. Yet, 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. 4.1(c): One possible reward-maximizing state abstraction generalizes across all states. Although this abstraction can be used to learn an optimal policy in Task A (i.e., always go right), this abstraction cannot be used to learn the optimal policy in Task B in which the column position is needed to determine whether to go left or right. While reward-maximizing abstractions may compress a task further than reward-predictive abstractions, reward-maximizing abstractions may also simplify a task to an extend that renders them proprietary to a single task. In contrast, reward-predictive representations are suitable for re-use across tasks that vary in rewards and transitions.

4.2 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 Figure 4.1, 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 (Figure 4.1(b)) can be re-used to plan a different policy in Task B, while the reward-maximizing state abstraction (Figure 4.1(c)) 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 [40].

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 latent states, then such a state representation is always reward maximizing and reward predictive. However, such a state representation is not "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 the context of our online learning simulations in the following section (Section 4.2.3).

4.2.1 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 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. (Out of a set of tasks, one task was randomly chosen for evaluation of a single state abstraction ϕ. Subsequently, this state abstraction ϕ is evaluated in all other remaining transfer tasks. In all simulations, the evaluation

and transfer tasks are distinct.) For each state abstraction in $\mathcal{H}$, we computed a compressed abstract MDP [85] for every tested MDP and solved it using value iteration [106, Chapter 4.4] (please also refer to Appendix B.3.1). 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

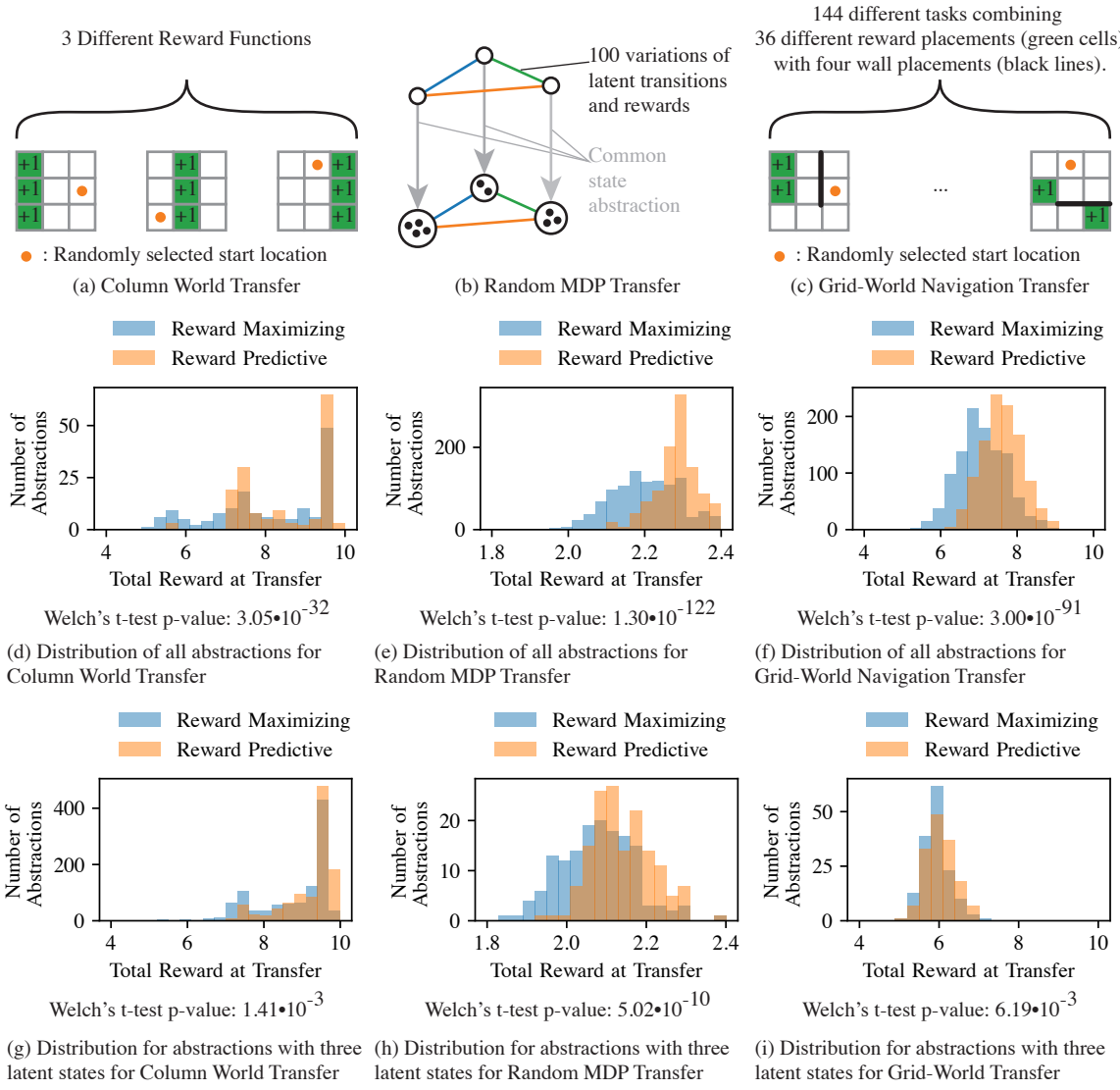
$$R_{\text{total}} = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} r_{n,t}, \tag{4.3}$$

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

$$RS_{\text{error}} = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} |r_{n,t} - \hat{r}_{n,t}|. \tag{4.4}$$

We considered three types of tasks: column-worlds (like those in the motivating example), 100 randomly generated MDPs, and grid worlds (Figures 4.2(a), 4.2(b), and 4.2(c)). For each transfer experiment, all possible state abstractions are enumerated using Algorithm U [58] to obtain a ground truth distribution over the hypothesis space $\mathcal{H}$. 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 Figure 4.2. 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 Figure 4.2) 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 the following two sections, we will present an extension to environments in which multiple structures are possible and have to be inferred.

Figures 4.2(d) and 4.2(g) present the results for the transfer experiment discussed in Figure 4.1. Both histograms 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 $\mathcal{H}$



(a) Column World Transfer

(b) Random MDP Transfer

(c) Grid-World Navigation Transfer

(d) Distribution of all abstractions for Column World Transfer

(e) Distribution of all abstractions for Random MDP Transfer

(f) Distribution of all abstractions for Grid-World Navigation Transfer

(g) Distribution for abstractions with three latent states for Column World Transfer

(h) Distribution for abstractions with three latent states for Random MDP Transfer

(i) Distribution for abstractions with three latent states for Grid-World Transfer

Figure 4.2: Minimizing reward-sequence prediction errors identifies state abstractions amenable for "deep transfer". In each grid-world task (4.2(a), 4.2(c)) the agent can transition up, down, left, or right to move to an adjacent grid cell. If the agent attempts to transition of the grid or across one of the black barriers in 4.2(c), then the agent remains at its current grid position. 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. 4.2(d), 4.2(e), 4.2(f): The histograms report averages over all repeats and transfer MDPs for all state abstractions that are possible in a nine state MDP. 4.2(g), 4.2(h), 4.2(i): The histograms report averages over all repeats and transfer MDPs for all state abstractions that compress nine states into three latent states. The p-values show that the mean difference in total reward is significant in each histogram.

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 Figure 4.2(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 (Figures 4.2(e) and 4.2(h)). 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 in Figure 4.2(h), but the shape changes and the median shifts. This shift 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 an 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. Figure 4.2(c) presents a transfer experiment where two reward locations and four different wall placements are permuted in a grid world. These changes in reward and wall locations resemble changes in the transition and reward functions. 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 Figures 4.2(f) and 4.2(i) 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 histogram in Figure 4.2(f) 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.

4.2.2 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 [26, 40], which it can use for inference. Figure 4.3 illustrates how the curriculum of tasks is randomly generated. This task curriculum is observed in sequence by the non-parametric Bayesian model and the model is signalled when a switch between tasks occurs. 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:

$$\Pr(\phi|M_t, \mathcal{B}_t, c_t) \propto \Pr(\phi|M_t)\Pr(\phi|\mathcal{B}_t, c_t), \tag{4.5}$$

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) [45, 112] prior for an intensity

$\alpha \geq 0$:

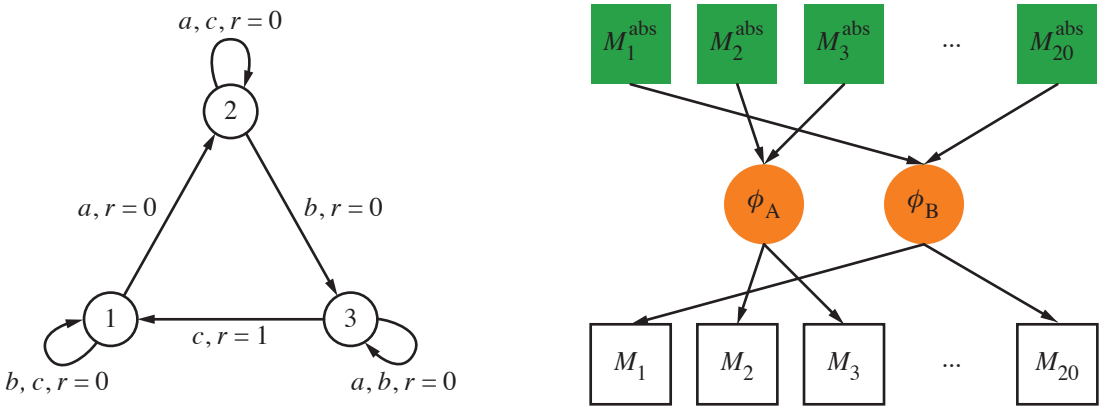
$$\Pr(\phi|\mathcal{B}_t, c_t) = \begin{cases} \frac{c_t(\phi)}{t-1+\alpha} & \text{if } \phi \in \mathcal{B}_t \\ \frac{\alpha}{t-1+\alpha} & \text{otherwise.} \end{cases} \tag{4.6}$$

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

$$\Pr(\phi|M_t) \propto e^{-\beta l(\phi)}. \tag{4.7}$$

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 $\mathcal{B}_t \cup \{\phi_{\text{next-best}}\}$. (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 all possible state abstractions $\phi \in \mathcal{H}$ gives insight into which state abstractions generalize across different tasks, while being algorithm agnostic; below we relax the need for an oracle.) In contrast to the previously presented simulation, this non-parametric Bayesian agent is constrained to only use state abstractions that compress nine-state MDPs to three-state MDPs. Consequently, the model is forced to generalize across different states and cannot default to only using the identity state abstraction, which does not compress an MDP and is both reward-predictive and reward-maximizing. If α increases, the resulting prior and posterior assign a higher probability to adding the next-best state abstraction $\phi_{\text{next-best}}$ into $\mathcal{B}_t$. In this case, the CRP prior influences the posterior more strongly. 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 $\mathcal{B}_t \cup \{\phi_{\text{next-best}}\}$ and the CRP prior is effectively ignored.

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



(a) Abstract MDP $M_{\phi,i}$

(b) Sequence of 20 MDPs

Figure 4.3: Transfer with Multiple State Abstractions Curriculum. 4.3(a): A curriculum of transfer tasks is generated by first constructing the three-state MDP. 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. 4.3(b) 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 [40]). 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.

abstract MDP:

$$l_{\text{maximizing}}(\phi) = \max_{s \in \mathcal{S}} \left(V^{\pi^*}(s) - V^{\pi_\phi}(s) \right), \tag{4.8}$$

where V^{π^*} is the optimal discounted value function [106], π^* is the optimal policy, and V^{π_ϕ} is the discounted value function of the policy π_ϕ evaluated in the task itself. Reward-predictive state abstractions are scored by the loss function $l_{\text{predictive}}$ bounding the reward-sequence prediction error

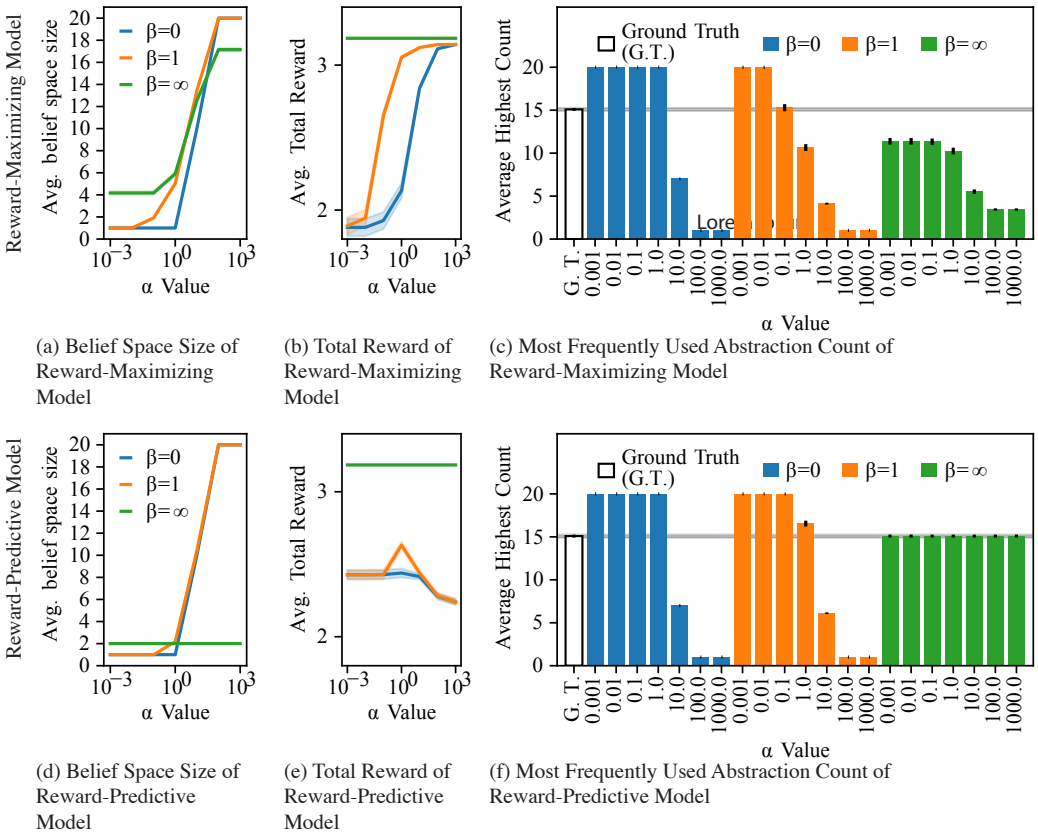
$$\forall i, \forall a_1, ..., a_t,\ l_{\text{predictive}}(\phi) C_{\gamma,t} \geq \left| \mathbb{E}[r_t | i, a_1, ..., a_t] - \hat{r}_t \right|, \tag{4.9}$$

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 LSFM using Equation (B.2).[1]. 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. For reward-predictive state abstractions, this property holds because each tested task in Figure 4.3 can be compressed (by construction) and the LSFM Bisimulation condition (Theorem 2 in Chapter 3) applies. In this case, if $l_{\text{predictive}} = 0$, then the state abstraction ϕ can be used to predict reward-sequences accurately. (Alternatively, one could also use RS_{error} as defined in Equation (4.4).)

Figure 4.4 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 (as described in the previous section, policies are computed using value iteration on the abstract MDP). 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 Equation (4.5). 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

[1] Here we use the same procedure described in Appendix B.2.1



(a) Belief Space Size of Reward-Maximizing Model

(b) Total Reward of Reward-Maximizing Model

(c) Most Frequently Used Abstraction Count of Reward-Maximizing Model

(d) Belief Space Size of Reward-Predictive Model

(e) Total Reward of Reward-Predictive Model

(f) Most Frequently Used Abstraction Count of Reward-Predictive Model

Figure 4.4: Results for transfer with multiple state abstractions experiment. 4.4(a), 4.4(d): Plot of how different α and β model parameters influence the average size of $\mathcal{B}_t$ after training. 4.4(b), 4.4(e): Performance of each model (average total reward per MDP) for different α and β model parameters. After observing the transition and reward tables of a task M_t in the task sequence, the average total reward was obtained by first computing a compressed abstract MDP for each abstraction and then solving each compressed MDP using value iteration, as described in Appendix B.3.1. The resulting mixture policy was then tested in the task M_t for 10 time steps while logging the sum of all obtained reward. If $\beta = \infty$ the agent obtains an optimal total reward level when using either loss function for ten time steps in each MDP. 4.4(c), 4.4(f): Plot of the average count for the most frequently used state abstraction. As described in Figure 4.3, 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).

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. Figure 4.4(f) shows 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, Figures 4.4(a) and 4.4(b) demonstrate that 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 between four and 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 the following section, 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 and obtain an optimal reward level while only using two different state abstractions (green curves in Figures 4.4(d) and 4.4(e)). This confirms the claim that reward-predictive state abstractions generalize across different tasks.

4.2.3 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 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 [39] that is updated after interacting for a certain number of

episodes in a particular task. 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 interacting for 200 episodes in a task. (We found that 200 episodes allow each tested algorithm to converge to an optimal policy.) Subsequently, the collected data is used to learn a new state abstraction. A reward-predictive state abstraction is obtained using the LSFM discussed in Section 4.1. A reward-maximizing state abstraction is obtained by clustering states with approximately equal Q-values into latent states. After interacting with a task and learning a new state abstraction, the belief set $\mathfrak{B}_t$ is updated using the posterior probabilities $\Pr(\phi|M_t, \mathfrak{B}_t, c_t)$. During learning in the next task, the state abstractions stored in the belief set $\mathfrak{B}_t$ are used to generalize Q-values across different states during learning in a task. 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. (If Q-learning would normally observe a transition (s, a, r, s'), the algorithm now observes a transition $(\phi(s), a, r, \phi(s'))$. Because Q-learning caches Q-values for latent states and multiple states map to the same latent state, the agent now generalizes Q-values across multiple states and can thus converge faster.) 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[39, 40], 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

$$\pi(s,a) = \sum_{\phi \in \mathfrak{B}_t \cup \{\phi_{\text{identity}}\}} \Pr(\phi|M_t, \mathfrak{B}_t, c_t)\pi_\phi(s,a), \tag{4.10}$$

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 Equation 4.5 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 Figure 4.5. The top row depicts two different maze maps that are used to construct a curriculum of five tasks. Each map 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

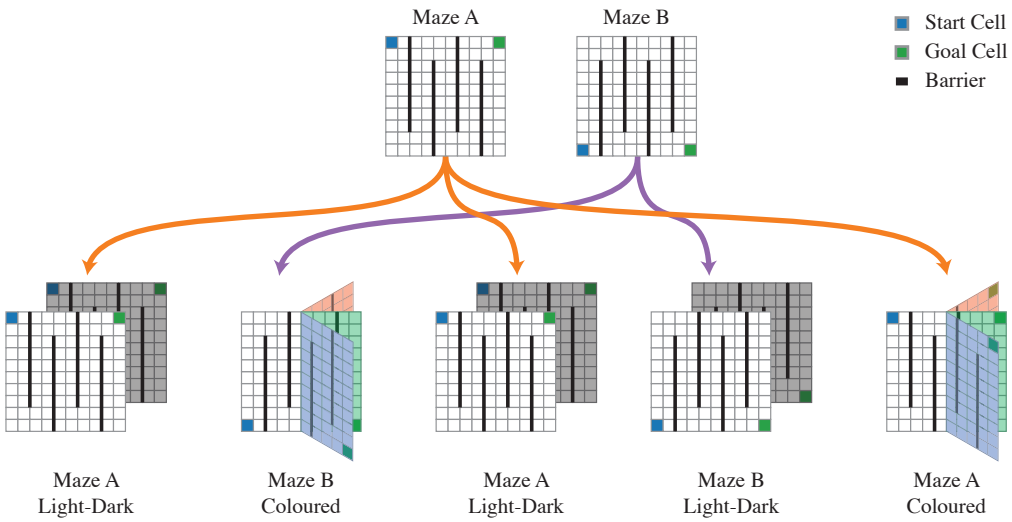


Figure 4.5: Maze curriculum. Maze A and Maze B are augmented with an irrelevant state variable to construct a five-task curriculum. In each maze, the agent starts at the blue grid cell and can move up, down, left, or right to collect a reward at the green goal cell. The black lines indicate barriers the agent cannot pass. Once the green goal cell is reached, the episode finishes and another episode is started. (These rewarding goal cells are absorbing states.) Transitions are probabilistic and succeed in the desired direction with probability 0.95; otherwise the agent remains at its current grid cell and cannot transition off the grid map or through a barrier.. 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 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 Figure 4.5 illustrates how the task sequence is constructed and how additional state variables are introduced. In the bottom left of Figure 4.5, 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 100 are equivalent and can be compressed to one latent state, for example.

The task "Maze B Coloured" (second map in bottom row of Figure 4.5) 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 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.

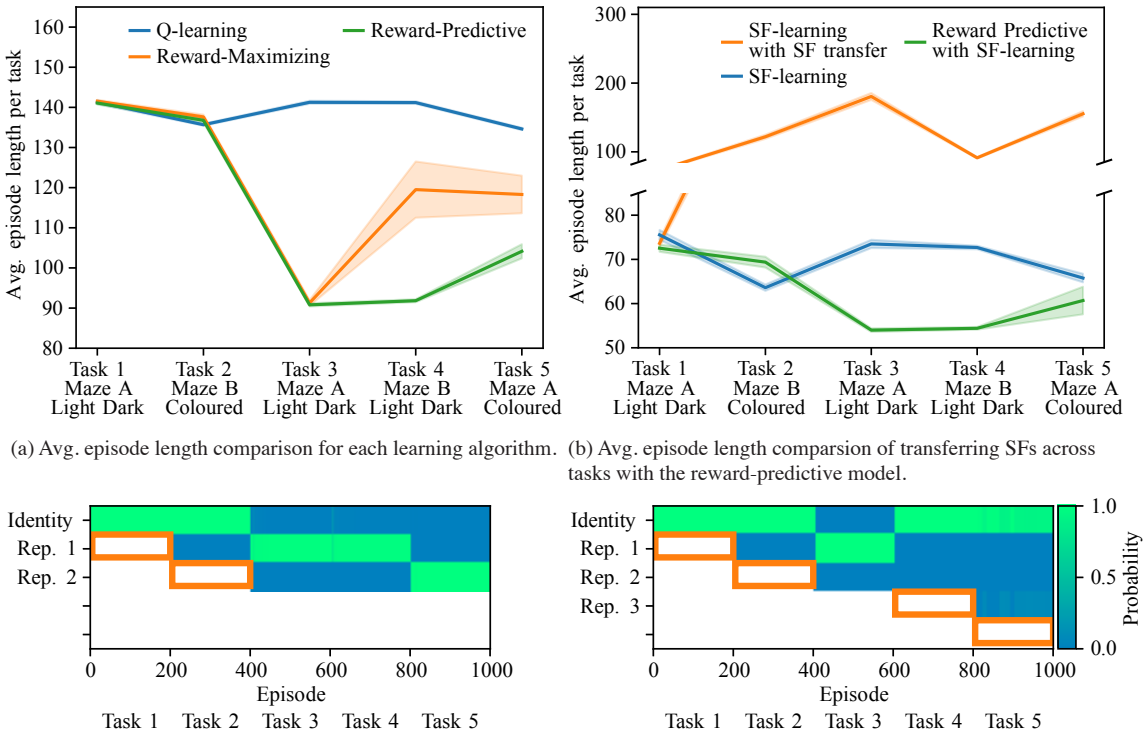
The bottom row of Figure 4.5 depicts the five-task curriculum. In this experiment, an agent can either learn how to maximize reward in each of the 200-state tasks or learn how to compress each task into 100 latent states, generalize information across different tasks, and ultimately learn an optimal policy faster and generate higher total reward.

Figure 4.6 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 Figure 4.6(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 Figure 4.6(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 Figures 4.6(c) and 4.6(d). On the first two tasks, the reward-predictive model adds two new state abstractions into its belief set (Figures 4.6(c)). 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 Figure 4.6(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. Specifically, on Task 3 and Task 4 the reward-predictive model re-uses the state abstraction learned in Task 1, though these tasks use different mazes (Maze A and Maze B in Figure 4.5); indicating that the learned state abstraction only models light/dark state equivalences and is independent of the transitions and rewards themselves. Similarly, on Task 5 the reward-predictive model re-uses the state abstraction learned in Task 2, despite both tasks using different mazes; indicating that this state abstraction only models colour state equivalences and does not depend on the transitions or rewards of either maze. 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 Figure 4.6(a)). The posterior probability plot for this model (Figure 4.6(d)) 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.

Figure 4.6(b) compares the average episode length of the reward-predictive model with transferring and adjusting SFs, the system used in prior work [11, 12, 64, 75, 89, 100, 123, 61, 70]. In the tested grid-world tasks, we found that our SF-learning algorithm implementation in combination with the used initialization heuristics converges faster to an optimal policy than the Q-learning algorithm, resulting in a shorter average episode lengths. The reward-predictive model can be adopted to use the SF-learning algorithm instead of the Q-learning algorithm and this model is presented in



(a) Avg. episode length comparison for each learning algorithm.

(b) Avg. episode length comparsion of transferring SFs across tasks with the reward-predictive model.

(c) Posterior probabilities during learning for the reward-predictive model.

(d) Posterior probabilities during learning for the reward-maximizing model.

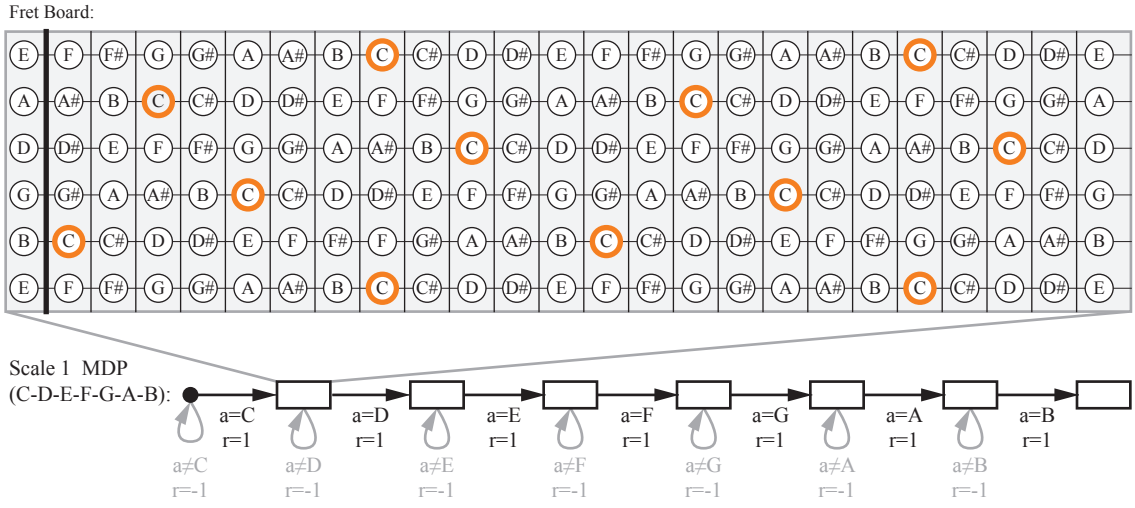
Figure 4.6: Transferring state representations influences learning speed on the maze curriculum. 4.6(a): Performance comparison of each learning algorithm that uses Q-learning to obtain an optimal policy. The reward-predictive model identifies two state abstractions and re-used them in tasks 3 through 5, resulting in faster learning than the reward-maximizing model. 4.6(b): Performance comparison of each learning algorithm that uses SF-learning to obtain an optimal policy. Similar to (A), the reward predictive model identifies two state abstractions and re-used them in tasks 3 through 5. Re-using previously learned SFs across tasks (orange curve) degrades performance. 4.6(a), 4.6(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. In Appendix B.3.2, Tables B.6 and B.7 lists the used hyper-parameters in detail. Figure B.1 also illustrates how the Bayesian model parameters α and β influence the number of learned abstractions and their performance. 4.6(c), 4.6(d): 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.

Figure 4.6(b). The blue curve in Figure 4.6(b) plots the average episode length when the SF-learning algorithm is used to find an optimal policy. In this simulation, the SF-learning algorithm does not transfer a representation and instead resets its weights when switching between tasks. The orange curve plots the average episode length when the SF-learning re-uses previously learned SFs instead of resetting its representation. Figure 4.6(b) demonstrates that re-using SFs degrades performance on the maze task sequence while the reward-predictive model outperforms a SF-learning baseline. 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. 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.

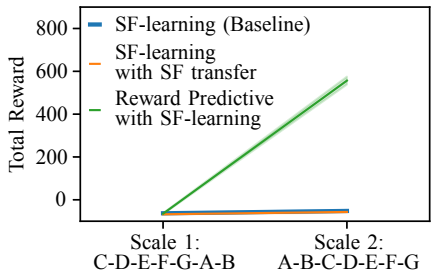
4.2.4 Comparison to Transferring Successor Features

Lastly, in this section, we illustrate the differences between re-using reward-predictive state abstractions and re-using successor features themselves [75, 89, 11]. Although reward-predictive state abstractions can be extracted from successor features, the resulting abstraction is a more abstract aspect of an MDP than simply reusing the successor features. Figure 4.7 presents a guitar-playing example to illustrate this idea. In this example, the task is to play a guitar scale, a sequence of notes such as C-D-E-F-G-A-B. On a guitar, the note "C" can be played by holding down a finger at one out of multiple possible locations on the fret board, as illustrated in Figure 4.7(a) . (Even within the same octave, the note "C" can be played in up to five different ways.) A skilled guitarist has internalized a representation that links fret-board positions to the notes they produce. In this example, a reward-predictive state abstraction captures this aspect of mapping all positions on the fret board to a latent state of playing the note "C".

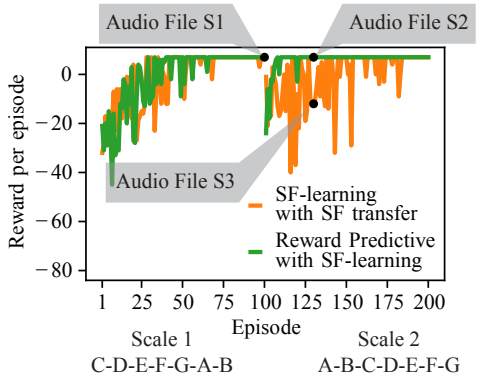
The guitar-scale task illustrated in Figure 4.7(a) is constructed such that the agent always starts at a separate start state. To play a scale correctly, the agent has to select an action sequence that corresponds to playing the note sequence correctly. The state is represented as a bit matrix, where each entry corresponds to a position (circle) on the fret board. Because the note "C" can be played at multiple fret-board locations (orange circles), each location is mapped to the same latent state. In the guitar-scale task the agent transitions through a sequence of fret-board locations by playing



(a) Guitar-Scale Task



(b) Total-reward obtained by each learning algorithm.



(c) Reward per episode during learning in Scale 1 and 2.

Figure 4.7: Guitar-Playing example. 4.7(a): Guitar-Scale task for scale C-D-E-F-G-A-B. The bottom schematic illustrates how the guitar-scale MDP is constructed for one octave: Starting at the start state (black dot), the agent progresses through different fret-board configurations by selecting which note to play next. For each correct transition, a reward of zero is given, and for each incorrect transition a reward of -1 is given. 4.7(b): Total reward for each algorithm after first learning an optimal policy for Scale 1 (C-D-E-F-G-A-B) and then learning an optimal policy for Scale 2 (A-B-C-D-E-F-G). 4.7(c): Reward per episode plot of one repeat for both the SF transfer and reward-predictive model. For the first 100 episodes, which are spent in scale task 1, both algorithms converge to an optimal reward level equally fast and learn to play the scale correctly. A recording of the optimal scale sequence is provided in Audio File S1. On scale task 2 (episodes 101 and onward), the reward-predictive model can re-use a previously learned state abstraction and converge to an optimal policy faster than the SF transfer algorithm. After only ten episodes in scale task 2, the reward-predictive model has learned how to play the scale correctly (please refer to Audio File S2) while the SF transfer algorithm has not yet converged to an optimal policy and does not play the scale correctly (please refer to Audio File S3).

a sequence of notes. Rewards are only maximized across time if the agent plays the correct scale (Figure 4.7(a), bottom schematic). Note that the illustrated state sequence is repeated three times, once for each octave. (The schematic illustrates only one chain to simplify the presentation.) Which octave is played is determined at random and the transition from the start state (black dot) into one of the fret boards that correspond to the note "C" is non-deterministic. This assumption allows us to reduce the action space from 60 fret board positions to 12 notes (A, A#, B, C, C#, D, D#, E, F, F#, G, G#). The last fret board (a fret board corresponding to the note "B" in this example) is an absorbing state.

For a sequence of two guitar-scale tasks, Figure 4.7(b) compares the performance of a reward-predictive model with that of transferring previously learned SFs. Note that these two guitar-scale tasks differ in their transitions, rewards, and optimal policy. While all algorithms perform similarly in learning the first scale (given that they have to learn the abstraction), only the reward-predictive model (green curve) exhibits transfer to the second scale. Each algorithm was simulated in each task for 100 episodes and each simulation was repeated ten times. (Table B.8 lists the used hyper-parameters.) Figure 4.7(c) plots the reward obtained in each episode for both the reward-predictive model and the SF transfer algorithm and illustrates that the reward-predictive model obtains an optimal policy faster on the second task. This performance improvement can be attributed to the fact that the reward-predictive model builds an internal representation that more closely models how to generalize across different fret-board locations, which is invariant to the scale (i.e, the reward sequence is identical if the agent correctly plays the scale in any of the octaves). Because only equivalences across fret-board locations are modelled, one would also expect a similar performance improvement for any randomly chosen scale. In contrast, SFs encode the visitation frequencies of future (latent) states under a specific policy, a property that changes between the two tasks. Note that this result is not generated because a portion of the note sequence overlaps between the two scales (C-D-E-F-G), otherwise the SF transfer algorithm would exhibit positive transfer on the second scale. Thus, the performance discrepancy in Figure 4.7(b) comes about because SFs and reward-predictive state abstractions model different aspects of an MDP.

4.3 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 [40], a strategy supported by empirical studies in humans [41]. 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 [86, 119, 94], 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 [76, 27]. 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 [95, 78, 75, 89]. 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.

Existing experiments searching for neural and behavioural correlates of the SR [75, 89] have not varied both rewards and transitions, because (unlike the reward-predictive model), the SR is not robust to these changes across environments. Our work motivates the development of targeted experimental designs that would test if human subjects can reuse a latent structure that is present in a set of tasks despite variations in transitions and rewards. For example, one could design a human subject study similar to [41] where participants solve a sequence of grid-world navigation problems, but augment the design to test if subjects reuse a latent structure present in a set of tasks despite variations in transitions and rewards, similar to the task sequence presented in Figure 4.5. As illustrated in Figure 4.6, the specific pattern of generalization across tasks is predicted to vary depending on whether agents use reward-predictive state abstractions or re-use SR abstractions. Thus, our work provides a concrete testable behavioural prediction that would discriminate between our work and existing work.

Offline hippocampal replay has been proposed to reflect sampling from a model to train model-free RL and facilitate planning [72, 69, 93, 5]. Our work provides a predicted amendment to this notion: we suggest that replay may be prioritized in such a way that facilitates the construction of reward-predictive state abstractions. In our work on learning (Section 4.2.3), while the agent is first interacting in a novel MDP, it retains an identity (i.e. un-compressed) state abstraction. Only after sufficiently learning and interacting in this task, the agent can then construct a new state abstraction that can be used for planning in the future. Indeed, for efficient learning and generalization, retaining the identity map while learning is critical; otherwise the agent is likely to create a sub-optimal abstraction that will not generalize. We suggest that the online use of the identity matrix may depend on the highly pattern-separated and conjunctive representations in the hippocampus, whereas the more abstract representations that facilitate generalization and transfer may be cortical [7]. Moreover, we speculate that one way this abstraction could be learned offline would be if, during replay, hippocampal events could be sequentially sampled from regions of the state space that are

most similar in a reward-predictive sense (i.e., those that incur the least reward-predictive loss). In this way, an abstract graph-like structure suitable for future planning could be constructed [91, 43] but further augmented so that it does not depend on temporal adjacency of transitions within the graph itself, but rather in terms of the ability to predict future expected reward sequences – facilitating a deeper form of transfer. This reward-predictive loss function for guiding replay may also shed light on recent studies in rodents demonstrating that replay is biased toward recently received rewards (e.g., food) rather than those that are currently desired (e.g., water) after revaluation, even though behaviour is directed toward the desired one [24]. While this pattern is counter-intuitive from the perspective that replay is used for future planning, it accords with that expected from an algorithm that compresses the state space based on reward-predictive representations, where reward is defined by the previously experienced reward function. Consequently, these representations do not generalize to any arbitrary task and are restricted to variations in transitions, rewards, and optimal policy. This restriction of reward-predictive state abstractions can be observed in Figure 4.5, where a representation learned for the light-dark maze would not be re-used on the coloured maze. Because the presented model demonstrates that generalization across different rewards and transitions is possible, future studies on replay would test subjects for generalization across different tasks instead of only testing for recall of a previously observed task structure.

The Tolman-Eichenbaum machine [117, 118] presents a model for generalization in the hippocampal-entorhinal system [15]. 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, 12, 64, 75, 89, 100, 123, 61] 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 and transition function. A similar effect has also been shown for variations of reward-maximizing state abstractions [3], 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 [70, 12], these methods still rely on initializing learning with a previously learned representation to obtain a performance gain over solving a task from scratch. Universal successor feature approximators (USFA) [21] mitigate the dependency of previously learned SF to a single policy by defining SFs as a function $\boldsymbol{\psi}^{\pi}(s, a; \boldsymbol{w})$, where the weight vector $\boldsymbol{w}$ describes a particular MDP. While this approach only requires learning one SF representation function for a family of policies, this model also assumes fixed transition functions. 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 [12, 70, 40, 41] that transfers SFs, latent transition functions, or latent reward functions to integrate the benefits of each transfer system.

In related work [100], 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 [26, 40], 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 [46, 63], the solution space of possible reward-predictive state abstractions is far more constrained. Prior deep learning models [38] 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 chapter uses LSFMs to compute $l_{\text{predictive}}$, several other methods exist to evaluate reward-predictive state abstractions [28, 36, 35].

4.3.1 Limitations and Future Directions

With the exception of Figure 4.2(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. 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) [68], 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 [17] or visual transfer tasks that can also be used in a human subject study [26].

4.4 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.

Chapter 5

Scaling Reward-Predictive Representations

In this chapter, we focus on scaling reward-predictive representations and the previously presented models to control tasks where state spaces can be uncountably infinite sets. While the theoretical analysis presented in Chapter 3 already provides a foundation for this case, the previously presented representation-learning algorithms are designed for finite MDPs. To remove this restriction and compute reward-predictive representations for arbitrary state spaces, we introduce a clustering algorithm that learns a reward-predictive representation without making any assumptions about the state space. It does so by assuming a regression oracle, a subroutine specifically designed for the task at hand that performs empirical risk minimization (ERM) [113]. Here, ERM is used to find a function to predict one-step rewards, SFs, and to map states to latent states.[1] In practice, this ERM subroutine is implemented using common deep learning techniques [48] that find a neural network by minimizing a loss error objective. However, the presented algorithm can also be combined with any other function-approximation technique.

The presented clustering algorithm extracts a reward-predictive representation from a data set of one-step transitions. Consequently, this transition data set determines across which states the resulting state representation generalizes. To control to what extend the used transition data set informs the learning algorithm about the task, we focus on visual control tasks where the state space can be compressed into a finite set of latent states. The formalism of Block MDPs [32] allows us to conduct controlled experiments and analyze the differences between a learned representation

[1] Although one could alternatively use LAMs and design an algorithm that predicts expected next latent states instead of SFs, we focus on LSFMs.

and the representation that is embedded into the task but hidden from the learning system itself. Our simulations suggest that a training data set needs to contain all possible transitions between these embedded latent task states. In this case, the transition data set implicitly communicates the task's latent structure and the presented clustering algorithm can infer this structure. If this embedded latent structure is only partially expressed in the training data set, then we find that the resulting reward-predictive representation is not suited for accurate reward-sequence prediction. This behaviour is a consequence of the fact that the studied models are designed for arbitrary transition and reward functions. Because these models do not incorporate any additional domain knowledge about the task itself, they cannot generalize to a latent structure not represented in the used training data and the resulting representations have high prediction errors. Note that assuming Block MDPs is not equivalent to assuming finite MDPs: Because the state space is (uncountably) infinite, a test data set may contain states that are not included in the training data. In this case, a learned representation needs to generalize to inputs not observed during training.

This chapter is divided into two sections. Section 5.1 presents a theoretical analysis of the reward-predictive representation-learning algorithm and proves under which conditions the algorithm converges to an approximate reward-predictive representation. Furthermore, if the ERM subroutine generates functions whose prediction errors can be bounded by a small enough value, then the representation learning algorithm converges to a reward-predictive representation that compresses the state space into as few latent states as possible without over-compressing the state space and introducing approximation errors. Learning such a *maximally-compressed* representation is desirable because a smaller latent space results in faster learning at transfer. In Section 5.2 we present the second result, a set of simulations illustrating how re-usable reward-predictive representations can be learned on visual control tasks. In these simulations, the ERM subroutine is implemented by a gradient-descent algorithm that trains a convolutional neural network to predicts rewards, SFs, and latent states. By varying the training data sets, we demonstrate that covering all embedded latent states is important for obtaining an accurate reward-predictive representation. Furthermore, we demonstrate on a visual control task that the learned reward-predictive representations are suitable for re-use across tasks that vary in their transitions and rewards.

5.1 Reward-Predictive Clustering

In Chapter 3, we characterized reward-predictive representations in terms of LSFMs, a theoretical model providing error bounds to test if a state representation is reward-predictive. These error bounds upper-bound the one-step reward prediction errors such that

$$\forall s \in \mathcal{S},\ \forall a \in \mathcal{A},\ \left|\boldsymbol{\phi}_s^\top \boldsymbol{w}_a - \mathbb{E}_p[r(s,a,s')|s,a]\right| \leq \varepsilon_r \tag{5.1}$$

and they bound the SF-prediction errors such that

$$\forall s \in \mathcal{S},\ \forall a \in \mathcal{A},\ \left|\left|\boldsymbol{\phi}_s^\top + \gamma \mathbb{E}_p\left[\boldsymbol{\phi}_{s'}^\top \overline{\boldsymbol{F}} \middle| s,a\right] - \boldsymbol{\phi}_s^\top \boldsymbol{F}_a\right|\right|_2 \leq \varepsilon_\psi. \tag{5.2}$$

Line (5.1) is a re-statement of Equation (3.16)[2] and Line (5.2) is a re-statement of Equation (3.19). If the state space is (uncountably) infinite, then verifying the bounds in Lines (5.1) and (5.2) within finitely many steps may not be possible without making further simplifying assumptions about the reward or transition function. Because the goal is to study tasks with arbitrary transition and reward functions and arbitrary state spaces, we forgo any additional assumptions and instead define an empirical equivalent to Lines (5.1) and (5.2). This empirical equivalent can be evaluated given a finite transition data set $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}_{i=1}^D$. Each data point (s_i, a_i, r_i, s_i') in this data set $\mathcal{D}$ describes a transition from state s_i to state s_i' where action a_i was selected and reward r_i was given. To simplify notation, we denote the set of all states observed in $\mathcal{D}$ with

$$\mathcal{D}_\mathcal{S} = \{s | \exists (s,a,r,s') \in \mathcal{D}\}, \tag{5.3}$$

the set of all state-action combinations contained in $\mathcal{D}$ with

$$\mathcal{D}_{\mathcal{SA}} = \{(s,a) | \exists (s,a,r,s') \in \mathcal{D}\}, \tag{5.4}$$

and the subset of transitions where action a is selected at state s with

$$\mathcal{D}(s,a) = \{(\tilde{s},\tilde{a},r,s') \in \mathcal{D} | \tilde{s} = s,\ \tilde{a} = a\}. \tag{5.5}$$

[2]Here, the term $\mathbb{E}_p[r(s,a,s')|s,a]$ describes the expected reward observed when selecting action a at state s.

Given such a data set, we first replace the expected one-step reward $\mathbb{E}_p[r(s,a,s')|s,a]$ in Line (5.1) with an empirical average across all observed state-action pairs:

$$\overline{r}(s,a) = \frac{1}{|\mathcal{D}(s,a)|} \sum_{(s,a,r,s') \in \mathcal{D}(s,a)} r. \tag{5.6}$$

Similarly, we replace the expected next SF feature $\mathbb{E}_p\left[\boldsymbol{\phi}_{s'}^\top \overline{\boldsymbol{F}} \middle| s,a\right]$ with an empirical average across all observed transitions from state s where action a is selected:

$$\overline{\boldsymbol{\psi}}(s,a;\phi) = \frac{1}{|\mathcal{D}(s,a)|} \sum_{(s,a,r,s') \in \mathcal{D}(s,a)} (1-\gamma)(\phi(s))^\top \overline{\boldsymbol{F}}. \tag{5.7}$$

Because SFs predict the visitation frequencies of future latent states, the SF vector $\overline{\boldsymbol{\psi}}(s,a;\phi)$ depends on the state-representation function $\phi : s \mapsto \boldsymbol{\phi}_s$ that determines the mapping between each state s and each latent feature vector $\boldsymbol{\phi}_s$. The left side of Equation (5.7) does not explicitly state a dependency on the matrix $\overline{\boldsymbol{F}}$, because this matrix $\overline{\boldsymbol{F}}$ is only a function of the state representation ϕ and the transition data set $\mathcal{D}$: By Equation (3.18), $\overline{\boldsymbol{F}} = \frac{1}{|\mathcal{A}|}\{\boldsymbol{F}_a\}_{a\in\mathcal{A}}$ and the matrices $\{\boldsymbol{F}_a\}_{a\in\mathcal{A}}$ are computed by solving a linear least-squares regression problem where the state representation $\phi : s \mapsto \boldsymbol{\phi}_s$ is used as a basis function. Appendix B.4.1 provides a detailed description of this procedure. Using the averages defined in Equations (5.6) and (5.7), we define the empirical equivalent of Lines (5.1) and (5.2) as

$$\forall (s,a,r,s') \in \mathcal{D},\ \left|\boldsymbol{\phi}_s^\top \boldsymbol{w}_a - \overline{r}(s,a)\right| \leq \varepsilon_r \text{ and } \left|\left|\boldsymbol{\phi}_s^\top + \gamma(\overline{\boldsymbol{\psi}}(s,a;\phi))^\top - \boldsymbol{\phi}_s^\top \boldsymbol{F}_a\right|\right|_2 \leq \varepsilon_\psi. \tag{5.8}$$

A representation function $\phi : s \mapsto \boldsymbol{\phi}_s$ satisfying Line (5.8) may only be reward-predictive *on the training data set* $\mathcal{D}$ and may not satisfy the original LSFM conditions stated in Lines (5.1) and (5.2). Consequently, this representation function may not be reward-predictive on the task itself. This distinction parallels the usual distinction between training and test data observed in machine learning [113]: While a representation may perfectly "fit" the training data by satisfying Line (5.8), it is possible that this representation may not perform well on as yet unseen test data and may not be suited for accurate reward-sequence prediction. Nevertheless, in the following sections, we demonstrate the contrary and argue for the practicality of our approach with a set of simulations where a state representation satisfying Line (5.8) is learned and can be used to predict reward sequences

with near zero prediction errors.

5.1.1 Representation Learning With Partition Refinement

Figure 5.1 illustrates how the partition-refinement algorithm obtains a reward-predictive representation on a 4×4 version of the column-world task. In this MDP, the agent can move up, down, left, or right to transition to adjacent grid cells. A reward is given in the green column. Similar to the example presented in Figure 1.1, a reward-predictive representation generalizes across different columns and abstracts away the different rows of the task. At each iteration t, the iterative partition-refinement algorithm constructs a *cluster function*

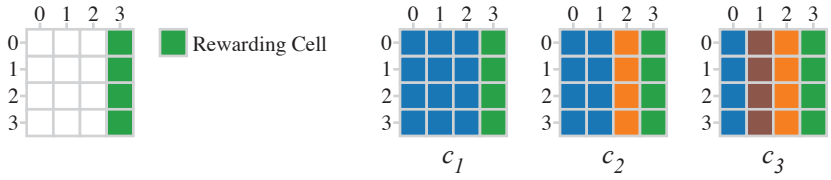
$$c_t : s \mapsto k \tag{5.9}$$

where k is the partition index (an integer) and s is a state that occurs in some transition in $\mathcal{D}$. Figure 5.1(b) illustrates the cluster-function sequence c_1, c_2, c_3 computed for the column-world task. First, the cluster function c_1 is constructed by merging states with equal one-step rewards into the same state partition, resulting in the blue and green partitions. Then, the resulting partitions are refined by computing SFs for all states and dividing partitions such that states with different SFs fall into different partitions. This refinement step is repeated until the cluster function can be used to construct a representation function

$$\phi_t : s \mapsto \boldsymbol{e}_{c_t(s)} \tag{5.10}$$

that satisfies the empirical LSFM conditions outlined in Line (5.8). This representation function ϕ_t is only defined on states s that occur in the transition data set, because the cluster function c_t is defined only for these states. The vector $\boldsymbol{e}_{c_t(s)}$ denotes a one-hot bit vector where entry $c_t(s)$ is set to one and all other entries are set to zero. In Figure 5.1(b), this refinement step is repeated until further refinement would lead to the same cluster function. At this point, the following discussion will show that a reward-predictive representation is obtained. Once this refinement procedure has converged, the resulting cluster function c_t is used to learn a state-representation function

$$\hat{\phi} = \arg \min_{\phi : \mathcal{S} \to \mathbb{R}^n} \sum_{s \in \mathcal{D}_\mathcal{S}} ||\phi(s) - \boldsymbol{e}_{c_t(s)}||_2^2. \tag{5.11}$$



(a) 4×4 Column World MDP (b) Partition Refinement Sequence

Figure 5.1: Partition Refinement on the Column-World MDP. 5.1(a): A 4×4 version of the Column-World MDP where reward is given in the right column and the agent can move either up, down, left, or right to transition to adjacent grid cells. 5.1(b): Partition sequence c_1, c_2, c_3 computed with partition refinement. Grid cells of the same colour belong to the same partitions. Each partition grid map illustrates the partitions obtained at different refinement steps.

This function $\hat{\phi}$ is returned by the reward-predictive representation-learning algorithm.

A cluster function c_t satisfying the empirical LSFM conditions (Line (5.8)) can be trivially obtained by placing each observed state into its own singleton partition. Nonetheless, such a cluster function c_t is not desirable because it does not inform the regression problem in Line (5.11) about how to generalize across different inputs. Consequently, the learned representation function $\hat{\phi}$ is not optimized to generalize across different states and, during evaluation, may not perform well on previously unseen test states. Therefore, we desire the cluster function c_t to construct as few state partitions as possible. We refer to state representations that minimize the number of constructed state partitions as *maximally compressed*. Before discussing how to obtain such a maximally compressed representation in Section 5.1.4, we first present how the cluster algorithm obtains a reward-predictive representation on a finite-state MDP and then generalize the discussion to arbitrary state spaces where function approximation is needed.

One-Step Reward Clustering

The clustering algorithm starts by constructing a cluster function c_1 that merges two states into the same cluster if they are approximately equal in their empirical one-step rewards. Assuming the transition data set contains a transition for all state-action pairs (an assumption we relax in Section 5.1.2), the algorithm first constructs a vector

$$\overline{\boldsymbol{r}}(s) = [\overline{r}(s, a_1), ..., \overline{r}(s, a_n)]^\top \tag{5.12}$$

that concatenates the average rewards $\overline{r}(s, a_i)$ for all actions $a_1, ..., a_n$ for each observed state s. Using these vectors $\overline{\boldsymbol{r}}$, a cluster function c_1 is constructed such that two states s and $\tilde{s}$ belong to

the same cluster if their one-step rewards differ by at most ε_r. Formally, this property of c_1 can be expressed as

$$c_1(s) = c_1(\tilde{s}) \implies ||\overline{\boldsymbol{r}}(s) - \overline{\boldsymbol{r}}(\tilde{s})||_1 \leq \varepsilon_r. \tag{5.13}$$

Suppose we construct a weight vector $\boldsymbol{w}_a$ for each action a such that the ith entry averages $\overline{r}(s,a)$ across all states s associated with the ith partition, then

$$\boldsymbol{w}_a(i) = \frac{1}{|\{s \in \mathcal{D}_{\mathcal{S}} | c_t(s) = i\}|} \sum_{s \in \mathcal{D}_{\mathcal{S}} | c_t(s) = i} \overline{r}(s,a). \tag{5.14}$$

Because Equation (5.14) averages over values that are at most ε_r apart (by construction of c_1 in Line (5.13)), we have that $|\boldsymbol{w}_a(i) - \overline{r}(s,a)| \leq \varepsilon$ and more generally

$$\left| \boldsymbol{e}_{c_t(s)}^\top \boldsymbol{w}_a - \overline{r}(s,a) \right| \leq \varepsilon_r. \tag{5.15}$$

Therefore the reward condition stated in the empirical LSFM conditions (Line (5.8)) is satisfied for a state representation $s \mapsto \boldsymbol{e}_{c_t(s)}$. Because the clustering algorithm is constrained to only refine a partitioning, two states that are separated into two different partitions will not be merged into the same partition at a later iteration. Consequently, each cluster function in the generated sequence of cluster functions $c_1, c_2, ...$ satisfies this reward condition.

Partition Refinement by Successor Feature Clustering

Once the first cluster function c_1 is constructed, the clustering is refined by iteratively dividing existing state partitions. This procedure is similar to the reward clustering presented in the previous paragraph, but involves clustering different *feature descriptor* vectors $\overline{\boldsymbol{\xi}}$. At refinement iteration t, a state representation ϕ_{t-1} is constructed using the cluster function c_{t-1} of the previous iteration such that

$$\forall s \in \mathcal{D}_{\mathcal{S}},\ \phi_{t-1}(s) = \boldsymbol{e}_{c_{t-1}(s)}. \tag{5.16}$$

Then, the feature descriptor vector concatenates the next SF vectors $\overline{\boldsymbol{\psi}}(s,a;\phi_{t-1})$ for all actions and

$$\overline{\boldsymbol{\xi}}(s;c_{t-1}) = \left[\left(\overline{\boldsymbol{\psi}}\left(s,a_1;\phi_{t-1}\right)\right)^\top, ..., \left(\overline{\boldsymbol{\psi}}\left(s,a_n;\phi_{t-1}\right)\right)^\top \right]^\top. \tag{5.17}$$

Similar to the previous discussion, if the state space is finite and each state s can be described by an index, then the vector $\overline{\boldsymbol{\xi}}$ can be computed using Equation (5.7), assuming the transition data set contains a transition for each state-action combination. How to estimate $\overline{\boldsymbol{\xi}}(s; c_{t-1})$ when this assumption does not hold is described in the following section. Using these feature descriptor vectors, the cluster function c_t is constructed such that two states s and $\tilde{s}$ are merged into the same partition if $\overline{\boldsymbol{\xi}}(s; c_{t-1})$ and $\overline{\boldsymbol{\xi}}(\tilde{s}; c_{t-1})$ are almost equal. Formally, the cluster function c_t is constructed such that

$$c_t(s) = c_t(\tilde{s}) \implies \left|\left|\overline{\boldsymbol{\xi}}(s, c_{t-1}) - \overline{\boldsymbol{\xi}}(\tilde{s}, c_{t-1})\right|\right|_1 \leq \varepsilon_\psi. \tag{5.18}$$

With this refinement procedure, a sequence of cluster functions $c_1, ..., c_T$ is constructed until two consecutive cluster functions construct the same state partitions. At this point, the refinement procedure has converged and

$$c_T(s) = c_T(\tilde{s}) \implies \left|\left|\overline{\boldsymbol{\xi}}(s, c_T) - \overline{\boldsymbol{\xi}}(\tilde{s}, c_T)\right|\right|_1 \leq \varepsilon_\psi. \tag{5.19}$$

To arrive at the SF fixed point in Line (5.8) (empirical LSFM conditions), we observe that, for any two states s and $\tilde{s}$ that belong to the same partition, their SF vectors $\overline{\boldsymbol{\psi}}$ are at most ε_ψ apart. This fact follows from the construction of the $\overline{\boldsymbol{\xi}}$ vectors, which concatenate the next SF vectors $\overline{\boldsymbol{\psi}}$ for all actions a (Equation (5.17)). Therefore,

$$\forall a \in \mathcal{A},\ c_T(s) = c_T(\tilde{s}) \implies \left|\left|\overline{\boldsymbol{\psi}}(s, a; \phi_T) - \overline{\boldsymbol{\psi}}(\tilde{s}, a; \phi_T)\right|\right|_1 \leq \varepsilon_\psi. \tag{5.20}$$

We define a set of square matrices $\{\boldsymbol{F}_a\}_{a \in \mathcal{A}}$ such that, for the ith row,

$$\boldsymbol{F}_a(i) = \boldsymbol{e}_i^\top + \gamma \frac{1}{\{s \in \mathcal{D}_\mathcal{S} | c_T(s) = i\}} \sum_{s \in \mathcal{D}_\mathcal{S} | c_T(s) = i} (\overline{\boldsymbol{\psi}}(\tilde{s}, a; \phi_T))^\top. \tag{5.21}$$

By construction of c_T (Line (5.20)), the summation in Equation (5.21) averages over elements that are at most ε_ψ apart (in terms of their L1-norm). Therefore,

$$\left|\left|\overline{\boldsymbol{\psi}}(s,a;\phi_T) - \frac{1}{\{s \in \mathcal{D}_\mathcal{S} | c_T(s) = i\}} \sum_{s \in \mathcal{D}_\mathcal{S} | c_T(s)=i} \overline{\boldsymbol{\psi}}(\tilde{s},a;\phi_T)\right|\right|_1 \leq \varepsilon_\psi \tag{5.22}$$

$$\iff \left|\left|\boldsymbol{e}_i + \gamma\overline{\boldsymbol{\psi}}(s,a;\phi_T) \underbrace{-\boldsymbol{e}_i - \gamma \frac{1}{\{s \in \mathcal{D}_\mathcal{S} | c_T(s) = i\}} \sum_{s \in \mathcal{D}_\mathcal{S} | c_T(s)=i} \overline{\boldsymbol{\psi}}(\tilde{s},a;\phi_T)}_{=-(\boldsymbol{F}_a(i))^\top \text{ by (5.21)}}\right|\right|_1 \leq \varepsilon_\psi \tag{5.23}$$

$$\iff \left|\left|\boldsymbol{e}_i + \gamma\overline{\boldsymbol{\psi}}(s,a;\phi_T) - (\boldsymbol{F}_a(i))^\top\right|\right|_1 \leq \varepsilon_\psi. \tag{5.24}$$

In Line (5.23), the vector difference is scaled down by a factor $\gamma < 1$ and extended with $\boldsymbol{e}_i$ terms. Line (5.24) is obtained by substitution using Equation (5.21). Because $(\boldsymbol{e}_{c_T(s)})^\top \boldsymbol{F}_a = (\boldsymbol{F}_a(i))^\top$, we arrive at an SF fixed-point bound and

$$\left|\left|\boldsymbol{e}_{c_T(s)}^\top + \gamma\big(\overline{\boldsymbol{\psi}}(s,a;\phi_T)\big)^\top - \gamma\boldsymbol{e}_{c_T(s)}^\top \boldsymbol{F}_a\right|\right|_1 \leq \varepsilon_\psi. \tag{5.25}$$

Consequently, if the vector $\overline{\boldsymbol{\psi}}(s,a;\phi_T)$ is of dimension d,

$$\left|\left|\boldsymbol{e}_{c_T(s)}^\top + \gamma\big(\overline{\boldsymbol{\psi}}(s,a;\phi_T)\big)^\top - \gamma\boldsymbol{e}_{c_T(s)}^\top \boldsymbol{F}_a\right|\right|_2 \leq \sqrt{d}\varepsilon_\psi \tag{5.26}$$

and the SF fixed-point bound of the empirical LSFM conditions (Line (5.8)) is satisfied for a re-scaled ε_ψ. Consequently, this partition-refinement procedure has converged to a reward-predictive representation if two consecutive refinement steps do not change the state partitions. This result is formally proven in Theorem 5 in Appendix A.3.

The outlined refinement procedure can always arrive at a cluster function c_T satisfying the empirical LSFM conditions (Line (5.8)) by placing each observed state into a singleton partition. Because the reward-predictive representation algorithm only refines a state partition by separating states into two different partitions but never merging two states into the same partition, no two cluster functions in the sequence of cluster functions $c_1, ..., c_T$ can repeat. Hence, the algorithm cannot loop and the cluster function sequence $c_1, ..., c_T$ is always finite. (See also the proof of Theorem 5.)

5.1.2 Function Approximation

The previously presented partition-refinement procedure computes the vectors $\overline{\boldsymbol{r}}(s)$ and $\overline{\boldsymbol{\xi}}(s, c_{t-1})$ by first estimating the empirical average reward $\overline{r}$ and the empirical average next SF vectors $\overline{\boldsymbol{\psi}}$ at each observed state $s \in \mathcal{D}_{\text{state}}$ for each action $a \in \mathcal{A}$. If the state space is finite and the transition data set contains each state-action combination, then $\overline{r}(s,a)$ and $\overline{\boldsymbol{\psi}}(s,a)$ can be computed using Equations (5.6) and (5.7). If the state space is (uncountably) infinite, then using Equations (5.6) and (5.7) may not be possible. Moreover, each state may occur only exactly once in the transition data set and may only be paired with one action. In this case, the representation-learning algorithm uses empirical risk minimization to approximate $\overline{r}$ and $\overline{\boldsymbol{\psi}}$ and predict $\overline{r}(s,a)$ and $\overline{\boldsymbol{\psi}}(s,a)$ for state-action combinations not contained in the transition data set.

Figure 5.2 illustrates when $\overline{r}(s,a)$ and $\overline{\boldsymbol{\psi}}(s,a)$ are predicted using a learned model. In this example, states are described as positions in $\mathbb{R}^2$ and all points lying in the shaded square belong to the same partition and latent state. Specifically, selecting action a from within the grey square results in a transition to the right and a reward of zero, while selecting action b results in a transition to the top and a reward of one. We assume that the transition data set only contains the two transitions indicated by the blue arrows. In this case, we have $\overline{r}(p,a) = 0$ and $\overline{r}(q,b) = 1$, because (p,a) and (q,a) are state-action combinations contained in the transition data set and a reward of zero and one was given, respectively. To estimate $\overline{r}$ for the missing state-action combinations (p,b) and (q,a), we approximate the function $\overline{r}$ with a function $\hat{g}_r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and then use $\hat{g}_r$ to predict $\overline{r}$ for the these missing state-action combinations. Such a function $\hat{g}_r$ can be found by minimizing the least-squared-error objective $\sum_{(s,a,r,s') \in \mathcal{D}} (g(s,a) - \overline{r}(s,a))^2$ that evaluates the difference between $\overline{r}(s,a)$ and the predicted value $g(s,a)$. In practice, each state-action pair (s,a) may occur only once in the transition data set and the sampled one-step reward $r = \overline{r}(s,a)$ in this case. Consequently, we focus on minimizing the difference between the predictions $\hat{g}_r(s,a)$ and the sampled rewards to find $\hat{g}_r$:

$$\hat{g}_r \in \arg\min_{g \in \mathcal{G}_r} \sum_{(s,a,r,s') \in \mathcal{D}} (g(s,a) - r)^2. \tag{5.27}$$

In this optimization problem, the function space $\mathcal{G}_r$ is searched to find $\hat{g}_r$. To ensure that $\hat{g}_r$ accurately approximates $\overline{r}$, an expert designer can implicitly inject domain knowledge into this optimization process by constraining $\mathcal{G}_r$. For example, if neural networks are used, these constraints can be implemented by picking a specific neural network architectures. If the resulting function

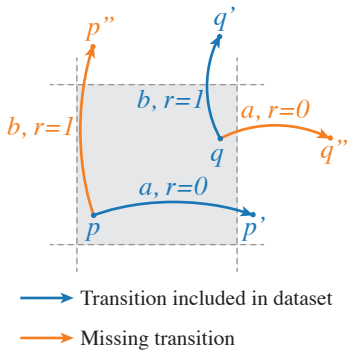


Figure 5.2: Function approximation is needed to predict $\overline{r}$ and $\overline{\boldsymbol{\psi}}$ for state-action combinations not observed in the transition data set. In this example, the state space consists of points in $\mathbb{R}^2$ and the action space consists of actions a and b. We assume that a maximally compressed reward-predictive representation merges all points in the grey square into one latent state. Selecting the action a from within the grey square results in a transition to the right and generates a reward of 0. Selecting the action b from within the grey square results in a transition to the top and generates a reward of 1. If the data set only contains the two transitions indicated by the blue arrows and the transitions indicated by the orange arrows are missing, then function approximation is used to predict $\overline{r}$ and $\overline{\boldsymbol{\psi}}$ for the missing state and action combinations (p, b) and (q, a). These function approximators need to be constrained such that they output the same one-step rewards and SF vectors for points that fall within the shaded square.

$\hat{g}_r$ accurately approximates $\overline{r}$, then we can use $\hat{g}_r$ to predict $\overline{r}(s, a)$ for state-action pairs (s, a) not included in the transition data set. These predictions can then be used to predict the vectors $\overline{\boldsymbol{r}}(s)$.

The SF vectors $\overline{\boldsymbol{\psi}}(s, a)$ for missing state-action combinations (s, a) are estimated using the same rationale. Here, a function $\hat{g}_\psi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^n$ is found to predict SF vectors as accurately as possible. This optimization process first estimates the matrices $\{\boldsymbol{F}_a\}_{a\in\mathcal{A}}$ and their average matrix $\overline{\boldsymbol{F}} = \frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}} \boldsymbol{F}_a$ using linear least-squares regression where ϕ_t is used a basis function (see also Appendix B.4.1). Similar to the previous discussion, we do not use the vector $\overline{\boldsymbol{\psi}}(s, a; \phi_t)$ to access prediction errors and instead compute the difference between the predicted vector and the SF vector sampled at the next state s', the vector $\boldsymbol{e}_{c_t(s')}^\top \overline{\boldsymbol{F}}$. In practice, a state-action pair (s, a) occurs only once in the transition dataset and in this case $\overline{\boldsymbol{\psi}}(s, a; \phi_t) = \boldsymbol{e}_{c_t(s')}^\top \overline{\boldsymbol{F}}$ if $(s, a, r, s') \in \mathcal{D}$. Formally, the function $\hat{g}_\psi$ can be minimizing the empirical risk [113]:

$$\hat{g}_\psi \in \arg\min_{g\in\mathcal{G}_\psi} \sum_{(s,a,r,s')\in\mathcal{D}} ||(g(s, a))^\top - \boldsymbol{e}_{c_T(s')}^\top \overline{\boldsymbol{F}}||_2^2. \tag{5.28}$$

Similar to approximating one-step rewards, an expert designer can constrain the searched function space $\mathcal{G}_\psi$ to ensure that $\hat{g}_\psi$ accurately approximates $\overline{\boldsymbol{\psi}}$. For example, if neural networks are used,

these constraints can be implemented by picking specific neural network architectures. If the resulting function $\hat{g}_\psi$ accurately predicts the next SF vectors, then $\hat{g}_\psi(s,a) \approx \overline{\boldsymbol{\psi}}(s,a;\phi)$. In this case, the function $\hat{g}_\psi$ can be used to predict the SF vectors for state-action pairs (s,a) not contained in the transition data set. These predictions can then be used to predict the feature descriptor vectors $\overline{\boldsymbol{\xi}}(s;c)$.

In the subsequent simulations, we demonstrate how deep learning techniques can be used to solve the optimization problem in Equation (5.27), where choosing particular network architectures and hyper-parameters constrain the function space $\mathcal{G}$ and lead to accurate approximations of $\overline{r}$. Note that Figure 5.2 only provides a minimal example illustrating the need for function approximation. In practice, especially when deep learning techniques are used, many more transitions are needed to find a function that accurately predicts $\overline{r}$ or $\overline{\boldsymbol{\psi}}$.

5.1.3 Iterative Reward-Predictive Representation Learning Algorithm

The presented clustering operations can be combined with function approximation to design an iterative reward-predictive representation algorithm. Algorithm 3 summarizes this procedure.

Algorithm 3 Iterative Reward-Predictive Representation Learning

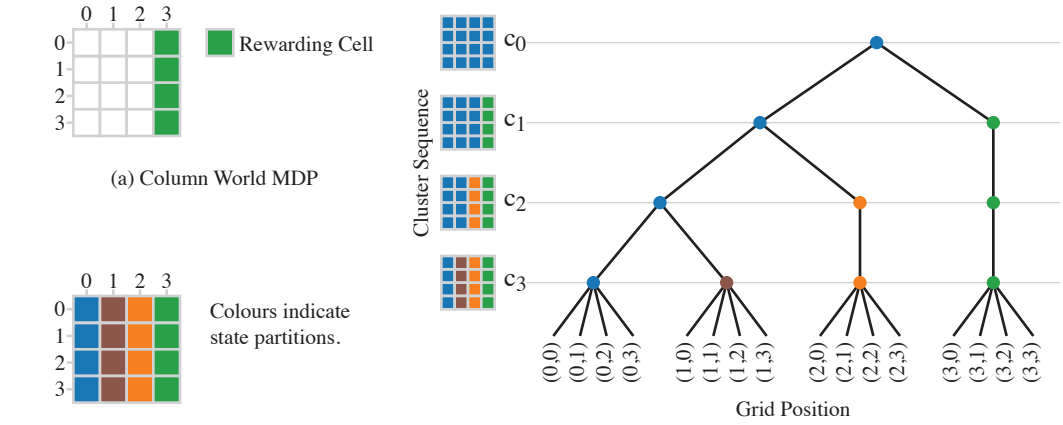
1: **Input:** A transition data set $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}_{i=1}^n$, $\hat{\varepsilon}_r, \hat{\varepsilon}_\psi > 0$.
2: $\hat{g}_r = \arg\min_{g \in \mathcal{G}_r} \sum_{(s,a,r,s') \in \mathcal{D}} (g(s,a) - r)^2$ ▷ Eq. (5.27)
3: Construct $\hat{\boldsymbol{r}}(s) = [\hat{g}_r(s,a_1), ..., \hat{g}_r(s,a_n)]^\top$, $\forall s \in \mathcal{D}_\mathcal{S}$
4: Construct c_1 such that $c_1(s) = c_1(\tilde{s}) \implies ||\hat{\boldsymbol{r}}(s) - \hat{\boldsymbol{r}}(\tilde{s})||_1 \leq \hat{\varepsilon}_r$. ▷ Eq. (5.13)
5: **for** $t = 2, ..., T$, until consecutive cluster functions construct the same partitions **do**
6: Compute LSFM matrix $\overline{\boldsymbol{F}} = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \boldsymbol{F}_a$ for $\phi_{t-1} : s \mapsto \boldsymbol{e}_{c_{t-1}(s)}$. ▷ App. B.4.1
7: $\hat{g}_\psi = \arg\min_{g \in \mathcal{G}_\psi} \sum_{(s,a,r,s') \in \mathcal{D}} ||g(s,a) - \boldsymbol{e}_{c_{t-1}(s')}^\top \overline{\boldsymbol{F}}||_2^2$ ▷ Eq. (5.28)
8: Construct $\hat{\boldsymbol{\xi}}(s, c_{t-1}) = \left[(\hat{g}_\psi(s,a_1))^\top, ..., (\hat{g}_\psi(s,a_n))^\top\right]^\top$, $\forall s \in \mathcal{D}_\mathcal{S}$
9: Construct $\hat{c}_t$ such that $\hat{c}_t(s) = \hat{c}_t(\tilde{s}) \implies ||\hat{\boldsymbol{\xi}}(s, c_{t-1}) - \hat{\boldsymbol{\xi}}(\tilde{s}, c_{t-1})||_1 \leq \hat{\varepsilon}_\psi$ ▷ Eq. (5.18)
10: Construct c_t by intersecting c_{t-1} with $\hat{c}_t$.
11: $\hat{\phi} = \arg\min_\phi \sum_{s \in \mathcal{D}_\mathcal{S}} ||\phi(s) - \boldsymbol{e}_{c_T(s)}||_2^2$. ▷ Eq. (5.11)
12: **return** $\hat{\phi}$

Algorithm 3 estimates a state representation to satisfy the empirical LSFM conditions (Line (5.8)) for ε_r and ε_ψ. To account for prediction errors when approximating one-step rewards or SFs, smaller clustering thresholds $\tilde{\varepsilon}_r$ and $\tilde{\varepsilon}_\psi$ are chosen. Algorithm 3 first estimates the function $\hat{g}_r$ to predict one-step rewards (Line 2) and then uses predictions made by $\hat{g}_r$ to construct the vectors $\hat{\boldsymbol{r}}(s)$, which predicts $\overline{\boldsymbol{r}}(s)$ for each state observation (Line 3). Subsequently, the reward clustering c_1 is

constructed in Line 8 using a threshold $\hat{\varepsilon}_r < \varepsilon_r$. The threshold $\hat{\varepsilon}_r$ is smaller than the required ε_r to account for prediction errors in $\hat{\boldsymbol{r}}(s)$. This reward clustering is then iteratively refined by first estimating the matrix $\overline{\boldsymbol{F}}$ (Line 6). Subsequently, this matrix is used to find the function $\hat{g}_\psi$ that approximates SF vectors (Line 7). The predictions made by the function $\hat{g}_\psi$ are then used to construct the vectors $\hat{\boldsymbol{\xi}}(s, c_{t-1})$, which predict the feature descriptor vectors $\overline{\boldsymbol{\xi}}(s, c_{t-1})$ for each observed state s. In Line 10, these feature-descriptor vectors are used to construct the intermediate cluster function $\hat{c}_t$. Here, the used threshold $\tilde{\varepsilon}_\psi$ is smaller than the required ε_ψ to account for approximation errors in $\hat{\boldsymbol{\xi}}(s, c_{t-1})$. To constrain the algorithm to only separate partitions, the next cluster function c_t is obtained by intersecting the partitions of the intermediate clustering $\hat{c}_t$ with the clustering of the previous iteration. This loop continues until two consecutive cluster functions construct the same state partitions (Line 5). The resulting cluster function c_T is then used to estimate a state-representation function that approximates the map $s \mapsto \boldsymbol{e}_{c_T(s)}$ as accurately as possible (Line 11).

5.1.4 Convergence to Maximally Compressed Representations

As discussed in Section 5.1.1, the goal here is to find a reward-predictive state representation that generalizes across different states where possible by constructing as few latent states as possible. To analyze under which conditions Algorithm 3 converges to such a maximally-compressed representation, we observe that the computed cluster-function sequence $c_1, ..., c_T$ encodes a hierarchical clustering of observed states. This hierarchical structure is illustrated in Figure 5.3 for a 4×4 version of the column world task. In this task, a reward-predictive representation only generalizes across the four columns and compresses the 16 different grid cells into four latent states for $\varepsilon_\psi = \varepsilon_r = 0$ (Figure 5.3(b)). The partition-refinement procedure implemented by Algorithm 3 discovers this reward-predictive state representation by iteratively expanding the depth of the tree structure depicted in Figure 5.3(c). Algorithm 3 does not need to compute c_0 explicitly because the clustering c_1 can be directly computed by clustering the states by one-step rewards, as described in Section 5.1.1. (We include the initial cluster c_0 only to complete the tree structure for illustration purposes.) By the argument presented in Section 5.1.1, this refinement procedure stops in the column-world example once a cluster function is reached that generalizes only across columns. Because the computed cluster function c_3 already encodes a reward-predictive representation in this example, Algorithm 3



(a) Column World MDP

(b) Optimal Reward-Predictive Representation

(c) Cluster Tree Constructed by Refinement Algorithm

Figure 5.3: The cluster-function sequence computed by the iterative reward-predictive representation learning algorithm encodes a hierarchical clustering of states. 5.3(a): The 4×4 version of the Column-World MDP repeated from Figure 5.1(a) for illustration. 5.3(b): For $\varepsilon_\psi = \varepsilon_r = 0$, a maximally-compressed reward-predictive state representation generalizes across different columns, as indicated by the colouring. 5.3(c): The sequence of cluster functions computed by Algorithm 3 for the 4×4 Column-World MDP encodes a tree structure.

would not continue to further refine any state partitions and expand the cluster tree. In this section, we present a formal analysis of the assumptions under which Algorithm 3 discovers such a maximally-compressed reward-predictive representation.

In Algorithm 3, the number of constructed latent states depends on

1. the ability to find accurate approximations of one-step rewards and SF vectors and
2. the ability to cluster a set of vectors into as few clusters as possible.

We first focus on the second condition and assume access to a function approximation subroutine that computes a function whose prediction errors are always bounded.

Assumption 3 (ε-Perfect)**.** *The functions $\hat{g}_r$ and $\hat{g}_\psi$ computed in lines 2 and 7 are such that there exists small $\tilde{\varepsilon}_r, \tilde{\varepsilon}_\psi > 0$ such that for all observed states s, all actions a, and each iteration t,*

$$\left|\hat{g}_r(s,a) - \overline{r}(s,a)\right| \leq \tilde{\varepsilon}_r \text{ and } \left|\left|\hat{g}_\psi(s,a) - \overline{\boldsymbol{\psi}}(s,a;\phi_t)\right|\right|_2^2 \leq \tilde{\varepsilon}_\psi. \tag{5.29}$$

In practice, one cannot know if Assumption 3 holds nor how accurate predictions made by $\hat{g}_r$ and $\hat{g}_\psi$ are, a principle that is described in ERM [16]. However, recent advances in deep learning [16] has found that increasing the capacity of neural networks often makes it is possible to interpolate the

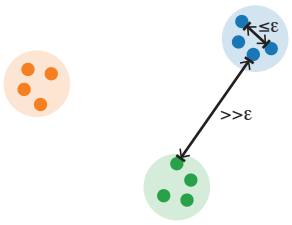


Figure 5.4: A set of points can be clustered in polynomial time if the inter-cluster distance of an optimal clustering is larger than the used cluster threshold ε.

training data and still perform almost perfectly on independently sampled test data. Consequently, it may be possible to satisfy Assumption 3 by using large enough neural network architectures, assuming a diverse enough training data set. In the following section, we demonstrate on two visual control tasks that it is in fact possible to find $\hat{g}_r$ and $\hat{g}_\psi$ that (almost) satisfy this ε-perfect assumption.

To obtain a maximally-compressed state representation, the cluster steps in Lines 4 and 9 must construct as few clusters as possible. Clustering a set of vectors such that vectors of the same clusters are at most ε apart while also minimizing the number of used clusters is an instantiation of finding an optimal hierarchical clustering, a computational problem that is NP-hard [60]. Alternatively, one could constrain the maximum number of clusters found at each iteration, for example by using the k-Means clustering algorithm. However, finding k clusters while minimizing the distance between any two vectors of the same cluster is also NP-hard [71]. In general, by Kleinberg's impossibility theorem [57] an algorithm that finds an arbitrary clustering for vectors of arbitrary scale while also minimizing the distance between points of the same cluster does not exist. Therefore we make an assumption about the used cluster thresholds. For example, suppose a set of points (or vectors) is concentrated into a number of different clusters as illustrated in Figure 5.4. These points can be clustered into the three coloured partitions in polynomial time by merging states that are at most ε apart into the same cluster. This strategy works if the inter-cluster distance is significantly bigger than the used threshold ε.

To ensure that each clustering operation of Algorithm 3 falls into the scenario illustrated in Figure (5.4), we make the following assumption about ε_r and ε_ψ.

Assumption 4 (Separability)**.** *For any arbitrary maximally-compressed reward-predictive cluster*

function c^*, the values $\hat{\varepsilon}_r, \hat{\varepsilon}_\psi > 0$ are assumed to be small enough such that

$$c^*(s) \neq c^*(\tilde{s}) \implies ||\hat{\boldsymbol{r}}(s) - \hat{\boldsymbol{r}}(\tilde{s})||_1 > \hat{\varepsilon}_r \text{ or } ||\hat{\boldsymbol{\xi}}(s, c^*) - \hat{\boldsymbol{\xi}}(\tilde{s}, c^*)||_1 > \hat{\varepsilon}_\psi. \tag{5.30}$$

Assumption 4 can always be satisfied if the cluster function c^* only merges states with either the same one-step rewards or the same SFs. This case is equivalent to setting $\varepsilon_r = \varepsilon_\psi = 0$ and for any two states s and $\tilde{s}$ that belong to the same partition

$$\overline{\boldsymbol{r}}(s) = \overline{\boldsymbol{r}}(\tilde{s}) \text{ and } \overline{\boldsymbol{\xi}}(s; c^*) = \overline{\boldsymbol{\xi}}(\tilde{s}; c^*). \tag{5.31}$$

Because the transition data set contains a finite number of states, the set of different $\overline{\boldsymbol{r}}$ and $\overline{\boldsymbol{\xi}}$ vectors is finite and one can always pick a positive $\tilde{\varepsilon}_r$ and $\tilde{\varepsilon}_\psi$ to satisfy the separability assumption.

Under these two assumptions, the following theorem can be proven guaranteeing that Algorithm 3 converges to a maximally compressed reward-predictive cluster function.

Theorem 5 (Convergence)**.** *Under Assumption 3, Algorithm 3 converges to a reward-predictive state representation. Additionally, if the hyper-parameters $\hat{\varepsilon}_r$ and $\hat{\varepsilon}_\psi$ are picked small enough to satisfy the Assumption 4, then Algorithm 3 converges to a maximally-compressed reward-predictive state representation.*

Theorem 5 formally proves what is suggested by the column-world example in Figure 5.3: If one-step rewards and SFs can be approximated up to a small enough error, then the iterative reward-predictive representation-learning algorithm converges to a maximally-compressed reward-predictive representation. Furthermore, this representation does not over-compress the state space of a given task and does not merge different columns in the Column-World example. Such an over-compressed representation could still be learned for higher ε_r and ε_ψ settings, as illustrated in Figure 5.5. Yet these representations are not guaranteed by Theorem 5 to construct the fewest possible number of latent states. Appendix A.3 presents a formal proof of Theorem 5

Figure 5.5 illustrates reward-predictive representations obtained by the proposed clustering algorithm on the puddle-world task previously used in Figure 3.6. In this task, the identity map is an optimal reward-predictive representation because retaining the exact grid position is important for predicting reward sequences accurately (Section 3.3 discusses this example). Figure 1 illustrates the

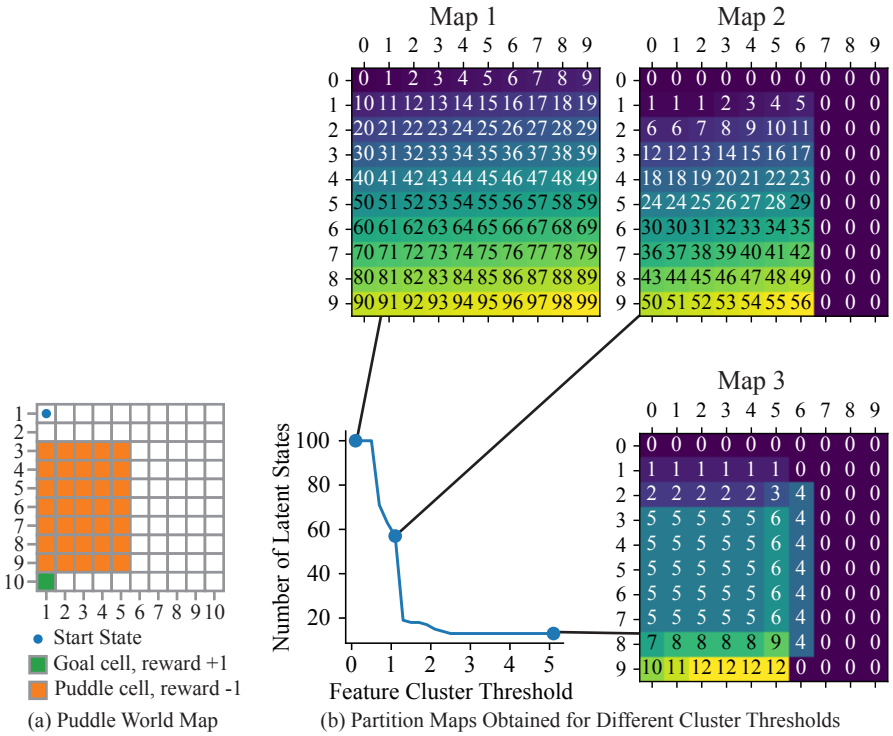


Figure 5.5: Approximate reward-predictive representations found in the puddle-world task. 5.5(a): Map of the puddle-world MDP previously presented in Figure 3.6 and repeated here for illustration. The agent can move up, down, left, or right to adjacent grid cells and either receive a reward of $+1$ for entering the goal cell or a reward of -1 for entering the puddle. Transitions are non-deterministic because choosing an action leads to not moving to another grid cell with 5% chance. 5.5(c): The number of found latent states decreases as the feature cluster threshold ε_ψ is increased. Map 1, Map 2, and Map 3 illustrate the found state partitions for different threshold settings.

partition maps found for different feature cluster threshold settings ε_ψ. As the cluster threshold increases, the number of latent states decreases and the found reward-predictive representations start out with the identity map (Map 1) and then become more "over-compressed" until only a one-step reward clustering is obtained (Map 3). Map 2 illustrates a representation found for an in-between threshold where partition map is refined for most areas of the grid, but there exists still a large state partition with index 0. This partition map highlights one key difference between the proposed clustering algorithm and the gradient descent algorithm used in Section 3.3: While the gradient descent algorithm builds tiles of roughly the same size as illustrated in Figure 3.6, the partition-refinement procedure may at first only focus on refining some portions of the state space and leave a larger state subset in the same partition for multiple iterations.

5.2 Generalization With Neural Networks

In this section, we demonstrate how the presented clustering algorithm can be combined with deep learning techniques and applied to visual control tasks. The first control task is an extension of the puddle-world task presented first in Section 3.3. We focus on this task to illustrate that a transition data set needs to be informative about the task's embedded grid structure. If a data set does not completely cover this embedded structure, then the learned reward-predictive representation is not suitable for accurate reward-sequence prediction. Subsequently, we present an image version of the combination-lock problem previously presented in Section 3.5.1. With this task, we illustrate that neural networks that model a reward-predictive representation can be re-used across tasks with different transitions and rewards. Further, at transfer, such a previously learned representation network is re-used without any modifications to accelerate learning of an optimal policy. These results extend the previously presented discussion on transfer and highlight that reward-predictive representations also generalize across tasks where state spaces are uncountably infinite and function approximation is used.

5.2.1 Latent State Coverage Determines Predictive Performance

The transition data set used for representation learning influences how the resulting representation generalizes across states. We study this dependency using an image version of the puddle-world task. This task is a deterministic version of the puddle-world task presented in Figure 3.6(a), but instead

of providing an agent with a state index, this index is converted into an image. This conversion is non-deterministic and the chance of observing the exact same state image twice is zero. Figure 5.6(a) outlines this procedure. First, the 10×10 grid (left schematic in Figure 5.6(a)) is mapped onto the square area depicted in the centre of Figure 5.6(a). This square area is the set of points

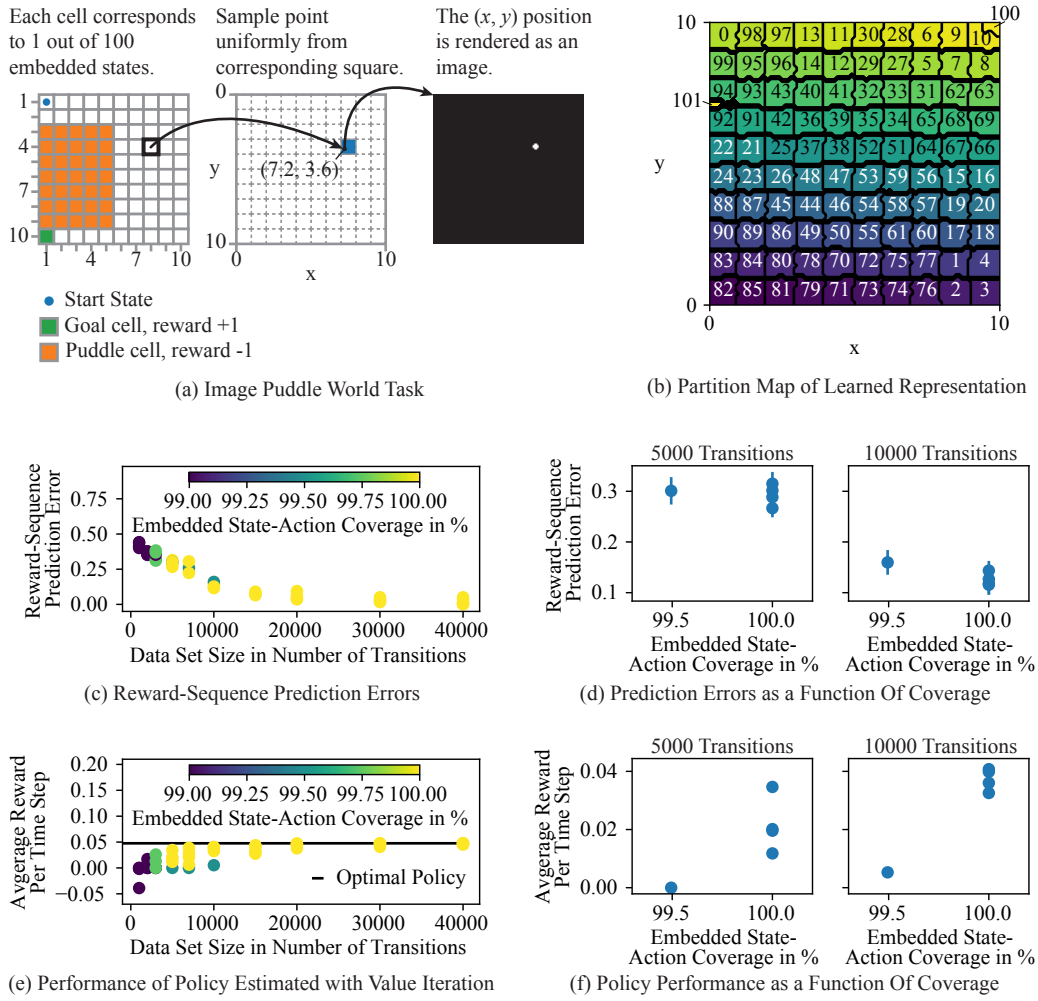
$$\{(x, y) \in \mathbb{R}^2 | 0 < x < 10, 0 < y < 10\}. \tag{5.32}$$

Each grid cell is mapped to one of the square patches within this area (as indicated by the centre grid schematic in Figure 5.6(a)). For example, the grid cell $(1, 1)$ is mapped to the set $\{(x, y) \in \mathbb{R}^2 | 0 < x < 1, 0 < y < 1\}$, the grid cell $(2, 1)$ is mapped to the set $\{(x, y) \in \mathbb{R}^2 | 1 < x < 2, 0 < y < 1\}$, and so forth. To render an image, a grid cell—for example the cell at $(8, 4)$—is first mapped to its corresponding square patch (the blue square in Figure 5.6(a)). Then, a point is sampled uniformly at random from this patch and a white dot is rendered at this location in a black 100×100 image. Because the set of points that can be sampled is uncountably infinite, the number of different images that correspond to one grid cell is uncountably infinite.[3]

The puddle world's grid structure can be found by the reward-predictive clustering algorithm, as illustrated in Figure 5.6(b). Here, the ERM subroutine is implemented by a gradient descent algorithm to approximate one-step rewards, SFs, and the resulting reward-predictive representation with convolutional neural networks [48]. (Table B.9 provides a detailed description of the implementation and used hyper-parameter.) Figure 5.6(b) shows a contour plot of how different (x, y) positions are matched with different latent states. For this plot, a set of (x, y) positions is first rendered into images and these images were then fed through the learned convolutional neural network. This representation network is constrained to output a probability vector over the learned latent states. The contour plot in Figure 5.6(b) visualizes the latent state with the highest probability value for each (x, y) position.

The only domain knowledge the clustering algorithm receives is to use a 2-layer convolutional neural network to process a 100×100 pixel image. Otherwise, the algorithm only observes a transition data set and does not receive any information about the grid structure of the task or how many latent states are present. Figure 5.6(b) illustrates how closely the underlying grid structure is

[3] We also use anti-aliasing to render each image such that two close by points are not rendered as the exact same image.



(a) Image Puddle World Task

(b) Partition Map of Learned Representation

(c) Reward-Sequence Prediction Errors

(d) Prediction Errors as a Function Of Coverage

(e) Performance of Policy Estimated with Value Iteration

(f) Policy Performance as a Function Of Coverage

Figure 5.6: Low embedded state coverage decreases predictive performance in the image puddle-world task. 5.6(a): The image puddle-world task extends the puddle-world task (Figure 3.6(a)) by rendering grid positions as images. Grid positions are rendered by first mapping each grid cell to a square area. Then, an (x, y) position is sampled from this area and rendered as a white dot. Because the image rendering pipeline is non-deterministic, transitions appear non-deterministic. 5.6(b): Partition plot illustrating how different dot positions are associated with latent states by the learned reward-predictive representation network. 5.6(c): Reward-sequence prediction errors for each learned model and training data size. Prediction errors are averaged over 200 time-step reward sequences. 5.6(e): Using each learned model, the policy optimal with respect to this model is evaluated 20 times and the average reward per time step is recorded. 5.6(c), 5.6(e): The colouring plots if the embedded state-action space was covered by the used training data set. The dots and error bars plot averages and standard error of measure across 20 evaluation repeats. 5.6(d), 5.6(f): Reward-sequence prediction errors and policy performance plotted as a function of embedded state-action space coverage for two different data set sizes.

approximated. Approximation errors still corrupt the learned representation to some extent, resulting in discovering 102 latent states instead of 100. For example, the top left corner of the contour plot in Figure 5.6(b) shows that the partitions corresponding to latent state 10 and latent state 100 were incorrectly separated. This separation comes about because the used ERM subroutine is not satisfying the ε-perfect assumption for all possible inputs (states-action combinations). To accommodate approximation errors to some extent, we found it useful to exclude state partitions that contain only very few state observations in relation to the training data set size. For example, the simulation used in Figure 5.6(b) excluded any state partition that contained 20 or fewer state observations. At the end of training, this rule led to withholding 123 transitions out of 40000 transitions in total. This method of excluding state partitions is a heuristic and makes the assumption that the number of state observations that correspond to each latent state is roughly equal—an assumption that is satisfied by using uniform random action selection to generate transitions from the image puddle-world task.

To study the dependence between the learned reward-predictive representation and the used training data, we repeatedly ran the presented clustering algorithm on data sets of varying sizes.[4] By varying the size of the collected data set, we can control to what extent the task's embedded grid structure is covered. For each simulation, we evaluated the learned representation network's suitability for accurate reward-sequence prediction and computation of an optimal policy. The policy optimal with respect to the learned representation is computed by first estimating the transition and reward tables between the different latent states using the last cluster function c_T generated by the clustering algorithm. These transition and reward tables are then fed into the value-iteration algorithm [106, Chapter 4.4] to obtain a policy. This policy is then evaluated through multiple simulation runs and the average reward per time step is recorded. Figures 5.6(c) and 5.6(e) plot the obtained evaluation results for different data-set sizes. As the data-set size increases, the resulting representation's performance improves. For data sets with 30000 or more transitions, the learned model is (almost) optimal, because reward-sequence prediction errors are near zero and the policy obtained through value iteration achieves an almost optimal reward level. This trend is not surprising, because providing more data typically improves a deep learning system's performance.

Because a uniform distribution is used for sampling the state images, the probability of observing

[4]Transition data sets were collected by sampling multiple trajectories using uniformly random action selection. Then, a prefix of the first trajectory is truncated to obtain a transition data set of a specific size.

the same image twice is zero. Consequently, the training data differs from the test data. Because the learned model's performance is near optimal for large enough data sets, the learned representation can associate a previously unseen image with the correct latent state, leading to accurate predictions. Therefore, the learned representation correctly generalizes to previously unseen data.

The colouring of the plotted data points indicates to what extent the embedded latent state-action space is covered when generating the training data. A coverage of 100% means that all four actions at all grid cells were visited and the latent grid structure was fully explored. A coverage of less than 100% indicates that a certain fraction of latent state-action combinations were not visited and some portion of the grid structure was not fully explored. This colouring in Figures 5.6(c) and 5.6(e) indicates that only data sets with 100% latent state-action coverage lead to near optimal models. Furthermore, the policy performance plot in Figure 5.6(e) indicates that for data-set sizes between 5000 and 10000 transitions incomplete coverage leads to a worse performing model, even for a fixed data set size. This trend is also present in Figure 5.6(c). Partial latent-state-action coverage decreases performance of the learned reward-predictive representation because the clustering algorithm does not generalize to unseen latent states. Figures 5.6(d) and 5.6(f) further highlight this dependency and illustrate that using training data that does not fully cover the task's grid structure results in decreased performance. This behaviour is a consequence of the algorithm's generic design: Because the cluster algorithm is designed to process data of tasks with arbitrary reward or transition functions, the learned models do not encode an inductive bias that enables generalization to unseen transitions or rewards.

For example, if one grid cell is not covered by the training data, then the algorithm is not provided with any information about rewards at this grid cell or transitions to and from this grid cell. Making any predictions about such a missing grid cell means knowing how to generalize any learned transitions or rewards to this missing grid cell. In this context, knowing how to generalize means making additional assumptions about the transition or reward function that is not explained by the training data. Such an assumption could be implemented by providing an inductive bias to expect a grid world structure, for example. Consequently, any agent that does not make any assumptions about what transitions or rewards to expect cannot make an informed guess about such a missing grid cell. In fact, the agent is limited to inferences that stem only from the observed training data. The agent may not even infer that such a missing grid cell exists, because the agent has not observed any corresponding data. This inability of generalizing to unseen latent states

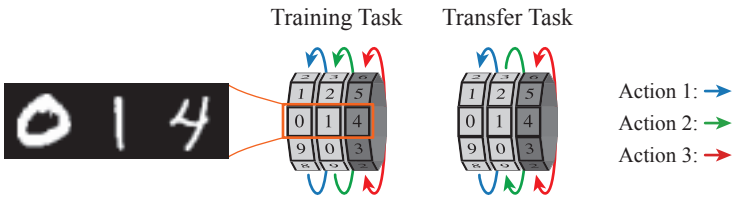


Figure 5.7: MNIST Combination-Lock Task. In both the training and transfer task, the agent can choose from one of three actions to rotate one of the three dials by one number. Each dial has ten sides and any number combination is rendered using the MNIST image data set [62], as illustrated by the image on the left. In the training task, the right dial (dark grey dial) spins at random at every time step and has no effect on obtaining reward. Here, to obtain reward, the agent has to use actions 1 and 2 to rotate the left and centre dials so both show the digit nine. While the right dial (dark grey dial) also spins at random in the transfer task, the transition and reward functions are different to the training task. Here, the rotation direction of the centre dial is inverted and the agent has to rotate the dial into any combination that starts with eight and two.

or actions is a consequence of designing an agent that assumes any generic transition and reward function.

5.2.2 Generalization Across Tasks With Neural Networks

While the presented model does not generalize to unseen latent states within a task, a learned reward-predictive representation can be efficiently re-used in previously unseen tasks, assuming the modelled latent structure is observed. In this section, we present simulations to demonstrate this form of transfer with a neural network. These results extend the transfer discussion in Chapters 3 and 4 to visual control tasks where function approximation is needed.

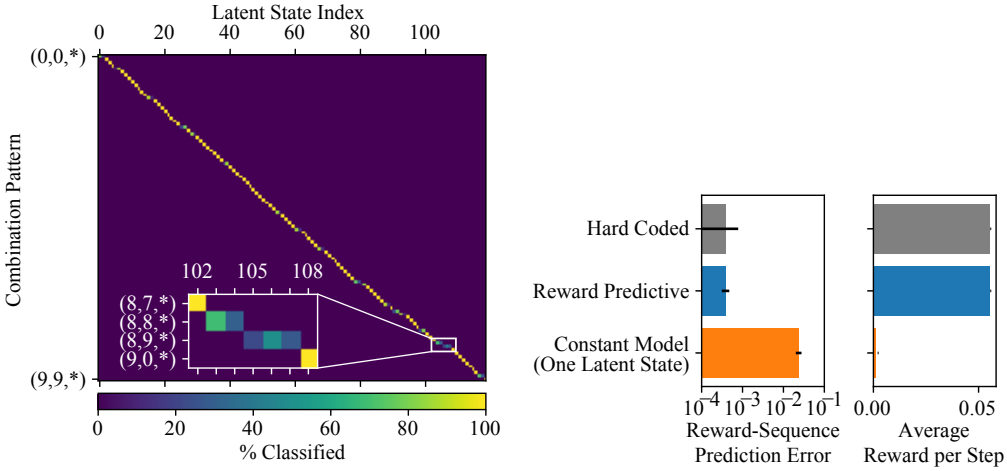
Figure 5.7 illustrates the used training and transfer tasks: Here, we use an extension of the combination-lock task previously presented in Section 3.5.1 where each dial has ten different positions and the lock's number combination is converted into an image using digits from the MNIST image data set [62]. Because the right dial always spins at random, both tasks have 100 embedded latent states. Still, the number of possible state images is large, because the MNIST data set contains multiple different images for the same digit and because the third digit in the state image is not relevant for the task. In both tasks, the agent has to repeatedly move the left and centre dials one number at a time to reach a rewarding number combination. Both training and transfer tasks differ in their rewards and transitions, because a different number combination is rewarded and because the rotation direction of the centre dial is reversed in the transfer task.

To learn a reward-predictive representation, a training data set of 400000 transitions is sampled

from the training task using uniform random action selection. This data set is generated using images from the MNIST training set. Then, the reward-predictive clustering algorithm is run on this data set using a deep learning subroutine to approximate rewards, SFs, and the resulting reward-predictive representation with a deep convolutional neural network. To satisfy the ε-perfect assumption as closely as possible, this neural network was scaled in size by adding multiple layers and integrating other deep learning techniques such as dropout [99] and batch normalization [50]. Table B.10 describes the used network in detail.

Figure 5.8 illustrates how the learned reward-predictive representation network generalizes across different images and reports the learned model's performance. For the training task, the reward-predictive clustering algorithm discovered 119 latent states, more latent states than actually needed. The matrix plot in Figure 5.8(a) shows that these extra states were constructed because number combinations corresponding the the same embedded latent state are not associated with the same learned latent state. For example, images containing number combinations that start with $8, 9$ are mapped into three different latent states. This is caused by approximation errors when estimating the feature descriptor vectors $\hat{\boldsymbol{\xi}}$ or reward vectors $\hat{\boldsymbol{r}}$ for different observed states. These approximation errors push these vectors apart for states that should be clustered together. Consequently, the clustering operation assigns different latent states to these images. Furthermore, to some extent, approximation errors also cause images containing completely different number combinations to be incorrectly associated with the same latent state. Still, these approximation errors affects only a very small number of states as illustrated in Figure 5.8(b), where most off-diagonal entries are close to zero.

We also evaluated the learned reward-predictive representation network using the same evaluation procedure used in Section 5.2.1. The box-plots in Figure 5.8(b) summarize the obtained results. During evaluation, the MNIST test set is used to render the image states to ensure that the training data is different from the evaluation (test) data. Furthermore, we also train a neural network to classify images based on the left and centre digits into the 100 embedded latent states. This neural network forms a baseline model: While neither neural network can perfectly classify all test images, the baseline model is trained using optimal latent state labels and the reward-predictive representation network is trained using the labels found by the clustering algorithm. Besides using different labels, the deep learning routine used to obtain this baseline model is identical to the deep learning routine used to learn the reward-predictive representation network. Both plots in



(a) Classification of Number Combinations Images into Latent States

(b) Comparison of Learned Representation On Training Task With Test MNIST Images

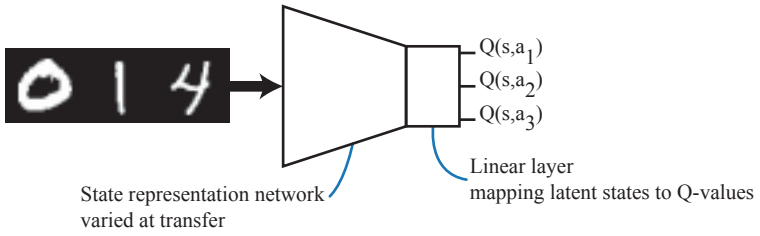
Figure 5.8: Reward-predictive representation learned for the MNIST combination-lock training task. 5.8(a): Matrix plot where each row shows which number combinations are classified into which latent state. For this plot, the state observations of the training data set were used. The combination $(8, 7, *)$ corresponds to any number combination where the left and centre dials are set to eight and seven, respectively. Each row of this matrix plot sums to 100%. An optimal clustering would assign each combination a separate latent state, resulting in a diagonal matrix. However, approximation errors lead to spurious latent states that are visualized as additional columns. 5.8(b): Bar-plot comparing the performance of the learned reward-predictive representation with a network trained to predict the 100 embedded states and a one-latent-state model for reference. Reward-sequence prediction errors and the average reward per step (of the policy optimal with respect to the learned model) are computed as described in Section 5.2.1.

Figure 5.8(b) suggest that the learned reward-predictive representation network performs similarly to the baseline model. Even the baseline model occasionally predicts incorrect reward sequences, because this network is not capable of perfectly classifying all test images correctly. A constant model that merges all states into one latent state performs poorly. This result is expected, because this model does not use the correct latent representation.

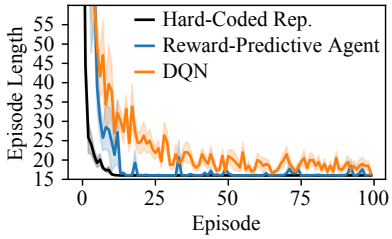
Next, we test if re-using the learned reward-predictive representation network leads to learning an optimal policy faster in the transfer combination lock (Figure 5.7). To obtain a controlled transfer experiment, we adopted the DQN algorithm [73] to learn an optimal policy and vary the initialization and learning conditions of the used Q-network.[5] Figure 5.9(a) illustrates the used Q-network architecture: The DQN algorithm uses the exact same network architecture as the learned reward-predictive representation but initializes all weights using Xavier initialization [47] (an initialization method commonly used in deep learning). The reward-predictive agent, which re-uses the learned reward-predictive representation, replaces all but the top-most linear layer of the Q-network with the transferred representation. If this representation is suitable for transfer, then this reward-predictive agent should learn an optimal policy faster than the default agent, because most network weights are initialized to a (close to) optimal solution. Note that the reward-predictive agent only updates the top-most linear layer and the transferred representation network's weights are fixed during learning in the transfer task. In this comparison, the DQN algorithm serves as an experimental control. As a reference, we are also testing a hard-coded representation agent that replaces all lower network layers with a classifier that has access to the transfer lock's embedded latent state. Because this agent has access to the internal state of the combination-lock task (a form of ground-truth knowledge), input images are always classified into the correct embedded latent state. This hard-coded representation agent is used to set a baseline for the transfer simulation and should perform best.

Figure 5.9(b) compares how quickly the three different agent configurations reach a short (optimal) episode length. Because the combination-lock task only rewards entering the goal state at which an episode finishes, always completing an episode within as few steps as possible indicates an optimal policy. Here, an optimal policy completes the transfer combination-lock task in 16 steps. In this transfer task the agent always starts at combination $(0, 0, 0)$ and has to reach the combination pattern $(8, 2, *)$ by rotating the left and centre dials. This can be accomplished in (at least) 16 steps. The plot in Figure 5.9(b) indicates that the reward-predictive agent (blue curve) converges

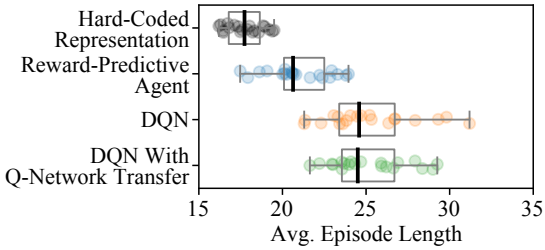
[5] Table B.11 in Appendix B.4.2 lists the used hyper-parameter.



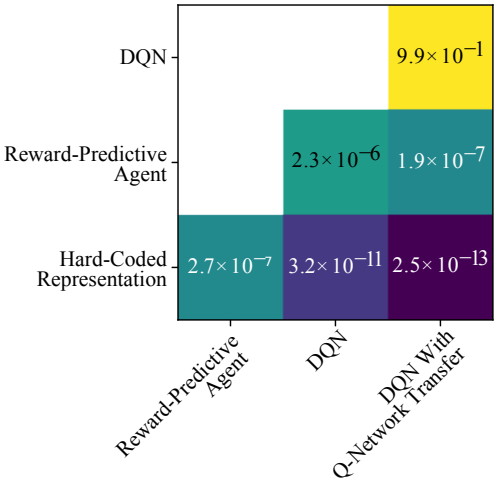
(a) The Tested Deep Q-Networks Integrate Different Representation Networks



(b) Eposide Length On The Transfer Task



(c) Avg. Episode Length On The Transfer Task



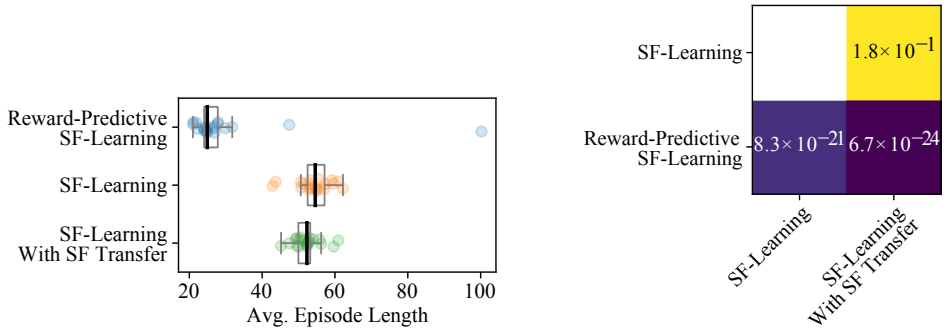
(d) P-values Testing For Difference in Avg.Performance

Figure 5.9: Reward-predictive representation networks can be re-used in the MNIST combination-lock transfer task without modifications. 5.9(a): The used Q-network is a deep convolutional neural network mapping each image to a vector of Q-values. For each simulation, the used network architecture is identical to the network architecture used by the clustering algorithm. The reward-predictive agent initializes this network with the learned reward-predictive representation network and does not update the weights of the representation network during learning. 5.9(b): Average episode length comparison between the reward-predictive agent, the DQN agent, and the hard-coded representation agent. Each simulation is repeated 20 times and the shaded area indicates the standard error of measure. 5.9(c): Box-plot of the average episode length of each evaluated agent averaged over 20 simulation repeats. 5.9(d): Welch's t-test p-values testing for a significant difference in the average episode length of each agent. A low p-value indicates a significant difference in average episode length. A goodness of fit test (Kolmogorov–Smirnov test) of the collected data with a normal distribution resulted in p-values of at least 68% and does not suggest that the data does not follow a normal distribution. Consequently, using the Welch's t-test, which assumes normally distributed data, is appropriate. This plot indicates that there is a significant difference in performance between using a hard-coded representation, a previously learned reward-predictive representation network, and training the Q-network from scratch using the DQN algorithm. Re-using a Q-network learned for the training task does not lead to a significant performance improvement.

significantly faster than the DQN (orange curve), suggesting that re-using the previously learned reward-predictive representation leads to faster convergence. As expected, the hard-coded representation agent (black curve) outperforms the other two agents. There is a significant performance difference between the hard-coded representation agent and the reward-predictive agent, because approximation errors of the re-used reward-predictive representation network limit fast convergence to an optimal policy to some extent. Still, this reward-predictive agent converges to an optimal policy without changing any weights of the re-used representation network or adapting the representation network in any other way to the task at hand. This finding stands in contrast to prior work on Deep SF-learning algorithms [12] and meta-learning methods such as MAML [37], where previously learned networks are adapted to the task at hand.

Figure 5.9(c) presents a box-plot comparing the tested algorithm's performance with a fourth DQN agent configuration where learning is initialized with a Q-network trained on the training combination-lock task (box-plot with green points labelled "DQN With Q-Network Transfer"). The transferred Q-network initialization is then updated until the algorithm converges to an optimal policy. Updating the Q-network and changing its weights is necessary because the transitions and rewards differ between the combination-lock training and transfer task (Figure 5.7) and consequently the optimal policy differs as well. Still, transferring a Q-network trained on the combination-lock training task does not improve performance significantly, because the policy optimal in the transfer combination-lock task differs too significantly from the policy optimal in the training task. This Q-network transfer algorithm is similar to the Deep SF framework presented by Barreto et al. [12], where a SF network is learned for a set of MDPs that differ in their rewards. These SF networks are constructed such that they are equivalent to a Q-network. Specifically, for MDP i, the predicted SF vector $\boldsymbol{\psi}^{\pi_i}(s,a) = Q^{\pi_i}(s,a)$, where π_i is the policy learned in MDP i. While Barreto et al. focus on multi-task transfer and how to combine multiple previously learned networks, the Q-network transfer algorithm presented in Figure 5.7 resembles this method for a single task setting. Yet, this system does not exhibit positive transfer on the combination-lock tasks, because these two tasks are varied in their transitions and rewards and most of the transferred network has to be re-trained to find an optimal policy.

Figure 5.9(d) reports p-values testing for significant differences in average performance. These values indicate that the reward-predictive agent significantly outperforms the DQN algorithm, demonstrating that re-using a previously learned reward-predictive representation network does



(a) Avg. Episode Length on Transfer Task

(b) P-Values Testing for Difference in Avg. Performance

Figure 5.10: Re-using reward-predictive representations leads to faster convergence than re-using SFs. 5.10(a): Box-plot of the average episode length for each agent across 20 repeats. Table B.12 lists the hyper-parameters used for this experiment. 5.10(b): P-values of a Welch's t-test testing for significant differences in average performance. A goodness of fit test (Kolmogorov–Smirnov test) of the collected data with a normal distribution resulted in p-values of at least 61% and does not suggest that the data does not follow a normal distribution. Consequently, using the Welch's t-test is appropriate.

accelerate learning in the MNIST combination-lock transfer task.

Lastly, Figure 5.10 highlights that transferring SFs as described in Chapter 2 accelerates learning less significantly than transferring reward-predictive representations. To obtain a fair comparison, we combine the (linear) SF-learning algorithm (Algorithm 2) with different state-representation functions to map images to a discrete set of latent states and record whether using a specific representation type accelerates learning on the MNIST combination-lock transfer task (Figure 5.7). The reward-predictive SF-learning agent re-uses the reward-predictive representation network learned for the combination-lock training task to map images to discrete latent states. These discrete latent states are then fed as input into the SF-learning algorithm. Consequently, this agent uses the reward-predictive representation to generalize across different states and compresses the task's image state space into a small latent space. For comparison, the default SF-learning algorithm uses a digit classifier that maps each of the 1000 different number combination shown in a state image to one of 1000 different latent states. While this classifier processes the state image inputs for the agent, the resulting agent does not generalize across states that are equivalent in terms of predicting reward sequences. To test if re-using previously learned SFs leads to accelerated learning, this default agent was first used to learn an optimal policy on the combination-lock training task. The learned successor features are then used to initialize learning in the combination-lock transfer task.

The results plotted in Figure 5.10 suggest that transferring the previously learned reward-predictive representation network accelerates learning significantly. While re-using SFs does accelerate learning slightly, as indicated by the box-plot and p-values for the "SF-Learning With SF Transfer" agent, this method is still outperformed by the reward-predictive agent. This performance improvement can be attributed to the fact that the reward-predictive model builds an internal representation that more closely models how to generalize across different number combinations. Because this aspect is preserved at transfer, this method outperforms the SF transfer method that models a task's optimal policy, which differs in the used tasks.

The transfer experiments presented in Figures 5.9 and 5.10 focus on testing whether using a specific representation type leads to positive transfer and therefore only use the DQN algorithm and SF-learning to obtain an optimal policy. Because reward-predictive representations can be used to predict reward-sequences and enable model-based RL, one could also implement a (deep) model-based RL agent. At transfer, this agent would first estimate the latent transition and reward tables through exploration and could then immediately compute an optimal policy. By using a reward-predictive representation to map state images to discrete latent states, it is possible to simulate a model-based RL algorithm such as RMax [23] on a visual control task and find an optimal policy faster than an equivalent model-free learning approach [101], similar to the experiments presented in Section 4.2.3. However, such a comparison is left to future work.

5.3 Discussion

In this chapter, we explored a clustering algorithm that learns reward-predictive representations for control tasks where state spaces can be uncountably infinite sets. Given a transition data set as input, this algorithm clusters different state observations and each cluster corresponds to a reward-predictive latent state. While the presented partition refinement approach is similar to the block splitting technique presented by Givan et al. [46], the presented cluster algorithm is designed to process state data from an uncountably infinite input set and is not constrained to a finite state space. State data is processed using an ERM subroutine to obtain a function, for example a neural network, to predict one-step rewards, SFs, and latent states. Furthermore, in Section 5.1 we study under which assumptions the algorithm is robust to prediction errors that stem from this ERM subroutine and when a maximally-compressed reward-predictive representation is obtained.

Specifically, the assumptions state that the used ERM subroutine needs to be ε-perfect and returns functions whose approximation errors can be bound by some (small) ε. While this assumption may seem restrictive, Belkin et al. [16] demonstrate that over-parameterized neural networks can be used to interpolate training data and exhibit near perfect performance during testing. The ε-perfect assumption is necessary because the clustering algorithm uses the ERM subroutine's result to inform the next clustering iteration. Both simulations presented in Section 5.2 demonstrate that not satisfying this ε-perfect assumption leads to constructing spurious latent states and "overrefining" the partitioning of the observed state set. In these cases, prediction errors perturb some of the vectors used for clustering, resulting in the construction of additional clusters. However, withholding clusters that correspond to only very few observed states from the next refinement iteration mitigates the construction of spurious latent states to some extent.

Because reward-predictive representations are used to predict reward sequences, these representations are evaluated on their predictive performance. Consequently, any approximation errors (caused by not adhering to the ε-perfection assumption) impact the resulting model's predictive performance—a property common to model-based RL algorithms [109, 110, 6]. Evaluating a model's predictive performance is more stringent than what is typically used for model-free RL algorithms such as DQN. Typically, model-free RL algorithms are evaluated on the learned optimal policy's performance and are not evaluated on their predictive performance. For example, while DQN can learn an optimal policy for a task, the learned Q-network's prediction errors may still be high for some inputs [120]. Prediction errors of this type are often tolerated, because model-free RL algorithms are evaluated based on the learned policy's performance and not their predictive performance. In contrast, reward-predictive representations are evaluated for their prediction accuracy. To achieve low prediction errors, the presented results suggest that finding ε-perfect approximations becomes important. Furthermore, the simulations on the MNIST combination-lock task demonstrate that this goal can be accomplished by using a larger neural network architecture.

While model-based and model-free hybrid architectures [80, 97, 92] also incorporate a reward-sequence-prediction component, these models predict reward-sequences only over very short horizons (for example, Oh et al. [80] use 10 time steps). These short reward-sequence predictions are used to accelerate a model-free learning process that obtains an optimal policy through trial and error

interaction. Because model-free learning is also used to to improve the model used for reward-sequence predictions, these model-based and model-free hybrids do not implement "strict" model-based RL systems. In contrast, reward-predictive representations are learned from a fixed data set that was collected using uniform random action selection. With the learned model, reward sequences can be predicted for horizons that are sufficiently long to assess the execution of a task from the beginning to the end. In fact, we demonstrate that value iteration can be used to extract an optimal policy directly from the learned model and without any additional trial-and-error learning. A similar observation can be made concerning deep bisimulation learning methods [44] that implement a model-free and model-based hybrid architecture to accelerate model-free learning and are not a form of "strict" model. However, learning reward-predictive representations on larger scale tasks comparable to the tasks used in related work [80, 97, 92, 38, 79, 66, 84], for example Atari games [17], is left for future work.

Section 5.2.2 extends the transfer results presented in Chapters 4 and 3 to visual control tasks where function approximation is needed. We demonstrate that the learned reward-predictive representation network generalizes across transitions and rewards just as the tabular reward-predictive representations studied in the previous chapters generalize across transitions and rewards. The key distinction between the simulations presented in this chapter is that the representation function is implemented with a neural network mapping images to latent state vectors instead of mapping state indices to latent state vectors. At transfer, the reward-predictive representation network does not need to be updated or adapted to the task at hand, because reward-predictive representations encode task knowledge that generalizes across variations in transitions and rewards. This choice stands in contrast to prior work on (Deep) SF transfer [12, 13, 61, 123], meta-learning [37], or multi-task learning [90, 30] (see also Khetarpal et al. [55] for a survey). These methods transfer a value function or policy model to initialize and accelerate learning. Because these methods transfer a model of a task's policy, these models have to be adapted to each transfer task, if the transfer task's optimal policy differs from the previously learned policies. Reward-predictive representations overcome this limitation by only modelling how to generalize across different states. Because reward-predictive representations do not encode the specifics of how to transition between different latent states or how latent states are tied to rewards, these representations are robust changes in transitions and rewards. Furthermore, the reward-predictive representation network is learned using a single task and the resulting network is sufficient to demonstrate positive transfer across different transitions

and rewards. This property stands also in contrast to Zhang et al. [122], where the focus is on extracting a common task structure from a set of tasks. Still, in a lifelong learning scenario, re-using the same reward-predictive representation network to solve every task may not be possible because an agent may have to generalize across different states (as discussed in Chapter 4). In the MNIST combination-lock example, transfer is possible if the right dial always spins at random, because the reward-predictive representation network has learned to ignore and abstract away this dial. However, if instead the center dial is randomized at transfer, then the previously learned reward-predictive representation cannot be re-used. As demonstrated in Section 4.2.3, one could use different (non-parametric) meta-learning models to combine multiple learned representations. Implementing these meta-learning models on visual control tasks is left for future work.

One assumption made in the presented experiments is that a task's state space can always be compressed into a small enough finite latent space. While compressing a finite data set into finitely many latent states is always possible (by associating each observed state with a distinct latent state), the presented model always learns a fully conjunctive representation. In the combination-lock examples, the reward-predictive representation associates a different latent state (one-hot vector) with each relevant combination pattern. This representation is conjunctive because it does not model the fact that the dials rotate independently. A disjunctive or factored representation could map each of the three dials independently into three separate latent state vectors and a concatenation of these vectors could be used to describe the task's latent state. Such a latent representation is similar to factored representations used in prior work [49, 31] and these factored representations permit a more compositional form of generalization across different tasks [54, 14, 25]. How to extract such factored representations from unstructured state spaces such as images still remains a challenging problem. We leave such an extension of reward-predictive representations and LSFMs to future work.

Lastly, the presented experiments generate training data sets by simulating trajectories using uniform random action selection. While this method is sufficient for learning reward-predictive representations on the tested tasks, a more directed exploration approach may be necessary in more complex control tasks. Here, the presented insights could be used to design model-based exploration strategies to optimize the data-collection process. For example, after collecting one trajectory, a reward-predictive representation could be computed given this data and latent state-action combinations could be identified that have not been visited so far. These missing latent state-action combinations could then be used to inform exploration in the next step and this process

could be repeated. Using latent states for efficient exploration has been studied before by Du et al. [32]. Yet, Du et al. focus on exploration efficiency and not on constructing a particular representation type. The results presented in this chapter could be used to develop new model-based exploration strategies to learn reward-predictive representations.

5.4 Conclusion

The presented results and clustering algorithm demonstrate how reward-predictive representations can be scaled to tasks with (uncountably) infinite state spaces where function-approximation techniques, such as deep learning, are required. We formally showed under what assumptions the presented algorithm converges to a maximally-compressed reward-predictive representation. Further, we demonstrated that the learned reward-predictive representation generalizes to test data not observed during training. Lastly, we illustrated that the learned reward-predictive representations generalize across tasks with different transitions and rewards. These results support the thesis statement proposed in Chapter 1 to tasks where state spaces are (uncountably) infinite.

Chapter 6

Conclusions and Future Directions

Implementing knowledge re-use in artificial reinforcement learning systems accelerates learning by avoiding repeatedly learning the same solution from scratch. This dissertation studies how to implement knowledge re-use in reinforcement learning systems. While RL systems are typically designed to maximize rewards in a single task, we find that a RL system's objective changes in the context of lifelong learning: Instead of finding a maximally compressed model to solve a single task, the presented research indicates that learning a model detailed enough to predict reward sequences leads to internalizing re-usable task knowledge. We demonstrate that knowledge re-use across tasks can be accomplished by re-using reward-predictive representations. Because reward-predictive representations model analogies between different task states they are not tied to specifics of the task's transitions or rewards. These analogies encode how to generalize across inputs in a specific task. If these analogies are preserved at transfer, then the learned reward-predictive representation can be re-used.

In Chapters 2 and 3 we present an analysis of which latent representations an intelligent agent can construct to support different predictions, leading to new connections between model-based and model-free RL. We find that different SF formulations can be used to implement either model-free learning or model-based learning. While prior work [11] implements a model-free SF learning mechanism, we introduce LSFMs and tie our new SF formulation to model-based learning. Specifically, we phrase model-based learning as a (reward-predictive) representation-learning problem. This method differs from the usual approach to model-based RL (e.g. Dyna [103] or RMax [23]) where the transition and reward function is approximated. Chapter 4 presents results suggesting that reward-predictive state abstractions generalize across tasks with different transition and reward functions.

This stands in contrast to prior work on SFs that focuses on transfer where only the reward function is varied [11, 12, 13, 75, 89, 100, 61, 123]. Furthermore, we study a lifelong learning scenario where re-using a single reward-predictive representation is not possible. Here, we demonstrate with a non-parametric Bayesian model that an agent can learn a set of different reward-predictive representations and can quickly identify which previously learned representation should be re-used. Lastly, we present in Chapter 5 a clustering algorithm that integrates deep learning techniques to learn reward-predictive representations on tasks with uncountably infinite state spaces. These results extend the previous transfer discussion and demonstrate that reward-predictive representations also generalize across tasks with uncountably infinite state spaces where function approximation is needed.

6.1 Future Directions

Biologically, our findings motivate further research to investigate whether brain systems involved in state representation implement reward-predictive representations. While existing work already presents neural and behavioural correlates of the SR [75, 89], these experiments focus only on variations in rewards. The models presented in this dissertation motivate experimental designs (similar to Franklin and Frank [41]) to test whether human subjects re-use and learn a latent structure that is preserved despite variations in transitions and rewards. Furthermore, offline hippocampal replay has been proposed to reflect sampling from a model to enable model-free learning and planning [72, 69, 93, 5]. In alignment with this notion, the transfer simulations presented in Section 4.2.3 retain an identity abstraction when learning in a completely novel task and representation learning is performed offline between task transitions. Retaining an identity map instead of generalizing in a completely novel task is important to for complete exploration and to obtain a representative data set that informs an accurate reward-predictive representation. The simulations in Section 5.2.1 on the image puddle world task further confirm the need for efficient exploration. If the training data set leaves portions of a task out, then the resulting reward-predictive representation is sub-optimal. Consequently it may be difficult to learn a reward-predictive representation in an online learning setting. Instead, the presented clustering algorithm would be used after exploring a novel task, similar to the transfer learning simulation with multiple abstractions presented in Section 4.2.3. The

connection between reward-predictive representations and model construction motivates further research on replay that would test if human subjects learn and re-use a latent structure across tasks that vary in their transitions and rewards.

In the combination lock simulations (Section 5.2.2), the learn reward-predictive representation maps different number combinations into a discrete set of latent states. This latent space is conjunctive representation, because each number combination is mapped to a distinct one-hot vector. Alternatively, one could also construct a disjunctive representation that separately maps each digit into one of ten different "dial latent states". The combination of these "dial latent states" could then be used for predicting reward-sequences. Such a representation is disjunctive and further explores the task's structure, similar to factored state representations [49]. A disjunctive or factored reward-predictive representation would allow a more compositional form of knowledge re-use similar to Kansky et al. [54], because at transfer one could only re-use and combined different reward-predictive components. However, how to learn such factored reward-predictive representations is left to future work.

Appendix A

Proofs of Theoretical Results

A.1 SF-learning and Q-Learning Connection (Chapter 2)

Proof of Proposition 1. Before proving the main statement, we first make the following observation. Assuming that for some t, $\boldsymbol{\theta}_t = \boldsymbol{F}_t \boldsymbol{w}$, then

$$\boldsymbol{w}^\top \boldsymbol{y}_{s,a,r,s'} = \boldsymbol{w}^\top \left(\boldsymbol{\xi}_{s,a} + \gamma \sum_{a'} b(s', a') \boldsymbol{\psi}^{\pi}_{s',a'} \right) \tag{A.1}$$

$$= r(s,a) + \gamma \sum_{a'} b(s', a') \boldsymbol{w}^\top \boldsymbol{\psi}^{\pi}_{s',a'} \tag{A.2}$$

$$= r(s,a) + \gamma \sum_{a'} b(s', a') \boldsymbol{w}^\top \boldsymbol{F}_t \boldsymbol{\xi}_{s',a'} \tag{A.3}$$

$$= r(s,a) + \gamma \sum_{a'} b(s', a') \boldsymbol{\theta}_t^\top \boldsymbol{\xi}_{s',a'} \tag{A.4}$$

$$= y_{s,a,r,s'}. \tag{A.5}$$

Equation (A.3) follows by substitution with Equation (2.16). The proof of the main statement is by induction on t.

Base Case: For $t = 1$, assume $\boldsymbol{\theta}_0 = \boldsymbol{F}_0\boldsymbol{w}$. Then

$$\boldsymbol{w}^\top \boldsymbol{F}_1 = \boldsymbol{w}^\top \left(\boldsymbol{F}_0 + \alpha_\psi \left(\boldsymbol{F}_0\boldsymbol{\xi}_{s,a} - \boldsymbol{y}_{s,a,r,s'}\right)^\top \boldsymbol{\xi}_{s,a}\right) \tag{A.6}$$

$$= \boldsymbol{w}^\top \boldsymbol{F}_0 + \alpha_\psi \left(\boldsymbol{w}^\top \boldsymbol{F}_0\boldsymbol{\xi}_{s,a} - \boldsymbol{w}^\top \boldsymbol{y}_{s,a,r,s'}\right)^\top \boldsymbol{\xi}_{s,a} \tag{A.7}$$

$$= \boldsymbol{\theta}_0 + \alpha_\psi \left(\boldsymbol{\theta}_0^\top \boldsymbol{\xi}_{s,a} - y_{s,a,r,s'}\right)^\top \boldsymbol{\xi}_{s,a} \tag{A.8}$$

$$= \boldsymbol{\theta}_1. \tag{A.9}$$

Equation (A.8) us obtained by substituting the identity in Equation (A.5) for $t = 0$. Equation (A.9) is obtained by substituting the linear TD iterate from Equation (2.13).

Induction Step: Assuming the hypothesis $\boldsymbol{w}^\top \boldsymbol{F}_t = \boldsymbol{\theta}_t^\top$ holds for t and proceeding as in the base case, then

$$\boldsymbol{w}^\top \boldsymbol{F}_{t+1} = \boldsymbol{w}^\top \left(\boldsymbol{F}_t + \alpha_\psi \left(\boldsymbol{F}_t\boldsymbol{\xi}_{s,a} - \boldsymbol{y}_{s,a,r,s'}\right)^\top \boldsymbol{\xi}_{s,a}\right) \tag{A.10}$$

$$= \boldsymbol{w}^\top \boldsymbol{F}_t + \alpha_\psi \left(\boldsymbol{w}^\top \boldsymbol{F}_t\boldsymbol{\xi}_{s,a} - \boldsymbol{w}^\top \boldsymbol{y}_{s,a,r,s'}\right)^\top \boldsymbol{\xi}_{s,a} \tag{A.11}$$

$$= \boldsymbol{\theta}_t + \alpha_\psi \left(\boldsymbol{\theta}_t^\top \boldsymbol{\xi}_{s,a} - y_{s,a,r,s'}\right)^\top \boldsymbol{\xi}_{s,a} \tag{A.12}$$

$$= \boldsymbol{\theta}_{t+1}. \tag{A.13}$$

Hence for all t, $\boldsymbol{w}^\top \boldsymbol{F}_t = \boldsymbol{\theta}_t$, as desired.

Note that this proof assumes that both iterates are applied for exactly the same transitions. This assumption is not restrictive assuming that control policies are constructed using the current parameters $\boldsymbol{\theta}_t$ in the case for TD-learning or the parameters $\boldsymbol{F}_t$ and $\boldsymbol{w}$ in the case for SF-learning. Even in the control case, where an ε-greedy exploration strategy is used, for example, both algorithms will produce an identical sequence of value functions and will chose actions with equal probability. □

A.2 LSFM Theorems (Chapter 3)

For an equivalence relation $\sim$ defined on a set $\mathcal{S}$, the set of all partition is denoted with $\mathcal{S}/\sim$. Each partition $[s] \in \mathcal{S}/\sim$ is a subset of $\mathcal{S}$ and $s \in [s]$.

Definition 5 (Bisimilarity [35])**.** *For an MDP $M = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma \rangle$ where $\langle \mathcal{S}, \Sigma, p \rangle$ is a measurable space with σ-algebra Σ and p is a Markov kernel labelled for each action $a \in \mathcal{A}$. Consider an*

equivalence relation $\sim_b$ *on the state space* $\mathcal{S}$ *such that each state partition* $[s']$ *also lies in the* σ*-algebra and* $\forall [s'] \in \mathcal{S}/\sim_b$, $[s'] \in \Sigma$. *The equivalence relation* $\sim_b$ *is a bisimulation if*

$$s \sim_b \tilde{s} \iff \forall a \in \mathcal{A},\ \mathbb{E}_p\left[r(s,a,s')|s,a\right] = \mathbb{E}_p\left[r(\tilde{s},a,s')|\tilde{s},a\right] \tag{A.14}$$

$$\text{and } \forall [s'] \in \mathcal{S}/\sim_b,\ p([s']|s,a) = p([s']|\tilde{s},a). \tag{A.15}$$

Using this definition, Theorem 1 can be proven.

Proof of Theorem 1. Consider any two states s and $\tilde{s}$ such that $\boldsymbol{\phi}_s = \boldsymbol{\phi}_{\tilde{s}}$. For both s and $\tilde{s}$ we have

$$\mathbb{E}_p\left[r(s,a,s')|s,a\right] = \boldsymbol{\phi}_s^\top \boldsymbol{w}_a = \boldsymbol{\phi}_{\tilde{s}}^\top \boldsymbol{w}_a = \mathbb{E}_p\left[r(\tilde{s},a,s')|\tilde{s},a\right], \tag{A.16}$$

and the bisimulation reward condition in Equation (A.14) holds. To show that also the bisimulation transition condition in Equation (A.15) holds, observe that

$$\boldsymbol{\phi}_s^\top = \boldsymbol{\phi}_{\tilde{s}}^\top \qquad \iff \tag{A.17}$$

$$\boldsymbol{\phi}_s^\top \boldsymbol{M}_a = \boldsymbol{\phi}_{\tilde{s}}^\top \boldsymbol{M}_a \qquad \iff \tag{A.18}$$

$$\mathbb{E}_p\left[\boldsymbol{\phi}_{s'}|s,a\right] = \mathbb{E}_p\left[\boldsymbol{\phi}_{s'}|\tilde{s},a\right] \qquad \iff \tag{A.19}$$

$$\sum_{i=1}^{n} p(s,a,[s_i])\boldsymbol{e}_i = \sum_{i=1}^{n} p(\tilde{s},a,[s_i])\boldsymbol{e}_i, \tag{A.20}$$

where $[s_i] \subset \mathcal{S}$ are all states that are mapped to the one-hot feature vector $\boldsymbol{e}_i$. Each side of the identity (A.20) computes an expectation over one-hot bit vectors and thus the ith entry of $\sum_{i=1}^{n} p(s,a,[s_i])\boldsymbol{e}_i$ contains the probability value $p(s,a,[s_i])$. Hence both s and $\tilde{s}$ have equal probabilities of transitioning into each state partition that is associated with $\boldsymbol{e}_i$. Define an equivalence relation $\sim_\phi$ such that

$$\forall s, \tilde{s} \in \mathcal{S},\ \boldsymbol{\phi}_s = \boldsymbol{\phi}_{\tilde{s}} \iff s \sim_\phi \tilde{s}. \tag{A.21}$$

Because all feature vectors $\boldsymbol{\phi}_s$ are one-hot bit vectors, there are at most n partitions and the set of all state partitions has size $|\mathcal{S}/\sim_\phi| = n$. Combining these observations, Equation (A.20) can be rewritten as

$$\forall [s'] \in \mathcal{S}/\sim_\phi,\ p([s']|s,a) = p([s']|\tilde{s},a). \tag{A.22}$$

By lines (A.16) and (A.22), the equivalence relation $\sim_\phi$ is a bisimulation relation and if $\boldsymbol{\phi}_s = \boldsymbol{\phi}_{\tilde{s}}$

then both s and $\tilde{s}$ are bisimilar. □

Lemma 2. *Assume an MDP, state representation $\phi : \mathcal{S} \to \{\boldsymbol{e}_1, ..., \boldsymbol{e}_n\}$, LSFM $\{\boldsymbol{F}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$, and arbitrary policy $\pi \in \Pi_\phi$. Let $\boldsymbol{F}^\pi$ be an $n \times n$ real valued matrix with each row $\boldsymbol{F}^\pi(i) = \mathbb{E}_\pi\left[\boldsymbol{e}_i^\top \boldsymbol{F}_a \middle| s\right]$, then*

$$\mathbb{E}_\pi\left[\boldsymbol{\phi}_s^\top \boldsymbol{F}_a \middle| s\right] = \boldsymbol{e}_i^\top \boldsymbol{F}^\pi, \tag{A.23}$$

where $\boldsymbol{\phi}_s = \boldsymbol{e}_i$ for some i. For a LAM $\{\boldsymbol{M}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$, let $\boldsymbol{M}^\pi$ be a $n \times n$ real-valued matrix with each row $\boldsymbol{M}^\pi(i) = \mathbb{E}_\pi\left[\boldsymbol{e}_i^\top \boldsymbol{M}_a \middle| s\right]$, then

$$\mathbb{E}_\pi\left[\boldsymbol{\phi}_s^\top \boldsymbol{M}_a \middle| s\right] = \boldsymbol{e}_i^\top \boldsymbol{M}^\pi. \tag{A.24}$$

Proof of Lemma 2. The first identities in (A.23) and (A.24) hold because $\boldsymbol{\phi}_s = \boldsymbol{e}_i$ for some i. Then,

$$\mathbb{E}_\pi\left[\boldsymbol{\phi}_s^\top \boldsymbol{F}_a \middle| s\right] = \sum_a \pi(s,a)\boldsymbol{\phi}_s \boldsymbol{F}_a = \sum_a \pi(s,a)\boldsymbol{e}_i^\top \boldsymbol{F}_a = \mathbb{E}_\pi\left[\boldsymbol{e}_i^\top \boldsymbol{F}_a \middle| s\right] = \boldsymbol{F}^\pi(i) = \boldsymbol{e}_i^\top \boldsymbol{F}^\pi$$

and

$$\mathbb{E}_\pi\left[\boldsymbol{\phi}_s^\top \boldsymbol{M}_a \middle| s\right] = \sum_a \pi(s,a)\boldsymbol{\phi}_s \boldsymbol{M}_a = \sum_a \pi(s,a)\boldsymbol{e}_i^\top \boldsymbol{M}_a = \mathbb{E}_\pi\left[\boldsymbol{e}_i^\top \boldsymbol{M}_a \middle| s\right] = \boldsymbol{M}^\pi(i) = \boldsymbol{e}_i^\top \boldsymbol{M}^\pi.$$

□

Definition 6 (Weighting Function)**.** *For an MDP $M = \langle \mathcal{S}, \mathcal{A}, p, r, \gamma\rangle$, let $\sim$ be an equivalence relation on the state space $\mathcal{S}$, creating a set of state partitions $\mathcal{S}/\sim$. Assume that each state partition $[s]$ is a measurable space $\langle [s], \Sigma_{[s]}, \omega_{[s]}\rangle$, where $\omega_{[s]}$ is a probability measure indexed by each partition $[s]$ with σ-algebra $\Sigma_{[s]}$. The function ω is called the weighting function.*

Lemma 3. *Assume an MDP, state representation $\phi : \mathcal{S} \to \{\boldsymbol{e}_1, ..., \boldsymbol{e}_n\}$, LSFM $\{\boldsymbol{F}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$, and arbitrary abstract policy $\pi \in \Pi_\phi$. Then,*

$$\forall s, \forall a\ \boldsymbol{\phi}_s^\top \boldsymbol{F}_a = \boldsymbol{\phi}_s^\top + \gamma \mathbb{E}_{p,\pi}\left[\boldsymbol{\phi}_{s'} \boldsymbol{F}_{a'} | s, a\right] \implies \forall s, \forall a, \exists \boldsymbol{M}_a \text{ such that } \boldsymbol{F}_a = \boldsymbol{I} + \gamma \boldsymbol{M}_a \boldsymbol{F}^\pi, \tag{A.25}$$

where the matrix $\boldsymbol{F}^\pi$ is constructed as described in Lemma 2.

Proof of Lemma 3. Consider an equivalence relation $\sim_\phi$ that is constructed using the state representation ϕ and

$$\forall s, \tilde{s} \in \mathcal{S},\ \boldsymbol{\phi}_s = \boldsymbol{\phi}_{\tilde{s}} \iff s \sim_\phi \tilde{s}. \tag{A.26}$$

The weighting function ω models a probability distribution of density function of visiting a state s that belongs to a state partition $[s] \in \mathcal{S}/\sim_\phi$. Because all states $s \in [s]$ are mapped to the same feature vector $\boldsymbol{\phi}_s$, we have that

$$\mathbb{E}_{\omega_{[s]}}\left[\boldsymbol{\phi}_s\right] = \boldsymbol{\phi}_s. \tag{A.27}$$

The stochastic matrix $\boldsymbol{M}_a$ is defined for every action a as

$$\boldsymbol{M}_a(i,j) = \mathbb{E}_{\omega_{[s]}}\left[\Pr\left\{s \xrightarrow{a} \boldsymbol{e}_j\right\}\right] \text{ with } \boldsymbol{\phi}_s = \boldsymbol{e}_i, \tag{A.28}$$

where $\Pr\left\{s \xrightarrow{a} \boldsymbol{e}_j\right\}$ is the probability of transitioning into the state partition associated with the latent state $\boldsymbol{e}_j$ and

$$\Pr\left\{s \xrightarrow{a} \boldsymbol{e}_j\right\} = p(s, a, [s_i]) \text{ such that } \forall s \in [s_i], \phi(s) = \boldsymbol{e}_j. \tag{A.29}$$

The identity in Equation (A.25) can be re-written as follows:

$$\begin{aligned}
\boldsymbol{\phi}_s^\top \boldsymbol{F}_a &= \boldsymbol{\phi}_s^\top + \gamma \mathbb{E}_{p,\pi}\left[\boldsymbol{\phi}_{s'} \boldsymbol{F}_{a'} | s, a\right] && \iff \\
\boldsymbol{\phi}_s^\top \boldsymbol{F}_a &= \boldsymbol{\phi}_s^\top + \gamma \mathbb{E}_{p}\left[\boldsymbol{\phi}_{s'} | s, a\right] \boldsymbol{F}^\pi && \iff \text{(by Lemma 2)} \\
\mathbb{E}_{\omega_{[s]}}\left[\boldsymbol{\phi}_s^\top \boldsymbol{F}_a\right] &= \mathbb{E}_{\omega_{[s]}}\left[\left(\boldsymbol{\phi}_s^\top + \gamma \mathbb{E}_{p}\left[\boldsymbol{\phi}_{s'} | s, a\right]\right) \boldsymbol{F}^\pi\right] && \iff \\
\boldsymbol{\phi}_s^\top \boldsymbol{F}_a &= \boldsymbol{\phi}_s^\top + \gamma \mathbb{E}_{\omega_{[s]}}\left[\mathbb{E}_{p}\left[\boldsymbol{\phi}_{s'} | s, a\right]\right] \boldsymbol{F}^\pi && \iff \text{(by (A.27))} \\
\boldsymbol{\phi}_s^\top \boldsymbol{F}_a &= \boldsymbol{\phi}_s^\top + \gamma \mathbb{E}_{\omega_{[s]}}\left[\sum_{j=1}^{n} \Pr\left\{s \xrightarrow{a} \boldsymbol{e}_j\right\} \boldsymbol{e}_j^\top\right] \boldsymbol{F}^\pi && \iff \\
\boldsymbol{\phi}_s^\top \boldsymbol{F}_a &= \boldsymbol{\phi}_s^\top + \gamma \underbrace{\left[\mathbb{E}_{\omega_{[s]}}\left[\Pr\left\{s \xrightarrow{a} \boldsymbol{e}_1\right\}\right], ..., \mathbb{E}_{\omega_{[s]}}\left[\Pr\left\{s \xrightarrow{a} \boldsymbol{e}_n\right\}\right]\right]}_{n\text{-dimensional row vector, because } \boldsymbol{e}_j \text{ is one-hot}} \boldsymbol{F}^\pi && \iff \\
\boldsymbol{\phi}_s^\top \boldsymbol{F}_a &= \boldsymbol{\phi}_s^\top + \gamma \left[\boldsymbol{M}_a(i,1), ..., \boldsymbol{M}_a(i,n)\right] \boldsymbol{F}^\pi && \iff \text{(by (A.28))} \\
\boldsymbol{\phi}_s^\top \boldsymbol{F}_a &= \boldsymbol{\phi}_s^\top + \gamma \boldsymbol{\phi}_s^\top \boldsymbol{M}_a \boldsymbol{F}^\pi && \iff \text{(by } \boldsymbol{\phi}_s = \boldsymbol{e}_i\text{)} \\
\boldsymbol{\phi}_s^\top \boldsymbol{F}_a &= \boldsymbol{\phi}_s^\top \left(\boldsymbol{I} + \gamma \boldsymbol{M}_a \boldsymbol{F}^\pi\right) && \text{(A.30)}
\end{aligned}$$

Equation (A.30) holds for any arbitrary state s and because each state is mapped to a one of the one-hot vectors $\{\boldsymbol{e}_1, ..., \boldsymbol{e}_n\}$,

$$\begin{aligned} \forall i \in \{1, ..., n\},\ \boldsymbol{e}_i^\top \boldsymbol{F}_a = \boldsymbol{e}_i^\top \left(\boldsymbol{I} + \gamma \boldsymbol{M}_a \boldsymbol{F}^\pi\right) \quad &\iff \\ \boldsymbol{F}_a = \boldsymbol{I} + \gamma \boldsymbol{M}_a \boldsymbol{F}^\pi. \end{aligned}$$

□

Lemma 4. *Consider a state representation $\phi : \mathcal{S} \to \{\boldsymbol{e}_1, ..., \boldsymbol{e}_n\}$, an arbitrary abstract policy $\pi \in \Pi_\phi$, an LSFM $\{\boldsymbol{F}_a, \boldsymbol{w}_a\}_{a \in \mathcal{A}}$ and LAM $\{\boldsymbol{M}_a, \boldsymbol{w}_a\}_{a \in \mathcal{A}}$ where each transition matrix $\boldsymbol{M}_a$ is stochastic. Then,*

$$\boldsymbol{F}_a = \boldsymbol{I} + \gamma \boldsymbol{M}_a \boldsymbol{F}^\pi \implies \exists \left(\boldsymbol{F}^\pi\right)^{-1} \text{ and } \left(\boldsymbol{F}^\pi\right)^{-1} = \boldsymbol{I} - \gamma \boldsymbol{M}^\pi, \tag{A.31}$$

where the matrix $\boldsymbol{M}^\pi$ is constructed as described in Lemma 2.

Proof of Lemma (4). By Lemma 2, one can write for any arbitrary i

$$\begin{aligned} \boldsymbol{e}_i^\top \boldsymbol{F}^\pi &= \mathbb{E}_\pi \left[\boldsymbol{e}_i^\top \boldsymbol{F}_a\right] && \iff \\ \boldsymbol{e}_i^\top \boldsymbol{F}^\pi &= \mathbb{E}_\pi \left[\boldsymbol{e}_i^\top \left(\boldsymbol{I} + \gamma \boldsymbol{M}_a \boldsymbol{F}^\pi\right)\right] && \iff \text{(by Lemma 3)} \\ \boldsymbol{e}_i^\top \boldsymbol{F}^\pi &= \boldsymbol{e}_i \boldsymbol{I} + \gamma \mathbb{E}_\pi \left[\boldsymbol{e}_i^\top \boldsymbol{M}_a\right] \boldsymbol{F}^\pi && \iff \\ \boldsymbol{e}_i^\top \boldsymbol{F}^\pi &= \boldsymbol{e}_i \boldsymbol{I} + \gamma \boldsymbol{e}_i^\top \boldsymbol{M}^\pi \boldsymbol{F}^\pi && \iff \text{(by Lemma 2)} \\ \boldsymbol{e}_i^\top \boldsymbol{F}^\pi &= \boldsymbol{e}_i \left(\boldsymbol{I} + \gamma \boldsymbol{M}^\pi \boldsymbol{F}^\pi\right) && \end{aligned} \tag{A.32}$$

Because Equation (A.32) holds for every i,

$$\begin{aligned} \boldsymbol{F}^\pi &= \boldsymbol{I} + \gamma \boldsymbol{M}^\pi \boldsymbol{F}^\pi && \iff \\ \boldsymbol{F}^\pi - \gamma \boldsymbol{M}^\pi \boldsymbol{F}^\pi &= \boldsymbol{I} && \iff \\ (\boldsymbol{I} - \gamma \boldsymbol{M}^\pi) \boldsymbol{F}^\pi &= \boldsymbol{I}. \end{aligned}$$

Because $\boldsymbol{M}^\pi$ is stochastic, it has a spectral radius of at most one and all its eigenvalues $\lambda_j \leq 1$.

Thus the matrix $\boldsymbol{I} - \gamma \boldsymbol{M}^{\pi}$ is invertible because

$$\det(\boldsymbol{I} - \gamma \boldsymbol{M}^{\pi}) \geq \det(\boldsymbol{I}) + (-\gamma)^n \det(\boldsymbol{M}^{\pi}) \geq 1 - \gamma^n \det(\boldsymbol{M}^{\pi}) = 1 - \gamma^n \underbrace{\prod_j \lambda_j}_{\leq 1} > 0. \tag{A.33}$$

Hence $\boldsymbol{F}^{\pi} = (\boldsymbol{I} - \gamma \boldsymbol{M}^{\pi})^{-1} \iff (\boldsymbol{F}^{\pi})^{-1} = \boldsymbol{I} - \gamma \boldsymbol{M}^{\pi}$. □

Using these lemmas, Theorem 2 can be proven.

Proof of Theorem 2. The proof is by reducing Equation (3.10) to Equation (3.8) using the previously established lemmas and then applying Theorem 1. Equation (3.10) can be re-written as follows:

$$\begin{aligned}
\boldsymbol{\phi}_s^{\top} \boldsymbol{F}_a &= \boldsymbol{\phi}_s^{\top} + \gamma \mathbb{E}_{p,\pi} \left[\boldsymbol{\phi}_{s'} \boldsymbol{F}_{a'} | s, a \right] && \Leftrightarrow \\
\boldsymbol{\phi}_s^{\top} \boldsymbol{F}_a &= \boldsymbol{\phi}_s^{\top} + \gamma \mathbb{E}_{p} \left[\boldsymbol{\phi}_{s'} | s, a \right] \boldsymbol{F}^{\pi} && \Leftrightarrow \text{(Lem. 2)} \\
\boldsymbol{\phi}_s^{\top} \boldsymbol{F}_a (\boldsymbol{I} - \gamma \boldsymbol{M}^{\pi}) &= \boldsymbol{\phi}_s^{\top} (\boldsymbol{I} - \gamma \boldsymbol{M}^{\pi}) + \gamma \mathbb{E}_{p} \left[\boldsymbol{\phi}_{s'} | s, a \right] \boldsymbol{F}^{\pi} (\boldsymbol{I} - \gamma \boldsymbol{M}^{\pi}) && \Leftrightarrow \text{(Lem. 4)} \\
\boldsymbol{\phi}_s^{\top} \boldsymbol{F}_a (\boldsymbol{I} - \gamma \boldsymbol{M}^{\pi}) &= \boldsymbol{\phi}_s^{\top} (\boldsymbol{I} - \gamma \boldsymbol{M}^{\pi}) + \gamma \mathbb{E}_{p} \left[\boldsymbol{\phi}_{s'} | s, a \right] \boldsymbol{F}^{\pi} \left(\boldsymbol{F}^{\pi} \right)^{-1} && \Leftrightarrow \text{(Lem. 4)} \\
\boldsymbol{\phi}_s^{\top} \boldsymbol{F}_a (\boldsymbol{I} - \gamma \boldsymbol{M}^{\pi}) &= \boldsymbol{\phi}_s^{\top} (\boldsymbol{I} - \gamma \boldsymbol{M}^{\pi}) + \gamma \mathbb{E}_{p} \left[\boldsymbol{\phi}_{s'} | s, a \right] && \Leftrightarrow \\
\boldsymbol{\phi}_s^{\top} (\boldsymbol{I} + \gamma \boldsymbol{M}_a \boldsymbol{F}^{\pi}) (\boldsymbol{I} - \gamma \boldsymbol{M}^{\pi}) &= \boldsymbol{\phi}_s^{\top} (\boldsymbol{I} - \gamma \boldsymbol{M}^{\pi}) + \gamma \mathbb{E}_{p} \left[\boldsymbol{\phi}_{s'} | s, a \right] && \Leftrightarrow \text{(Lem. 3)} \\
\boldsymbol{\phi}_s^{\top} \gamma \boldsymbol{M}_a \boldsymbol{F}^{\pi} (\boldsymbol{I} - \gamma \boldsymbol{M}^{\pi}) &= \gamma \mathbb{E}_{p} \left[\boldsymbol{\phi}_{s'} | s, a \right] && \Leftrightarrow \\
\boldsymbol{\phi}_s^{\top} \gamma \boldsymbol{M}_a &= \gamma \mathbb{E}_{p} \left[\boldsymbol{\phi}_{s'} | s, a \right] && \Leftrightarrow \text{(Lem. 4)} \\
\boldsymbol{\phi}_s^{\top} \boldsymbol{M}_a &= \mathbb{E}_{p} \left[\boldsymbol{\phi}_{s'} | s, a \right] && \text{(A.34)}
\end{aligned}$$

Using the reward condition stated in Equation (3.10) and Equation (A.34) Theorem 1 can be applied to conclude the proof.

To prove the last claim of Theorem 2, assume that

$$\boldsymbol{\phi}_s^{\top} \boldsymbol{F}_a = \boldsymbol{\phi}_s^{\top} + \gamma \mathbb{E}_{p,\pi} \left[\boldsymbol{\phi}_{s'} \boldsymbol{F}_{a'} | s, a \right] \tag{A.35}$$

for some policy $\pi \in \Pi_{\phi}$. Consider an arbitrary distinct policy $\tilde{\pi} \in \Pi_{\phi}$, then the fixed-point Eq. (A.35)

can be re-stated in terms of then policy $\tilde{\pi}$:

$$\boldsymbol{\phi}_s^\top \boldsymbol{F}_a = \boldsymbol{\phi}_s^\top + \gamma \mathbb{E}_{p,\pi} \left[\boldsymbol{\phi}_{s'} \boldsymbol{F}_{a'} | s, a \right] \qquad \Leftrightarrow \tag{A.36}$$

$$\boldsymbol{\phi}_s^\top \boldsymbol{M}_a = \mathbb{E}_p \left[\boldsymbol{\phi}_{s'} | s, a \right] \qquad \Leftrightarrow \text{(Eq. (A.34))} \tag{A.37}$$

$$\gamma \boldsymbol{\phi}_s^\top \boldsymbol{M}_a \boldsymbol{F}^{\tilde{\pi}} = \gamma \mathbb{E}_p \left[\boldsymbol{\phi}_{s'} | s, a \right] \boldsymbol{F}^{\tilde{\pi}} \qquad \Leftrightarrow \text{(multiply with } \boldsymbol{F}^{\tilde{\pi}} \text{ and } \gamma) \tag{A.38}$$

$$\boldsymbol{\phi}_s^\top + \gamma \boldsymbol{\phi}_s^\top \boldsymbol{M}_a \boldsymbol{F}^{\tilde{\pi}} = \boldsymbol{\phi}_s^\top + \gamma \mathbb{E}_p \left[\boldsymbol{\phi}_{s'} | s, a \right] \boldsymbol{F}^{\tilde{\pi}} \qquad \Leftrightarrow \text{(add } \boldsymbol{\phi}_s) \tag{A.39}$$

$$\boldsymbol{\phi}_s^\top (\boldsymbol{I} + \gamma \boldsymbol{M}_a \boldsymbol{F}^{\tilde{\pi}}) = \boldsymbol{\phi}_s^\top + \gamma \mathbb{E}_p \left[\boldsymbol{\phi}_{s'} | s, a \right] \boldsymbol{F}^{\tilde{\pi}} \qquad \Leftrightarrow \tag{A.40}$$

$$\boldsymbol{\phi}_s^\top \boldsymbol{F}_a = \boldsymbol{\phi}_s^\top + \gamma \mathbb{E}_p \left[\boldsymbol{\phi}_{s'} | s, a \right] \boldsymbol{F}^{\tilde{\pi}} \qquad \Leftrightarrow \text{(Lem. 3)} \tag{A.41}$$

$$\boldsymbol{\phi}_s^\top \boldsymbol{F}_a = \boldsymbol{\phi}_s^\top + \gamma \mathbb{E}_{p,\tilde{\pi}} \left[\boldsymbol{\phi}_{s'} \boldsymbol{F}_a | s, a \right] \qquad \Leftrightarrow \text{(Lem. 2)} \tag{A.42}$$

This argument shows that if Eq. (A.35) holds for a policy π, then Eq. (A.42) also holds for other arbitrary policy $\tilde{\pi}$. □

A.2.1 Approximate Reward-Predictive State Representations

This section presents formal proofs for Theorem 3 and 4. While the following proofs assume that the matrix $\overline{\boldsymbol{F}}$ is defined as stated in Equation (3.18), these proofs could be generalized to different definitions of $\overline{\boldsymbol{F}}$, assuming that the matrix $\overline{\boldsymbol{F}}$ is not a function of the state s and only depends on the matrices $\{\boldsymbol{F}_a\}_{a \in \mathcal{A}}$.

Lemma 5. *For an MDP, a state representation ϕ, a LSFM $\{\boldsymbol{F}_a, \boldsymbol{w}_a\}_{a \in \mathcal{A}}$ and a LAM $\{\boldsymbol{M}_a, \boldsymbol{w}_a\}_{a \in \mathcal{A}}$ where $\Delta = 0$,*

$$\varepsilon_p \leq \varepsilon_\psi \frac{1 + \gamma M}{\gamma}. \tag{A.43}$$

Proof of Lemma 5. The proof is by manipulating the definition of ε_ψ and using the fact that $\Delta = 0$ and $\boldsymbol{F}_a = \boldsymbol{I} + \gamma \boldsymbol{M}_a \overline{\boldsymbol{F}}$. Let $\overline{M} = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \boldsymbol{M}_a$, then

$$\overline{\boldsymbol{F}} = \boldsymbol{I} + \gamma \overline{\boldsymbol{M}} \overline{\boldsymbol{F}} \iff \boldsymbol{I} = (\boldsymbol{I} - \gamma \overline{\boldsymbol{M}}) \overline{\boldsymbol{F}} \tag{A.44}$$

Hence the square matrix $\boldsymbol{I} - \gamma \overline{\boldsymbol{M}}$ is a left inverse of the square matrix $\overline{\boldsymbol{F}}$. By associativity of matrix multiplication, $\boldsymbol{I} + \gamma \overline{\boldsymbol{M}}$ is also a right inverse of $\overline{\boldsymbol{F}}$ and $\overline{\boldsymbol{F}}(\boldsymbol{I} - \gamma \overline{\boldsymbol{M}})$.[1] Consequently, the norm of $\overline{\boldsymbol{F}}^{-1}$

[1] If $\overline{\boldsymbol{F}}^{-1} \overline{\boldsymbol{F}} = \boldsymbol{I}$ then $\overline{\boldsymbol{F}} = \overline{\boldsymbol{F}} \boldsymbol{I} = \overline{\boldsymbol{F}} (\overline{\boldsymbol{F}}^{-1} \overline{\boldsymbol{F}}) = (\overline{\boldsymbol{F}} \overline{\boldsymbol{F}}^{-1}) \overline{\boldsymbol{F}}$. If $\overline{\boldsymbol{F}}^{-1} \overline{\boldsymbol{F}} \neq \boldsymbol{I}$ were true, then it would contradict $\overline{\boldsymbol{F}} = (\overline{\boldsymbol{F}} \overline{\boldsymbol{F}}^{-1}) \overline{\boldsymbol{F}}$. Hence $\overline{\boldsymbol{F}} \overline{\boldsymbol{F}}^{-1} = \boldsymbol{I}$ and the right inverse exists.

can be bounded with

$$\left|\left|\overline{\boldsymbol{F}}^{-1}\right|\right| = \left|\left|(\boldsymbol{I} - \gamma\overline{\boldsymbol{F}})\right|\right| \leq 1 + \gamma M. \tag{A.45}$$

For an arbitrary state and action pair s, a,

$$\begin{aligned}
\boldsymbol{\delta}_{s,a}^\top &= \boldsymbol{\phi}_s^\top + \gamma\mathbb{E}_p\left[\boldsymbol{\phi}_{s'}^\top\overline{\boldsymbol{F}}\middle|s,a\right] - \boldsymbol{\phi}_s^\top\boldsymbol{F}_a && \text{(A.46)}\\
&= \boldsymbol{\phi}_s^\top + \gamma\mathbb{E}_p\left[\boldsymbol{\phi}_{s'}^\top\overline{\boldsymbol{F}}\middle|s,a\right] - \gamma\boldsymbol{\phi}_s^\top\boldsymbol{M}_a\overline{\boldsymbol{F}} + \gamma\boldsymbol{\phi}_s^\top\boldsymbol{M}_a\overline{\boldsymbol{F}} - \boldsymbol{\phi}_s^\top\boldsymbol{F}_a && \text{(A.47)}\\
&= \boldsymbol{\phi}_s^\top + \gamma\left(\mathbb{E}_p\left[\boldsymbol{\phi}_{s'}^\top\middle|s,a\right] - \boldsymbol{\phi}_s^\top\boldsymbol{M}_a\right)\overline{\boldsymbol{F}} + \gamma\boldsymbol{\phi}_s^\top\boldsymbol{M}_a\overline{\boldsymbol{F}} - \boldsymbol{\phi}_s^\top\boldsymbol{F}_a && \text{(A.48)}
\end{aligned}$$

Let $\boldsymbol{\varepsilon}_{s,a}^\top = \mathbb{E}_p\left[\boldsymbol{\phi}_{s'}^\top\middle|s,a\right] - \boldsymbol{\phi}_s^\top\boldsymbol{M}_a$. Re-arranging the identity in (A.48) results in

$$\begin{aligned}
\gamma\boldsymbol{\varepsilon}_{s,a}^\top\overline{\boldsymbol{F}} &= \boldsymbol{\delta}_{s,a}^\top - \boldsymbol{\phi}_s^\top - \gamma\boldsymbol{\phi}_s^\top\boldsymbol{M}_a\overline{\boldsymbol{F}} + \boldsymbol{\phi}_s^\top\boldsymbol{F}_a && \iff\\
\gamma\boldsymbol{\varepsilon}_{s,a}^\top &= \boldsymbol{\delta}_{s,a}^\top\overline{\boldsymbol{F}}^{-1} - \boldsymbol{\phi}_s^\top\overline{\boldsymbol{F}}^{-1} - \gamma\boldsymbol{\phi}_s^\top\boldsymbol{M}_a + \boldsymbol{\phi}_s^\top\boldsymbol{F}_a\overline{\boldsymbol{F}}^{-1} && \iff \text{(by (A.44))}\\
\gamma\boldsymbol{\varepsilon}_{s,a}^\top &= \boldsymbol{\delta}_{s,a}^\top\overline{\boldsymbol{F}}^{-1} - \boldsymbol{\phi}_s^\top\overline{\boldsymbol{F}}^{-1} - \gamma\boldsymbol{\phi}_s^\top\boldsymbol{M}_a + \boldsymbol{\phi}_s^\top(\boldsymbol{I} + \gamma\boldsymbol{M}_a\overline{\boldsymbol{F}})\overline{\boldsymbol{F}}^{-1} && \iff \text{(by } \Delta = 0)\\
\gamma\boldsymbol{\varepsilon}_{s,a}^\top &= \boldsymbol{\delta}_{s,a}^\top\overline{\boldsymbol{F}}^{-1} - \boldsymbol{\phi}_s^\top\overline{\boldsymbol{F}}^{-1} - \gamma\boldsymbol{\phi}_s^\top\boldsymbol{M}_a + \boldsymbol{\phi}_s^\top\overline{\boldsymbol{F}}^{-1} + \gamma\boldsymbol{\phi}_s^\top\boldsymbol{M}_a && \iff\\
\gamma\boldsymbol{\varepsilon}_{s,a}^\top &= \boldsymbol{\delta}_{s,a}^\top\overline{\boldsymbol{F}}^{-1} && \iff\\
\gamma\left|\left|\boldsymbol{\varepsilon}_{s,a}^\top\right|\right| &\leq \left|\left|\boldsymbol{\delta}_{s,a}^\top\right|\right|\left|\left|\overline{\boldsymbol{F}}^{-1}\right|\right| && \iff\\
\gamma\left|\left|\boldsymbol{\varepsilon}_{s,a}^\top\right|\right| &\leq \varepsilon_\psi\left|\left|\overline{\boldsymbol{F}}^{-1}\right|\right| && \iff \text{(by (3.19))}\\
\left|\left|\boldsymbol{\varepsilon}_{s,a}^\top\right|\right| &\leq \varepsilon_\psi(1+\gamma M)/\gamma && \iff \text{(by (A.45))} \qquad \text{(A.49)}
\end{aligned}$$

Note that the bound in Equation (A.49) does not depend on the state and action pair s, a and thus

$$\forall s,a,\ \left|\left|\mathbb{E}_p\left[\boldsymbol{\phi}_{s'}^\top\middle|s,a\right] - \boldsymbol{\phi}_s^\top\boldsymbol{M}_a\right|\right| \leq \varepsilon_\psi(1+\gamma M)/\gamma \implies \varepsilon_p \leq \varepsilon_\psi(1+\gamma M)/\gamma. \tag{A.50}$$

□

The following lemma is a restatement of Lemma 1 in the main paper.

Lemma 6. *For an MDP, a state representation ϕ, a LSFM $\{\boldsymbol{F}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$ and a LAM $\{\boldsymbol{M}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$ where $\Delta \geq 0$,*

$$\varepsilon_p \leq \varepsilon_\psi\frac{1+\gamma M}{\gamma} + C_{\gamma,M,N}\Delta, \tag{A.51}$$

where $C_{\gamma,M,N} = \frac{(1+\gamma)(1+\gamma M)N}{\gamma(1-\gamma M)}$

Proof. The proof reuses and extends the bound shown in Lemma 5. Using the LAM $\{\boldsymbol{M}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$, construct an LSFM $\{\boldsymbol{F}^*_a, \boldsymbol{w}^*_a\}_{a\in\mathcal{A}}$ such that

$$\boldsymbol{F}^*_a = \boldsymbol{I} + \gamma \boldsymbol{M}_a \underbrace{\frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}} \boldsymbol{F}^*_a}_{=\overline{\boldsymbol{F}}^*}. \tag{A.52}$$

If

$$\varepsilon^*_\psi = \sup_{s,a} \left| \boldsymbol{\phi}_s^\top + \gamma \mathbb{E}_p \left[\boldsymbol{\phi}_{s'}^\top \overline{\boldsymbol{F}}^* \middle| s, a \right] - \boldsymbol{\phi}_s^\top \boldsymbol{F}^*_a \right|, \tag{A.53}$$

then

$$\varepsilon_p \leq \varepsilon^*_\psi \frac{1+\gamma M}{\gamma}, \tag{A.54}$$

by Lemma 5. By linearity of the expectation operator, the SF-error for the LSFM $\{\boldsymbol{F}^*_a, \boldsymbol{w}^*_a\}_{a\in\mathcal{A}}$ and LSFM $\{\boldsymbol{F}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$ can be founded for any arbitrary state and action pair s, a with

$$\left|\left| \underbrace{\left(\boldsymbol{\phi}_s^\top + \gamma \mathbb{E}_p \left[\boldsymbol{\phi}_{s'}^\top \overline{\boldsymbol{F}}^* \middle| s, a \right] - \boldsymbol{\phi}_s^\top \boldsymbol{F}^*_a \right)}_{=\boldsymbol{\delta}^*_{s,a}} - \underbrace{\left(\boldsymbol{\phi}_s^\top + \gamma \mathbb{E}_p \left[\boldsymbol{\phi}_{s'}^\top \overline{\boldsymbol{F}} \middle| s, a \right] - \boldsymbol{\phi}_s^\top \boldsymbol{F}_a \right)}_{\boldsymbol{\delta}_{s,a}} \right|\right| \tag{A.55}$$

$$\leq \gamma N \left(\overline{\boldsymbol{F}}^* - \overline{\boldsymbol{F}} \right) + N \left(\boldsymbol{F}^*_a - \boldsymbol{F}_a \right). \tag{A.56}$$

As stated in Equation (A.55), the SF errors for the LSFM $\{\boldsymbol{F}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$ are defined as $\boldsymbol{\delta}_{s,a}$ and for the LSFM $\{\boldsymbol{F}^*_a, \boldsymbol{w}^*_a\}_{a\in\mathcal{A}}$ as $\boldsymbol{\delta}^*_{s,a}$. Because the LSFM $\{\boldsymbol{F}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$ has a $\Delta > 0$, we define

$$\overline{\boldsymbol{\Delta}} = \frac{1}{|\mathcal{A}|}\sum_a \boldsymbol{I} + \gamma \boldsymbol{M}_a \overline{\boldsymbol{F}} - \boldsymbol{F}_a \tag{A.57}$$

$$= \boldsymbol{I} + \gamma \overline{\boldsymbol{M}\boldsymbol{F}} - \overline{\boldsymbol{F}}. \tag{A.58}$$

By Equation (A.57), $||\overline{\boldsymbol{\Delta}}|| \leq \frac{1}{|\mathcal{A}|}\sum_a ||\boldsymbol{I} + \gamma \boldsymbol{M}_a \overline{\boldsymbol{F}} - \boldsymbol{F}_a|| \leq \Delta$ (by triangle inequality). Reusing Equation (A.58) one can write,

$$\overline{\boldsymbol{F}}^* - \overline{\boldsymbol{F}} = \boldsymbol{I} + \gamma \overline{\boldsymbol{M}\boldsymbol{F}}^* - \boldsymbol{I} - \gamma \overline{\boldsymbol{M}\boldsymbol{F}} + \overline{\boldsymbol{\Delta}} \tag{A.59}$$

$$= \gamma \overline{\boldsymbol{M}} (\overline{\boldsymbol{F}}^* - \overline{\boldsymbol{F}}) + \overline{\boldsymbol{\Delta}} \tag{A.60}$$

$$\iff ||\overline{\boldsymbol{F}}^* - \overline{\boldsymbol{F}}|| \leq \gamma M ||\overline{\boldsymbol{F}}^* - \overline{\boldsymbol{F}}|| + \Delta \tag{A.61}$$

$$\iff ||\overline{\boldsymbol{F}}^* - \overline{\boldsymbol{F}}|| \leq \frac{\Delta}{1-\gamma M} \tag{A.62}$$

Similarly, define

$$\boldsymbol{\Delta}_a = \boldsymbol{I} + \gamma \boldsymbol{M}_a \overline{\boldsymbol{F}} - \boldsymbol{F}_a, \tag{A.63}$$

then,

$$\begin{aligned}
\boldsymbol{F}^*_a - \boldsymbol{F}_a &= \boldsymbol{I} + \gamma \boldsymbol{M}_a \overline{\boldsymbol{F}}^* - \boldsymbol{I} - \gamma \boldsymbol{M}_a \overline{\boldsymbol{F}} + \boldsymbol{\Delta}_a && \text{(A.64)} \\
&= \gamma \boldsymbol{M}_a (\overline{\boldsymbol{F}}^* - \overline{\boldsymbol{F}}) + \boldsymbol{\Delta}_a && \text{(A.65)} \\
\iff ||\boldsymbol{F}^*_a - \boldsymbol{F}_a|| &\leq \gamma M ||\overline{\boldsymbol{F}}^* - \overline{\boldsymbol{F}}|| + \Delta && \text{(A.66)} \\
&\leq \gamma M \frac{\Delta}{1 - \gamma M} + \Delta && \text{(A.67)} \\
&= \frac{\Delta}{1 - \gamma M} && \text{(A.68)}
\end{aligned}$$

Substituting lines (A.62) and (A.68) into (A.56),

$$\left|\left|\boldsymbol{\delta}^*_{s,a} - \boldsymbol{\delta}_{s,a}\right|\right| \leq \frac{(1+\gamma) N \Delta}{1 - \gamma M}. \tag{A.69}$$

For both LSFM $\{\boldsymbol{F}_a, \boldsymbol{w}_a\}_{a \in \mathcal{A}}$ and $\{\boldsymbol{F}^*_a, \boldsymbol{w}^*_a\}_{a \in \mathcal{A}}$, the worst case SF prediction errors ε_ψ and ε^*_ψ are defined as

$$\varepsilon_\psi = \sup_{s,a} ||\boldsymbol{\delta}_{s,a}|| \text{ and } \varepsilon^*_\psi = \sup_{s,a} ||\boldsymbol{\delta}^*_{s,a}||. \tag{A.70}$$

To find a bound on $|\varepsilon_\psi - \varepsilon^*_\phi|$, the maximizing state and action pairs are defined as

$$s_{\text{sup}}, a_{\text{sup}} = \arg\sup_{s,a} ||\boldsymbol{\delta}_{s,a}|| \text{ and } s^*_{\text{sup}}, a^*_{\text{sup}} = \arg\sup_{s,a} ||\boldsymbol{\delta}^*_{s,a}||. \tag{A.71}$$

If $(s_{\text{sup}}, a_{\text{sup}}) = (s^*_{\text{sup}}, a^*_{\text{sup}})$ then

$$\left|\varepsilon_\psi - \varepsilon^*_\psi\right| \leq \frac{(1+\gamma) N \Delta}{1 - \gamma M}. \qquad \text{(by (A.69))} \tag{A.72}$$

If $(s_{\text{sup}}, a_{\text{sup}}) \neq (s^*_{\text{sup}}, a^*_{\text{sup}})$ and $\varepsilon_\psi \geq \varepsilon^*_\psi$, then

$$\begin{aligned}
\varepsilon_\psi - \varepsilon^*_\psi &= \left|\left|\boldsymbol{\delta}_{s_{\text{sup}},a_{\text{sup}}}\right|\right| - \left|\left|\boldsymbol{\delta}^*_{s^*_{\text{sup}},a^*_{\text{sup}}}\right|\right| && && \text{(A.73)}\\
&\leq \left|\left|\boldsymbol{\delta}_{s_{\text{sup}},a_{\text{sup}}}\right|\right| - \left|\left|\boldsymbol{\delta}^*_{s_{\text{sup}},a_{\text{sup}}}\right|\right| && \text{(by (A.71))} && \text{(A.74)}\\
&\leq \left|\left|\boldsymbol{\delta}_{s_{\text{sup}},a_{\text{sup}}} - \boldsymbol{\delta}^*_{s_{\text{sup}},a_{\text{sup}}}\right|\right| && \text{(by inv. triangle in eq.)} && \text{(A.75)}\\
&\leq \frac{(1+\gamma)N\Delta}{1-\gamma M}. && \text{(by (A.69))} && \text{(A.76)}
\end{aligned}$$

If $(s_{\text{sup}}, a_{\text{sup}}) \neq (s^*_{\text{sup}}, a^*_{\text{sup}})$ and $\varepsilon^*_\psi \geq \varepsilon_\psi$, then

$$\begin{aligned}
\varepsilon^*_\psi - \varepsilon_\psi &= \left|\left|\boldsymbol{\delta}^*_{s^*_{\text{sup}},a^*_{\text{sup}}}\right|\right| - \left|\left|\boldsymbol{\delta}_{s_{\text{sup}},a_{\text{sup}}}\right|\right| && && \text{(A.77)}\\
&\leq \left|\left|\boldsymbol{\delta}^*_{s^*_{\text{sup}},a^*_{\text{sup}}}\right|\right| - \left|\left|\boldsymbol{\delta}_{s^*_{\text{sup}},a^*_{\text{sup}}}\right|\right| && \text{(by (A.71))} && \text{(A.78)}\\
&\leq \left|\left|\boldsymbol{\delta}^*_{s^*_{\text{sup}},a^*_{\text{sup}}} - \boldsymbol{\delta}_{s^*_{\text{sup}},a^*_{\text{sup}}}\right|\right| && \text{(by inv. triangle ineq.)} && \text{(A.79)}\\
&\leq \frac{(1+\gamma)N\Delta}{1-\gamma M}. && \text{(by (A.69))} && \text{(A.80)}
\end{aligned}$$

By lines (A.72), (A.76), and (A.80),

$$\left|\varepsilon_\psi - \varepsilon^*_\psi\right| \leq \frac{(1+\gamma)N\Delta}{1-\gamma M} \implies \varepsilon^*_\psi \leq \varepsilon_\psi + \frac{(1+\gamma)N\Delta}{1-\gamma M}. \tag{A.81}$$

Substituting (A.81) into (A.54) results in the desired bound:

$$\varepsilon_p \leq \varepsilon^*_\psi \frac{1+\gamma M}{\gamma} \leq \left(\varepsilon_\psi + \frac{(1+\gamma)N\Delta}{1-\gamma M}\right)\frac{1+\gamma M}{\gamma} = \varepsilon_\psi \frac{1+\gamma M}{\gamma} + \frac{(1+\gamma)(1+\gamma M)N}{\gamma(1-\gamma M)}\Delta. \tag{A.82}$$

□

Using these lemmas, Theorem 3 can be proven.

Proof of Theorem 3. The proof is by induction on the sequence length T.

Base Case: For $T = 1$,

$$\left|\boldsymbol{\phi}_s^\top \boldsymbol{w}_{a_1} - \mathbb{E}_p\left[r_1 | s, a_1\right]\right| = \left|\boldsymbol{\phi}_s^\top \boldsymbol{w}_{a_1} - r(s, a_1)\right| \leq \varepsilon_r. \tag{A.83}$$

Induction Step: Assume that the bound (3.22) holds for T, then for $T+1$,

$$\left|\boldsymbol{\phi}_s^\top \boldsymbol{M}_{a_1} \cdots \boldsymbol{M}_{a_T} \boldsymbol{w}_{a_{T+1}} - \mathbb{E}_p\left[r_{T+1} | s, a_1, ..., a_{T+1}\right]\right| \tag{A.84}$$

$$= \Big|\boldsymbol{\phi}_s^\top \boldsymbol{M}_{a_1} \cdots \boldsymbol{M}_{a_T} \boldsymbol{w}_{a_{T+1}} - \mathbb{E}_p\left[\boldsymbol{\phi}_{s_2}^\top \boldsymbol{M}_{a_2} \cdots \boldsymbol{M}_{a_T} \boldsymbol{w}_{a_{T+1}} \big| s, a_1\right]$$

$$+ \mathbb{E}_p\left[\boldsymbol{\phi}_{s_2}^\top \boldsymbol{M}_{a_2} \cdots \boldsymbol{M}_{a_T} \boldsymbol{w}_{a_{T+1}} \big| s, a_1\right] - \mathbb{E}_p\left[r_{T+1} | s, a_1, ..., a_{T+1}\right]\Big| \tag{A.85}$$

$$\leq \Big|\left(\boldsymbol{\phi}_s^\top \boldsymbol{M}_{a_1} - \mathbb{E}_p\left[\boldsymbol{\phi}_{s_2}^\top \big| s, a_1\right]\right) \boldsymbol{M}_{a_2} \cdots \boldsymbol{M}_{a_T} \boldsymbol{w}_{a_{T+1}}\Big|$$

$$+ \Big|\mathbb{E}_p\left[\boldsymbol{\phi}_{s_2}^\top \boldsymbol{M}_{a_2} \cdots \boldsymbol{M}_{a_T} \boldsymbol{w}_{a_{T+1}} \big| s, a_1\right] - \mathbb{E}_p\left[r_{T+1} | s, a_1, ..., a_{T+1}\right]\Big| \tag{A.86}$$

$$\leq \Big|\Big|\boldsymbol{\phi}_s^\top \boldsymbol{M}_{a_1} - \mathbb{E}_p\left[\boldsymbol{\phi}_{s_2}^\top \big| s, a_1\right]\Big|\Big| \cdot \Big|\Big|\boldsymbol{M}_{a_2} \cdots \boldsymbol{M}_{a_T} \boldsymbol{w}_{a_{T+1}}\Big|\Big|$$

$$+ \Big|\mathbb{E}_p\left[\boldsymbol{\phi}_{s_2}^\top \boldsymbol{M}_{a_2} \cdots \boldsymbol{M}_{a_T} \boldsymbol{w}_{a_{T+1}} \big| s, a_1\right] - \mathbb{E}_p\left[r_{T+1} | s, a_1, ..., a_{T+1}\right]\Big|$$

$$\leq \Big|\Big|\boldsymbol{\phi}_s^\top \boldsymbol{M}_{a_1} - \mathbb{E}_p\left[\boldsymbol{\phi}_{s_2}^\top \big| s, a_1\right]\Big|\Big| \cdot \Big|\Big|\boldsymbol{M}_{a_2} \cdots \boldsymbol{M}_{a_T} \boldsymbol{w}_{a_{T+1}}\Big|\Big|$$

$$+ \Big|\mathbb{E}_p\left[\boldsymbol{\phi}_{s_2}^\top \boldsymbol{M}_{a_2} \cdots \boldsymbol{M}_{a_T} \boldsymbol{w}_{a_{T+1}} - \mathbb{E}_p\left[r_{T+1} | s_1, a_2 ..., a_{T+1}\right] \big| s, a_1\right]\Big| \tag{A.87}$$

$$\leq \varepsilon_p M^{T-1} W$$

$$+ \varepsilon_p \sum_{t=1}^{T-1} M^t W + \varepsilon_r \tag{A.88}$$

$$= \varepsilon_p \sum_{t=1}^{(T+1)-1} M^t W + \varepsilon_r. \tag{A.89}$$

□

Theorem 6. *For an MDP, state representation $\phi : \mathcal{S} \to \mathbb{R}^n$, and for all $T \geq 1, s, a_1, ..., a_T$,*

$$\left|\boldsymbol{\phi}_s^\top \boldsymbol{M}_{a_1} \cdots \boldsymbol{M}_{a_{T-1}} \boldsymbol{w}_{a_T} - \mathbb{E}_p\left[r_T | s, a_1, ..., a_T\right]\right| \leq \left(\varepsilon_\psi \frac{1+\gamma M}{\gamma} + C_{\gamma,M,N}\Delta\right) \sum_{t=1}^{T-1} M^t W + \varepsilon_r.$$

Proof of Theorem 6. The proof is by reusing the bound in Theorem 3 and substituting ε_p with the bound presented in Lemma 1. □

Theorem 4, which is stated in the main paper, can be proven as follows.

Proof of Theorem 4. The value error term can be upper-bounded with

$$\left|V^\pi(s) - \boldsymbol{\phi}_s^\top \boldsymbol{v}^\pi\right| \leq \sum_{a\in\mathcal{A}} \pi(s,a) \left|r(s,a) + \gamma \mathbb{E}_p\left[V^\pi(s')|s,a\right] - \boldsymbol{\phi}_s^\top \boldsymbol{w}_a - \gamma \boldsymbol{\phi}_s^\top \boldsymbol{M}_a \boldsymbol{v}^\pi\right| \tag{A.90}$$

$$\leq \sum_{a\in\mathcal{A}} \pi(s,a) \left|r(s,a) - \boldsymbol{\phi}_s^\top \boldsymbol{w}_a\right| + \gamma \left|\mathbb{E}_p\left[V^\pi(s')|s,a\right] - \boldsymbol{\phi}_s^\top \boldsymbol{M}_a \boldsymbol{v}^\pi\right| \tag{A.91}$$

The second term in Equation (A.91) is bounded by

$$\begin{aligned}
&\left|\mathbb{E}_p\left[V^\pi(s')|s,a\right] - \boldsymbol{\phi}_s^\top \boldsymbol{M}_a \boldsymbol{v}^\pi\right| \\
&= \left|\mathbb{E}_p\left[V^\pi(s')|s,a\right] - \mathbb{E}_p\left[\boldsymbol{\phi}_{s'}^\top \boldsymbol{v}^\pi \middle| s,a\right] + \mathbb{E}_p\left[\boldsymbol{\phi}_{s'}^\top \boldsymbol{v}^\pi \middle| s,a\right] - \boldsymbol{\phi}_s^\top \boldsymbol{M}_a \boldsymbol{v}^\pi\right| \\
&= \sup_s \left|V^\pi(s) - \boldsymbol{\phi}_s^\top \boldsymbol{v}^\pi\right| + \left|\mathbb{E}_p\left[\boldsymbol{\phi}_{s'}^\top \boldsymbol{v}^\pi \middle| s,a\right] - \boldsymbol{\phi}_s^\top \boldsymbol{M}_a \boldsymbol{v}^\pi\right| \\
&= \sup_s \left|V^\pi(s) - \boldsymbol{\phi}_s^\top \boldsymbol{v}^\pi\right| + \left|\left|\mathbb{E}_p\left[\boldsymbol{\phi}_{s'}^\top \middle| s,a\right] - \boldsymbol{\phi}_s^\top \boldsymbol{M}_a\right|\right| ||\boldsymbol{v}^\pi|| \\
&= \sup_s \left|V^\pi(s) - \boldsymbol{\phi}_s^\top \boldsymbol{v}^\pi\right| + \varepsilon_p ||\boldsymbol{v}^\pi||
\end{aligned} \tag{A.92}$$

Substituting (A.92) into (A.91) results in

$$\left|V^\pi(s) - \boldsymbol{\phi}_s^\top \boldsymbol{v}^\pi\right| \leq \sum_{a\in\mathcal{A}} \pi(s,a) \left(\left|r(s,a) - \boldsymbol{\phi}_s^\top \boldsymbol{w}_a\right| + \gamma \left(\sup_s \left|V^\pi(s) - \boldsymbol{\phi}_s^\top \boldsymbol{v}^\pi\right| + \varepsilon_p ||\boldsymbol{v}^\pi||\right)\right). \tag{A.93}$$

Let $B = \sup_s \left|V^\pi(s) - \boldsymbol{\phi}_s^\top \boldsymbol{v}^\pi\right|$, then

$$\begin{aligned}
\left|V^\pi(s) - \boldsymbol{\phi}_s^\top \boldsymbol{v}^\pi\right| &\leq \sum_{a\in\mathcal{A}} \pi(s,a)(\varepsilon_r + \gamma\left(B + \varepsilon_p ||\boldsymbol{v}^\pi||\right)) \\
&= \varepsilon_r + \gamma B + \gamma \varepsilon_p ||\boldsymbol{v}^\pi||
\end{aligned} \tag{A.94}$$

The bound in Equation (A.94) does not depend on any particular state and action pair s, a and thus

$$\begin{aligned}
\forall s,a,\ \left|V^\pi(s) - \boldsymbol{\phi}_s^\top \boldsymbol{v}^\pi\right| \leq \varepsilon_r + \gamma B + \gamma\varepsilon_p ||\boldsymbol{v}^\pi|| &\implies B \leq \varepsilon_r + \gamma B + \gamma \varepsilon_p ||\boldsymbol{v}^\pi|| \\
&\implies B \leq \frac{\varepsilon_r + \gamma \varepsilon_\psi ||\boldsymbol{v}^\pi||}{1-\gamma}.
\end{aligned} \tag{A.95}$$

To bound the Q-value function,

$$\left|Q^\pi(s,a) - \boldsymbol{\phi}_s^\top \boldsymbol{q}_a\right| \leq \left|r(s,a) + \gamma \mathbb{E}_p\left[V^\pi(s')|s,a\right] - \boldsymbol{\phi}_s^\top \boldsymbol{w}_a - \gamma \boldsymbol{\phi}_s^\top \boldsymbol{M}_a \boldsymbol{v}^\pi\right|, \tag{A.96}$$

which is similar to Equation (A.91) and the proof proceeds in the same way. The LSFM bound

$$\frac{\varepsilon_r + \gamma\varepsilon_p \, ||\boldsymbol{v}^\pi||}{1-\gamma} \leq \frac{\varepsilon_r + \varepsilon_\psi(1+\gamma M) \, ||\boldsymbol{v}^\pi|| + \gamma C_{\gamma,M,N}\Delta \, ||\boldsymbol{v}^\pi||}{1-\gamma} \tag{A.97}$$

follows by Lemma 1. □

A.2.2 Bound on Error Term Δ

The following proposition formally proofs the bound presented in Equation (3.25).

Proposition 2. *For a data set* $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}_{i=1}^D$,

$$\Delta \leq \max_a ||\boldsymbol{\Phi}_a^+||_2^2 \mathcal{L}_\psi, \tag{A.98}$$

where each row of $\boldsymbol{\Phi}_a$ *is set to a row-vector* $\boldsymbol{\phi}_s$ *for a transition* $(s, a, r, s') \in \mathcal{D}$ *that uses action* a*, and* $\boldsymbol{\Phi}_a^+$ *is the pseudo-inverse of* $\boldsymbol{\Phi}_a$*.*

Proof of Proposition 2. For a data set $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}_{i=1}^D$, construct the matrix $\boldsymbol{\Phi}_a$ and similarly construct the matrix $\boldsymbol{\Phi}_a'$ where each row of $\boldsymbol{\Phi}_a'$ is set to a row-vector $\boldsymbol{\phi}_{s'}$ for a transition $(s, a, r, s') \in \mathcal{D}$ that uses action a. The transition matrix of a LAM can be obtained using a least squares regression and

$$\boldsymbol{M}_a = \arg\min_{\boldsymbol{M}} ||\boldsymbol{\Phi}_a \boldsymbol{M} - \boldsymbol{\Phi}_a'||_2^2 \implies \boldsymbol{M}_a = \boldsymbol{\Phi}_a^+ \boldsymbol{\Phi}_a', \tag{A.99}$$

where $\boldsymbol{\Phi}_a^+$ is the pseudo-inverse of $\boldsymbol{\Phi}_a$. Using this notation, one can write

$$\boldsymbol{\Phi}_a + \gamma \boldsymbol{\Phi}_a' \overline{\boldsymbol{F}} - \boldsymbol{\Phi}_a \boldsymbol{F}_a = \boldsymbol{L}_a \iff \tag{A.100}$$

$$\boldsymbol{\Phi}_a^+ \boldsymbol{\Phi}_a + \gamma \boldsymbol{\Phi}_a^+ \boldsymbol{\Phi}_a' \overline{\boldsymbol{F}} - \boldsymbol{\Phi}_a^+ \boldsymbol{\Phi}_a \boldsymbol{F}_a = \boldsymbol{\Phi}_a^+ \boldsymbol{L}_a \iff \tag{A.101}$$

$$\boldsymbol{I} + \gamma \boldsymbol{M}_a \overline{\boldsymbol{F}} - \boldsymbol{F}_a = \boldsymbol{\Phi}_a^+ \boldsymbol{L}_a \iff \tag{A.102}$$

$$||\boldsymbol{I} + \gamma \boldsymbol{M}_a \overline{\boldsymbol{F}} - \boldsymbol{F}_a||_2^2 \leq ||\boldsymbol{\Phi}_a^+||_2^2 ||\boldsymbol{L}_a||_2^2. \tag{A.103}$$

Note that $\mathcal{L}_\psi = \sum_{a \in \mathcal{A}} \boldsymbol{L}_a$, and thus

$$\Delta = \max_a ||\boldsymbol{I} + \gamma \boldsymbol{M}_a \overline{\boldsymbol{F}} - \boldsymbol{F}_a||_2^2 \leq \max_a ||\boldsymbol{\Phi}_a^+||_2^2 \mathcal{L}_\psi. \tag{A.104}$$

□

A.3 Convergence Proofs of Clustering Algorithm

A.3.1 Norm Identities

Before presenting the convergence proof, we first define the following quantities and prove identities of their norms.

We now define the empirical partition-to-partition transition matrices as counting the empirical probabilities of transitioning from a partition i to a partition j. This transition matrix is equivalent to finding the transition matrices $\{\boldsymbol{M}_a\}_{a\in\mathcal{A}}$ of a LAM and then computing $\overline{\boldsymbol{M}} = \frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}}\boldsymbol{M}_a$.

Definition 7 (Empirical Partition Transition Matrix)**.** *For a transition dataset $\mathcal{D}$ and cluster function c, we define an empirical partition-to-partition transition matrix $\boldsymbol{M}$ as*

$$\overline{\boldsymbol{M}}(i,j) = \frac{|\{(s,a,r,s')\in\mathcal{D}|c(s)=i\wedge c(s')=j\}|}{|\{(s,a,r,s')\in\mathcal{D}|c(s)=i\}|}. \tag{A.105}$$

Note that the matrix $\overline{\boldsymbol{M}}(i,j)$ is left-stochastic and all rows sum to one.

For the following lemma we consider the L1 matrix norm which is defined for a matrix

$$||\boldsymbol{A}||_1 = \max_j \sum_i |\boldsymbol{A}(i,j)|. \tag{A.106}$$

Lemma 7. *For a transition dataset $\mathcal{D}$ and cluster function c, we can compute the L1 matrix norm of the normalized SF matrix $(1-\gamma)\overline{\boldsymbol{F}}$ with*

$$\left|\left|(1-\gamma)\overline{\boldsymbol{F}}^\top\right|\right|_1 = 1. \tag{A.107}$$

Further, the L1 matrix norm of $1-\gamma$ scaled transposed inverse of $\overline{\boldsymbol{F}}$ is

$$\left|\left|(\overline{\boldsymbol{F}}^{-1})^\top/(1-\gamma)\right|\right|_1 = 1. \tag{A.108}$$

Note that $\overline{\boldsymbol{F}} = (\boldsymbol{I}-\gamma\overline{\boldsymbol{M}})^{-1}$.

Proof. The proof of both identities proceeds by showing that all rows sum to one. Consider the left-stochastic transition matrix $\overline{\boldsymbol{M}}$ (for some cluster function c and dataset $\mathcal{D}$). Each row of $\overline{\boldsymbol{M}}$

sums to one. Consequently, each column of $\overline{\boldsymbol{M}}^\top$ sums to one and

$$\left|\left|\overline{\boldsymbol{M}}^\top\right|\right|_1 = \max_j \sum_i \left|\overline{\boldsymbol{M}}^\top(i,j)\right| = \max_i \underbrace{\sum_j \left|\overline{\boldsymbol{M}}(i,j)\right|}_{=1} = 1. \tag{A.109}$$

Further, multiplying multiple left-stochastic matrices with themselves results in a left-stochastic matrix. Consequently each row the matrix product $\overline{\boldsymbol{M}}^t$ for some integer $t \geq 1$ sums to one and

$$\left|\left|\left(\overline{\boldsymbol{M}}^t\right)^\top\right|\right|_1 = \max_j \sum_i \left|\left(\overline{\boldsymbol{M}}^t\right)^\top (i,j)\right| = \max_i \underbrace{\sum_j \left|\left(\overline{\boldsymbol{M}}^t\right)(i,j)\right|}_{=1} = 1. \tag{A.110}$$

We assume that the matrices $\{\boldsymbol{F}_a\}_{a\in\mathcal{A}}$ are constructed such that Equation (3.21), the matching condition $\boldsymbol{F}_a = \boldsymbol{I} + \gamma \boldsymbol{M}_a \overline{\boldsymbol{F}}$, holds for every action a. Here, each entry $\boldsymbol{M}_a(i,j)$ describes the empirical probability of transitioning from partition i to partition j when action a is selected. Because the matrix $\overline{\boldsymbol{F}} = \frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}} \boldsymbol{F}_a$ and the matrix $\overline{\boldsymbol{M}} = \frac{1}{|\mathcal{A}|}\sum_{a\in\mathcal{A}} \boldsymbol{M}_a$, we have that

$$\overline{\boldsymbol{F}} = \boldsymbol{I} + \gamma \overline{\boldsymbol{M}} = \sum_{t=1}^{\infty} \gamma^{t-1} \overline{\boldsymbol{M}}^{t-1}. \tag{A.111}$$

Transposing and scaling this matrix by a factor $1-\gamma$ results in the identity

$$(1-\gamma)\overline{\boldsymbol{F}} = (1-\gamma)\sum_{t=1}^{\infty} \gamma^{t-1} \overline{\boldsymbol{M}}^{t-1}. \tag{A.112}$$

Each row of the matrix $(1-\gamma)\overline{\boldsymbol{F}}$ sums to one, because for an arbitrary row i,

$$\sum_j \left((1-\gamma)\overline{\boldsymbol{F}}\right)(i,j) = \sum_j \left((1-\gamma)\sum_{t=1}^{\infty} \gamma^{t-1}\overline{\boldsymbol{M}}^{t-1}\right)(i,j) \tag{A.113}$$

$$= \sum_j (1-\gamma)\sum_{t=1}^{\infty} \gamma^{t-1}\overline{\boldsymbol{M}}^{t-1}(i,j) \tag{A.114}$$

$$= (1-\gamma)\sum_{t=1}^{\infty} \gamma^{t-1} \underbrace{\sum_j \overline{\boldsymbol{M}}^{t-1}(i,j)}_{=1} \tag{A.115}$$

$$= (1-\gamma)\sum_{t=1}^{\infty} \gamma^{t-1} \tag{A.116}$$

$$= 1, \tag{A.117}$$

where line (A.114) follows by the distributive and commutative property of addition and multiplication between matrices and scalars. Line (A.116) follows by using the closed-form solution of a geometric series, and $\sum_{t=1}^{\infty} \gamma^{t-1} = \frac{1}{1-\gamma}$. Because each row of the matrix $(1-\gamma)\overline{\boldsymbol{F}}$ sums to one, we obtain Equation (A.107):

$$\left|\left|(1-\gamma)\overline{\boldsymbol{F}}^{\top}\right|\right|_1 = \max_j \sum_i \left|\left((1-\gamma)\overline{\boldsymbol{F}}^{\top}\right)(i,j)\right| = \max_i \underbrace{\sum_j \left|(1-\gamma)\overline{\boldsymbol{F}}(i,j)\right|}_{=1 \text{ by line (A.117)}} = 1. \tag{A.118}$$

For Equation (A.108), we consider an arbitrary row i and compute

$$\sum_j \left(\boldsymbol{I} - \gamma\overline{\boldsymbol{M}}\right)(i,j) = 1 - \gamma \underbrace{\sum_j \overline{\boldsymbol{M}}(i,j)}_{=1} = 1-\gamma, \tag{A.119}$$

where the first equality follows from $\boldsymbol{I}$ being the identity matrix and the second equality follows from each row of $\overline{\boldsymbol{M}}$ summing to one because $\overline{\boldsymbol{M}}$ is left-stochastic. Consequently,

$$\begin{aligned}
\left|\left|(\overline{\boldsymbol{F}}^{-1})^{\top}/(1-\gamma)\right|\right|_1 &= \left|\left|(\boldsymbol{I}-\gamma\overline{\boldsymbol{M}})^{\top}/(1-\gamma)\right|\right|_1 && \text{(A.120)}\\
&= \max_j \sum_i (\boldsymbol{I}-\gamma\overline{\boldsymbol{M}})^{\top}(i,j)/(1-\gamma) && \text{(A.121)}\\
&= \max_i \underbrace{\sum_j (\boldsymbol{I}-\gamma\overline{\boldsymbol{M}})(i,j)}_{=1-\gamma}/(1-\gamma) && \text{(A.122)}\\
&= \frac{1-\gamma}{1-\gamma} \qquad \text{(by Eq. (A.119))} && \text{(A.123)}\\
&= 1. && \text{(A.124)}
\end{aligned}$$

□

Definition 8 (LAM Feature Descriptor)**.** *For a cluster function c and a transition dataset $\mathcal{D}$, we define the* LAM feature descriptor *as*

$$\overline{\boldsymbol{\eta}}(s,a;c) = \frac{1}{|\mathcal{D}(s,a)|} \sum_{(s,a,r,s')\in\mathcal{D}(s,a)} \boldsymbol{e}_{c(s')}. \tag{A.125}$$

Definition 9. *For two cluster functions c and c^* such that $c \in \mathcal{C}(c^*)$, the indices of each partition*

in clustering c^* that are contained in partition i in clustering c is

$$\mathcal{I}(i) = \{j | \exists s. c(s) = i \wedge c^*(s) = j\}. \tag{A.126}$$

Definition 10. *For two cluster functions c and c^* such that $c \in \mathcal{C}(c^*)$, the projection matrix $\mathbf{\Phi}$ is defined as*

$$\mathbf{\Phi}(i,j) = \mathbf{1}_{[i \in \mathcal{I}(j)]}, \tag{A.127}$$

where the right side of Equation (A.127)*, is an indicator function that evaluates to one if $i \in \mathcal{I}(j)$. This matrix is of size $n \times m$, where n is the number of partitions induced by c^*, m is the number of partitions induced by c, and $n \geq m$ because $c \in \mathcal{C}(c^*)$.*

Lemma 8. *For two cluster functions c and c^* such that $c \in \mathcal{C}(c^*)$,*

$$\left|\left|\mathbf{\Phi}^\top\right|\right|_1 = 1 \tag{A.128}$$

Proof. Because $c \in \mathcal{C}(c^*)$, each row index i can only occur in one index set $\mathcal{I}(j)$. In other words, if $i \in \mathcal{I}(j)$ then $i \notin \mathcal{I}(\tilde{j})$ for $j \neq \tilde{j}$. Therefore, exactly one entry in each row of the matrix $\mathbf{\Phi}$ is set to one and each row sums to one:

$$\forall i, \ \sum_j \mathbf{\Phi}(i,j) = 1. \tag{A.129}$$

Consequently,

$$\left|\left|\mathbf{\Phi}^\top\right|\right|_1 = \max_j \sum_i \left|\left(\mathbf{\Phi}^\top\right)(i,j)\right| = \max_i \underbrace{\sum_j |\mathbf{\Phi}(i,j)|}_{=1} = 1. \tag{A.130}$$

□

A.3.2 Sub-Cluster Properties

The convergence proof of the clustering algorithm presented in Chapter 5 argues in part by induction on the set of all possible sub-cluster.

Definition 11 (Sub-Cluster)**.** *For a cluster function c^* the set of all* sub-cluster *is defined as*

$$\mathcal{C}(c^*) = \{c : \mathcal{S} \to \mathbb{N} | \forall s, \tilde{s}, \ c(s) \neq c(\tilde{s}) \implies c^*(s) \neq c^*(\tilde{s})\}. \tag{A.131}$$

An element $c \in \mathcal{C}(c^*)$ is a sub-clustering of c^*, because if c separates two different states, then they are also separated in c^*. This set of sub-cluster is closed under intersection.

Lemma 9 (Closed Under Intersection)**.** *If $c, \tilde{c} \in \mathcal{C}(c^*)$ for some cluster function c^*, then $c \cap \tilde{c} \in \mathcal{C}(c^*)$ where $c \cap \tilde{c}$ intersects the cluster function c and $\tilde{c}$ and for any two states s and $\tilde{s}$,*

$$(c \cap \tilde{c})(s) = (c \cap \tilde{c})(\tilde{s}) \iff c(s) = c(\tilde{s}) \text{ and } \tilde{c}(s) = \tilde{c}(\tilde{s}). \tag{A.132}$$

Proof. Consider an arbitrary cluster function c^*. Negating line (A.132) we have hat

$$(c \cap \tilde{c})(s) \neq (c \cap \tilde{c})(\tilde{s}) \iff c(s) \neq c(\tilde{s}) \text{ or } \tilde{c}(s) \neq \tilde{c}(\tilde{s}). \tag{A.133}$$

for two states s and $\tilde{s}$. Because $c, \tilde{c} \in \mathcal{C}(c^*)$,

$$(c(s) \neq c(\tilde{s}) \implies c^*(s) \neq c^*(\tilde{s})) \tag{A.134}$$

$$\text{and } (\tilde{c}(s) \neq \tilde{c}(\tilde{s}) \implies c^*(s) \neq c^*(\tilde{s})) \tag{A.135}$$

$$\implies ((c \cap \tilde{c})(s) \neq (c \cap \tilde{c})(\tilde{s}) \implies c^*(s) \neq c^*(\tilde{s})) \tag{A.136}$$

$$\implies c \cap \tilde{c} \in \mathcal{C}(c^*). \tag{A.137}$$

□

Before proving a property relating the feature descriptor vectors $\overline{\boldsymbol{\xi}}$ between different sub-clusterings, we first focus on the LAM feature descriptor vector $\overline{\boldsymbol{\eta}}$.

Lemma 10. *For a cluster function c^*, and two states s and $\tilde{s}$ occuring in a transition dataset,*

$$\forall c \in \mathcal{C}(c^*),\ \left|\left|\overline{\boldsymbol{\eta}}(s, a; c) - \overline{\boldsymbol{\eta}}(\tilde{s}, a; c)\right|\right|_1 \leq \left|\left|\overline{\boldsymbol{\eta}}(s, a; c^*) - \overline{\boldsymbol{\eta}}(\tilde{s}, a; c^*)\right|\right|_1 \tag{A.138}$$

Proof. We first observe that in each row of the matrix $\boldsymbol{\Phi}$ exactly one entry is set to one (as stated in the proof of Lemma 8), and for any state s we have by construction of $\boldsymbol{\Phi}$ that

$$\boldsymbol{e}_{c(s)}^\top = \boldsymbol{e}_{c^*(s)}^\top \boldsymbol{\Phi}. \tag{A.139}$$

Using the projection matrix $\boldsymbol{\Phi}$ we can express $\overline{\boldsymbol{\eta}}(s,a;c)$ in terms of $\overline{\boldsymbol{\eta}}(s,a;c^*)$:

$$\begin{aligned}
(\overline{\boldsymbol{\eta}}(s,a;c))^\top &= \frac{1}{|\mathcal{D}(s,a)|} \sum_{(s,a,r,s')\in\mathcal{D}(s,a)} \boldsymbol{e}_{c(s')}^\top && \text{(by Definition 8)} && \text{(A.140)}\\
&= \frac{1}{|\mathcal{D}(s,a)|} \sum_{(s,a,r,s')\in\mathcal{D}(s,a)} \boldsymbol{e}_{c^*(s)}^\top \boldsymbol{\Phi} && \text{(by Equation (A.139))} && \text{(A.141)}\\
&= (\overline{\boldsymbol{\eta}}(s,a;c^*))^\top \boldsymbol{\Phi} && \text{(by Definition 8)} && \text{(A.142)}\\
\iff \overline{\boldsymbol{\eta}}(s,a;c) &= \boldsymbol{\Phi}^\top \overline{\boldsymbol{\eta}}(s,a;c^*) && && \text{(A.143)}
\end{aligned}$$

Using identity (A.143), we can obtain the bound (A.138):

$$\begin{aligned}
\left|\left|\overline{\boldsymbol{\eta}}(s,a;c) - \overline{\boldsymbol{\eta}}(\tilde{s},a;c)\right|\right|_1 &= \left|\left|\boldsymbol{\Phi}^\top \overline{\boldsymbol{\eta}}(s,a;c^*) - \boldsymbol{\Phi}^\top \overline{\boldsymbol{\eta}}(\tilde{s},a;c^*)\right|\right|_1 && \text{(by Eq. (A.143))} && \text{(A.144)}\\
&= \left|\left|\boldsymbol{\Phi}^\top \left(\overline{\boldsymbol{\eta}}(s,a;c^*) - \overline{\boldsymbol{\eta}}(\tilde{s},a;c^*)\right)\right|\right|_1 && && \text{(A.145)}\\
&\leq \underbrace{\left|\left|\boldsymbol{\Phi}^\top\right|\right|_1}_{=1} \left|\left|\overline{\boldsymbol{\eta}}(s,a;c^*) - \overline{\boldsymbol{\eta}}(\tilde{s},a;c^*)\right|\right|_1 && \text{(by Lemma (8))} && \text{(A.146)}\\
&= \left|\left|\overline{\boldsymbol{\eta}}(s,a;c^*) - \overline{\boldsymbol{\eta}}(\tilde{s},a;c^*)\right|\right|_1 && && \text{(A.147)}
\end{aligned}$$

□

Now we prove a property relating the feature descriptor vectors $\overline{\boldsymbol{\xi}}$ between different sub-clusterings.

Lemma 11. *For a cluster function c^*, and two states s and $\tilde{s}$ occurring in a transition data set,*

$$\forall c \in \mathcal{C}(c^*),\ \left|\left|\overline{\boldsymbol{\xi}}(s,c) - \overline{\boldsymbol{\xi}}(\tilde{s},c)\right|\right|_1 \leq \left|\left|\overline{\boldsymbol{\xi}}(s,c^*) - \overline{\boldsymbol{\xi}}(\tilde{s},c^*)\right|\right|_1 \tag{A.148}$$

Proof. Because $\overline{\boldsymbol{\xi}}$ concatenates the SF vectors $\overline{\boldsymbol{\psi}}$ for all actions, we first focus on these SF vectors. We can express $\overline{\boldsymbol{\psi}}(s,a;\boldsymbol{\phi})$ in terms of $\overline{\boldsymbol{\eta}}(s,a;c)$ for a representation function $\boldsymbol{\phi} : s \mapsto \boldsymbol{e}_{c(s)}$. Using Equations (5.7) and (A.125) we can express $\overline{\boldsymbol{\psi}}(s,a;\boldsymbol{\phi})$ in terms of $\overline{\boldsymbol{\eta}}(s,a;c)$ for a representation function $\boldsymbol{\phi} : s \mapsto \boldsymbol{e}_{c(s)}$:

$$\begin{aligned}
(\overline{\psi}(s,a;\boldsymbol{\phi}))^\top &= \frac{1}{|\mathcal{D}(s,a)|} \sum_{(s,a,r,s')\in\mathcal{D}(s,a)} (1-\gamma)\boldsymbol{e}_{c(s')}^\top \overline{\boldsymbol{F}} && \text{(by Eq. (5.7))} && \text{(A.149)}\\
&= (\overline{\boldsymbol{\phi}}(s,a;c))^\top (1-\gamma)\overline{\boldsymbol{F}}. && \text{(by Eq. (A.125))} && \text{(A.150)}
\end{aligned}$$

For the cluster function c^* and the corresponding representation function $\boldsymbol{\phi}^* : s \mapsto \boldsymbol{e}_{c^*(s)}$ we denote the SF matrix with $\overline{\boldsymbol{F}}^*$ and

$$(\overline{\psi}(s,a;\boldsymbol{\phi}^*))^\top = (\overline{\boldsymbol{\phi}}(s,a;c^*))^\top (1-\gamma)\overline{\boldsymbol{F}}^*. \tag{A.151}$$

By invertibility of $\overline{\boldsymbol{F}}^*$ (Lemma 4), we can transform Equation (A.151):

$$\overline{\psi}(s,a;\boldsymbol{\phi}^*) = (1-\gamma)(\overline{\boldsymbol{F}}^*)^\top \overline{\boldsymbol{\eta}}(s,a;c^*) \tag{A.152}$$

$$\iff ((\overline{\boldsymbol{F}}^*)^{-1})^\top \overline{\psi}(s,a;\boldsymbol{\phi}^*)/(1-\gamma) = \overline{\boldsymbol{\eta}}(s,a;c^*) \tag{A.153}$$

Further, by Equation (A.143)

$$\overline{\boldsymbol{\phi}}(s,a;c) = \boldsymbol{\Phi}^\top \overline{\boldsymbol{\phi}}(s,a;c^*) \tag{A.154}$$

and we can express the SF vector $\overline{\psi}(s,a;\boldsymbol{\phi})$ in terms of the vector $\overline{\psi}(s,a;\boldsymbol{\phi}^*)$:

$$\begin{aligned}
\overline{\boldsymbol{\psi}}(s,a;\boldsymbol{\phi}) &= \overline{\boldsymbol{F}}^\top \overline{\boldsymbol{\eta}}(s,a;c) && && \text{(A.155)}\\
&= (1-\gamma)\overline{\boldsymbol{F}}^\top \boldsymbol{\Phi}^\top \overline{\boldsymbol{\eta}}(s,a;c^*) && \text{(by Eq. (A.143))} && \text{(A.156)}\\
&= (1-\gamma)\overline{\boldsymbol{F}}^\top \boldsymbol{\Phi}^\top ((\overline{\boldsymbol{F}}^*)^{-1})^\top \overline{\boldsymbol{\psi}}(s,a;\boldsymbol{\phi}^*)/(1-\gamma) && \text{(by Eq. (A.153))} && \text{(A.157)}\\
&= \overline{\boldsymbol{F}}^\top \boldsymbol{\Phi}^\top ((\overline{\boldsymbol{F}}^*)^{-1})^\top \overline{\boldsymbol{\psi}}(s,a;\boldsymbol{\phi}^*) && && \text{(A.158)}
\end{aligned}$$

Using Equation (A.158), we can bound the difference in SF vectors for two arbitrary states s and $\tilde{s}$:

$$\begin{aligned}
\left|\left|\overline{\boldsymbol{\psi}}(s,a;\boldsymbol{\phi}) - \overline{\boldsymbol{\psi}}(\tilde{s},a;\boldsymbol{\phi})\right|\right|_1 &= \left|\left|\overline{\boldsymbol{F}}^\top \boldsymbol{\Phi}^\top ((\overline{\boldsymbol{F}}^*)^{-1})^\top \overline{\boldsymbol{\psi}}(s,a;\boldsymbol{\phi}^*) - \overline{\boldsymbol{F}}^\top \boldsymbol{\Phi}^\top ((\overline{\boldsymbol{F}}^*)^{-1})^\top \overline{\boldsymbol{\psi}}(\tilde{s},a;\boldsymbol{\phi}^*)\right|\right|_1 && \text{(A.159)}\\
&\leq \left|\left|\overline{\boldsymbol{F}}^\top \boldsymbol{\Phi}^\top ((\overline{\boldsymbol{F}}^*)^{-1})^\top\right|\right|_1 \left|\left|\overline{\boldsymbol{\psi}}(s,a;\boldsymbol{\phi}^*) - \overline{\boldsymbol{\psi}}(\tilde{s},a;\boldsymbol{\phi}^*)\right|\right|_1 && \text{(A.160)}\\
&\leq \underbrace{\left|\left|\overline{\boldsymbol{F}}^\top\right|\right|_1}_{=1} \underbrace{\left|\left|\boldsymbol{\Phi}^\top\right|\right|_1}_{=1} \underbrace{\left|\left|((\overline{\boldsymbol{F}}^*)^{-1})^\top\right|\right|_1}_{=1} \left|\left|\overline{\boldsymbol{\psi}}(s,a;\boldsymbol{\phi}^*) - \overline{\boldsymbol{\psi}}(\tilde{s},a;\boldsymbol{\phi}^*)\right|\right|_1 && \text{(A.161)}\\
&= \left|\left|\overline{\boldsymbol{\psi}}(s,a;\boldsymbol{\phi}^*) - \overline{\boldsymbol{\psi}}(\tilde{s},a;\boldsymbol{\phi}^*)\right|\right|_1. && \text{(A.162)}
\end{aligned}$$

Line (A.162) follows by lemmas 8 and 7 and the previous two lines by properties of the matrix

p-norm $||\cdot||_1$ (where $p = 1$). Because the vector $\overline{\boldsymbol{\xi}}$ concatenates the vectors $\overline{\boldsymbol{\psi}}$ for each action,

$$\forall c \in \mathcal{C}(c^*),\ \left|\left|\overline{\boldsymbol{\xi}}(s,c) - \overline{\boldsymbol{\xi}}(\tilde{s},c)\right|\right|_1 = \sum_{a\in\mathcal{A}} \left|\left|\overline{\boldsymbol{\psi}}(s,a;\boldsymbol{\phi}) - \overline{\boldsymbol{\psi}}(\tilde{s},a;\boldsymbol{\phi})\right|\right|_1 \tag{A.163}$$

$$\leq \sum_{a\in\mathcal{A}} \left|\left|\overline{\boldsymbol{\psi}}(s,a;\boldsymbol{\phi}^*) - \overline{\boldsymbol{\psi}}(\tilde{s},a;\boldsymbol{\phi}^*)\right|\right|_1 \quad \text{(by Eq. (A.162))} \tag{A.164}$$

$$= \left|\left|\overline{\boldsymbol{\xi}}(s,c^*) - \overline{\boldsymbol{\xi}}(\tilde{s},c^*)\right|\right|_1. \tag{A.165}$$

□

Now we prove a property about the separability of the vectors $\overline{\boldsymbol{r}}$ and $\overline{\boldsymbol{\xi}}$ that are constructed for sub-clustering. Because the following lemma is needed for the convergence proof of the clustering algorithm, the separability assumption (Assumption 4) is necessary.

Lemma 12 (Vector Separability of Sub-Cluster)**.** *Assume a maximally compressed reward-predictive cluster function c^* and values $\hat{\varepsilon}_r, \hat{\varepsilon}_\psi > 0$ that satisfy Assumption 4. Then, for any $c, \tilde{c} \in \mathcal{C}(c^*)$ such that $c(s) \neq c(\tilde{s}) \implies \tilde{c}(s) \neq \tilde{c}(\tilde{s})$,*

$$c(s) \neq c(\tilde{s}) \implies ||\hat{\boldsymbol{r}}(s) - \hat{\boldsymbol{r}}(\tilde{s})||_1 > \hat{\varepsilon}_r \text{ or } ||\hat{\boldsymbol{\xi}}(s,\tilde{c}) - \hat{\boldsymbol{\xi}}(\tilde{s},\tilde{c})||_1 > \hat{\varepsilon}_\psi. \tag{A.166}$$

Proof of Lemma 12. By Assumption 4,

$$c^*(s) \neq c^*(\tilde{s}) \implies ||\hat{\boldsymbol{r}}(s) - \hat{\boldsymbol{r}}(\tilde{s})||_1 > \hat{\varepsilon}_r \text{ or } ||\hat{\boldsymbol{\xi}}(s,c^*) - \hat{\boldsymbol{\xi}}(\tilde{s},c^*)||_1 > \hat{\varepsilon}_\psi. \tag{A.167}$$

By inversion of the implication, we have that

$$||\hat{\boldsymbol{r}}(s) - \hat{\boldsymbol{r}}(\tilde{s})||_1 \leq \hat{\varepsilon}_r \text{ and } ||\hat{\boldsymbol{\xi}}(s,c^*) - \hat{\boldsymbol{\xi}}(\tilde{s},c^*)||_1 \leq \hat{\varepsilon}_\psi \implies c^*(s) = c^*(\tilde{s}) \tag{A.168}$$

By Lemma 11, $||\hat{\boldsymbol{\xi}}(s,\tilde{c}) - \hat{\boldsymbol{\xi}}(\tilde{s},\tilde{c})||_1 \leq ||\hat{\boldsymbol{\xi}}(s,c^*) - \hat{\boldsymbol{\xi}}(\tilde{s},c^*)||_1$ for $\tilde{c} \in \mathcal{C}(c^*)$. (Here, we pick $\tilde{c}$ such that it is also a sub-clustering of c and $c(s) \neq c(\tilde{s}) \implies \tilde{c}(s) \neq \tilde{c}(\tilde{s})$.) Consequently,

$$||\hat{\boldsymbol{r}}(s) - \hat{\boldsymbol{r}}(\tilde{s})||_1 \leq \hat{\varepsilon}_r \text{ and } ||\hat{\boldsymbol{\xi}}(s,\tilde{c}) - \hat{\boldsymbol{\xi}}(\tilde{s},\tilde{c})||_1 \leq \hat{\varepsilon}_\psi \implies c^*(s) = c^*(\tilde{s}) \tag{A.169}$$

By Definition 11, if $c \in \mathcal{C}(c^*)$, then

$$c(s) \neq c(\tilde{s}) \implies c^*(s) \neq c^*(\tilde{s}) \tag{A.170}$$

and by inversion of the implication

$$c^*(s) = c^*(\tilde{s}) \implies c(s) = c(\tilde{s}). \tag{A.171}$$

Combining Lines (A.169) and (A.171) results in

$$||\hat{\boldsymbol{r}}(s) - \hat{\boldsymbol{r}}(\tilde{s})||_1 \leq \hat{\varepsilon}_r \text{ and } ||\hat{\boldsymbol{\xi}}(s, \tilde{c}) - \hat{\boldsymbol{\xi}}(\tilde{s}, \tilde{c})||_1 \leq \hat{\varepsilon}_\psi \implies c(s) = c(\tilde{s}) \tag{A.172}$$

and by inversion

$$c(s) \neq c(\tilde{s}) \implies ||\hat{\boldsymbol{r}}(s) - \hat{\boldsymbol{r}}(\tilde{s})||_1 > \hat{\varepsilon}_r \text{ or } ||\hat{\boldsymbol{\xi}}(s, \tilde{c}) - \hat{\boldsymbol{\xi}}(\tilde{s}, \tilde{c})||_1 > \hat{\varepsilon}_\psi. \tag{A.173}$$

□

We now make an additional assumption about the cluster algorithm:

Assumption 5 (Compactness)**.** *At every cluster operation in Algorithm 3 a maximally compressed clustering is obtained.*

Assumption 5 makes explicit that the clustering operation must not construct clusters when not necessary. However, this assumption is not restrictive in the presented context, because if the function approximation routine is ε-perfect (Assumption 3) and the separability assumption (Assumption 4) is satisfied, then computing a compact clustering is trivial (one can greedily merge vectors into the same cluster that fall in their distance below the used threshold).

A.3.3 Convergence Proof

First, we provide a formal definition of a maximally-compressed reward-predictive representation.

Definition 12 (Maximally-Compressed Reward Predictive)**.** *A cluster function c^* is* maximally-compressed reward predictive *if the representation function $\boldsymbol{\phi} : s \mapsto \boldsymbol{e}_{c^*(s)}$ satisfies the empirical LSFM conditions* (5.8) *for $\varepsilon_r = \varepsilon_\psi = 0$ while minimizing the number of constructed latent states.*

Now we present the lemmas needed for the convergence proof.

Lemma 13 (Termination Condition)**.** *If the refinement loop in Algorithm 3 terminates with a cluster function c_T, then c_T encodes an empirical approximate reward-predictive representation.*

Proof. The proof follows from the arguments presented in Section 5.1.1. If c_T is returned by Algorithm 3, then the vectors $\boldsymbol{w}_a$ can be constructed using Equation (5.14). Then Equation (5.15) follows and the reward condition in Line (5.8) is satisfied. For the SF fixed point, we use Equation (5.21) to construct the matrices $\boldsymbol{F}_a$. By the argumentation presented in Section 5.1.1, Equation (5.26) follows and the SF fixed-point condition in Line (5.8) is satisfied. Therefore the representation function $\phi : s \mapsto \boldsymbol{e}_{c_T(s)}$ is an empirical approximate reward-predictive representation. □

Lemma 14 (Refinement)**.** *Consider the cluster function sequence $c_0, c_1, ..., c_t$ generated by Algorithm 3. Then,*

$$\forall 0 \leq i < j,\ c_i(s) \neq c_i(\tilde{s}) \implies c_j(s) \neq c_j(\tilde{s}). \tag{A.174}$$

Proof. The proof is by construction of c_t for $t \geq 1$ and follows from line 10 of Algorithm 3. □

We refer to a cluster function c^* that satisfies the empirical LSFM conditions (5.8) and minimizes the number created latent states as *maximally compressed*.

The following lemma states the main result needed for proving convergence to a maximally compressed representation.

Lemma 15. *Under Assumption 4 and 5, Algorithm 3 constructs a sequence of cluster functions $c_1, ..., c_t \in \mathcal{C}(c^*)$ for some maximally-compressed reward-predictive cluster function c^*.*

Proof. The proof is by induction on the cluster function sequence.
Base Case: Algorithm 3 constructs the reward cluster function c_1 such that

$$c(s) = c(\tilde{s}) \implies ||\hat{\boldsymbol{r}}(s) - \hat{\boldsymbol{r}}(\tilde{s})||_1 \leq \hat{\varepsilon}_r \tag{A.175}$$

and by Assumption 5, the number of cluster's is minimized. By inversion of Line A.176,

$$||\hat{\boldsymbol{r}}(s) - \hat{\boldsymbol{r}}(\tilde{s})||_1 > \hat{\varepsilon}_r \implies c(s) \neq c(\tilde{s}). \tag{A.176}$$

By separability (Assumption 4),

$$c^*(s) \neq c^*(\tilde{s}) \implies ||\hat{\boldsymbol{r}}(s) - \hat{\boldsymbol{r}}(\tilde{s})||_1 > \hat{\varepsilon}_r \tag{A.177}$$

$$\implies c(s) \neq c(\tilde{s}) \qquad \text{(by Line (A.176))} \tag{A.178}$$

$$\implies c_1 \in \mathcal{C}(c^*). \qquad \text{(by definition of } \mathcal{C}(c^*)) \tag{A.179}$$

Induction Hypothesis (I.H.): For $T \in \mathbb{N}$ and a maximally-compressed reward-predictive cluster function c^*, we assume that for all $t \in (1, T]$, $c_{t-1} \in \mathcal{C}(c^*)$.

Induction Step: For the induction step, we assume a smaller clustering threshold ε such that

$$c_T(s) \neq c_T(s) \implies ||\hat{\boldsymbol{r}}(s) - \hat{\boldsymbol{r}}(\tilde{s})||_1 > \hat{\varepsilon}_r \text{ or } ||\hat{\boldsymbol{\xi}}(s, c_{T-1}) - \hat{\boldsymbol{\xi}}(\tilde{s}, c_{T-1})||_1 > \varepsilon. \tag{A.180}$$

Below we will relax this assumption. If $||\hat{\boldsymbol{r}}(s) - \hat{\boldsymbol{r}}(\tilde{s})||_1 > \hat{\varepsilon}_r$, then $c^*(s) \neq c^*(\tilde{s})$ as well because c^* is maximally-compressed reward-predictive (by the argument in the base case). Suppose $c_T(s) \neq c_T(s) \implies ||\hat{\boldsymbol{\xi}}(s, c_{T-1}) - \hat{\boldsymbol{\xi}}(\tilde{s}, c_{T-1})||_1 > \varepsilon$. Because we assume a smaller cluster threshold ε,

$$c_T(s) = c_T(s) \implies ||\hat{\boldsymbol{\xi}}(s, c_{T-1}) - \hat{\boldsymbol{\xi}}(\tilde{s}, c_{T-1})||_1 \leq \varepsilon. \tag{A.181}$$

Combining Line (A.181) with the implication $c_T(s) \neq c_T(s) \implies ||\hat{\boldsymbol{\xi}}(s, c_{T-1}) - \hat{\boldsymbol{\xi}}(\tilde{s}, c_{T-1})||_1 > \varepsilon$ results in

$$c_T(s) = c_T(s) \iff ||\hat{\boldsymbol{\xi}}(s, c_{T-1}) - \hat{\boldsymbol{\xi}}(\tilde{s}, c_{T-1})||_1 \leq \varepsilon. \tag{A.182}$$

Because approximations of vectors $\overline{\boldsymbol{\xi}}$ are ε-perfect and because a smaller cluster threshold is used in Algorithm 3 to account for approximation errors, we have that

$$||\hat{\boldsymbol{\xi}}(s, c_{T-1}) - \hat{\boldsymbol{\xi}}(\tilde{s}, c_{T-1})||_1 \leq \varepsilon \iff \overline{\boldsymbol{\xi}}(s, c_{T-1}) = \overline{\boldsymbol{\xi}}(\tilde{s}, c_{T-1}). \tag{A.183}$$

Combining Lines (A.183) and (A.182) leads to

$$c_T(s) = c_T(s) \iff \overline{\boldsymbol{\xi}}(s, c_{T-1}) = \overline{\boldsymbol{\xi}}(\tilde{s}, c_{T-1}). \tag{A.184}$$

Consequently,

$$\hat{c}_T(s) \neq \hat{c}_T(\tilde{s}) \tag{A.185}$$

$$\implies \overline{\boldsymbol{\xi}}(s, c_{T-1}) \neq \overline{\boldsymbol{\xi}}(\tilde{s}, c_{T-1}) \qquad \text{(by Line (A.184))} \tag{A.186}$$

$$\implies \overline{\boldsymbol{\xi}}(s, c^*) \neq \overline{\boldsymbol{\xi}}(\tilde{s}, c^*) \qquad \text{(by applying Lemma 11 to the difference on both sides)} \tag{A.187}$$

$$\implies c^*(s) \neq c^*(\tilde{s}) \qquad \text{(by } c^* \text{ satisfying Definition 12)} \tag{A.188}$$

$$\implies \hat{c}_T \in \mathcal{C}(c^*). \qquad \text{(by Definition 11)} \tag{A.189}$$

$$\implies c_T = \hat{c}_T \cap c_{T-1} \in \mathcal{C}(c^*). \qquad \text{(by Lemma 9 and I.H.)} \tag{A.190}$$

This proves that $c_T \in \mathcal{C}(c^*)$ where c^* is maximally-compressed reward predictive.

To complete the proof, we show that $\hat{\varepsilon}_\psi$ can be used to cluster the vectors $\hat{\boldsymbol{\xi}}$ at every iteration. Here, the separability assumption (Assumption 4) allows us to apply Lemma 12 and obtain that for every $t \in (1, T]$,

$$c_t(s) \neq c_t(s) \implies ||\hat{\boldsymbol{r}}(s) - \hat{\boldsymbol{r}}(\tilde{s})||_1 > \hat{\varepsilon}_r \text{ or } ||\hat{\boldsymbol{\xi}}(s, c_{t-1}) - \hat{\boldsymbol{\xi}}(\tilde{s}, c_{t-1})||_1 > \hat{\varepsilon}_\psi. \tag{A.191}$$

Consequently, we can set $\hat{\varepsilon}_\psi = \varepsilon$ in the induction argument above.

□

Proof of Theorem 5. By Lemmas 13 and 14, Algorithm 3 converges to a reward-predictive state representation within finitely many steps. At the very least, the algorithm terminates at a state representation that associates a separate latent state with each state observation.

Under Assumptions 4 and 5, the algorithm terminates on a cluster function $c_t \in \mathcal{C}(c^*)$ (by Lemma 15). Because c^* is assumed to be maximally-compressed reward-predictive representation, then the sub-clustering $c_t \in \mathcal{C}(c^*)$ is also maximally compressed (any sub-clustering of a maximally-compressed reward-predictive representation c^* is either not reward-predictive, or it is also a maximally-compressed reward-predictive representation). Consequently, the last cluster function constructed by Algorithm 3 maximally-compressed reward predictive representation. □

Appendix B

Simulation Implementation and Hyper-Parameters

B.1 Learning Rates Used In Chapter 2 Simulations

Table B.1: Learning rates used for Q-learning in Figures 2.2 and 2.3.

	Tested α	Slight Rew. Change Best α	Sig. Rew. Change Best α
Q-learning with ε-Greedy	0.1, 0.3, 0.5, 0.7, 0.9	0.7	0.1
Q-learning with Optimistic	0.1, 0.3, 0.5, 0.7, 0.9	0.9	0.9

Table B.2: Learning rates used for SF-learning in Figures 2.2 and 2.3.

			Slight Rew. Change		Sig. Rew. Change	
	Tested α_{SF}	Tested α_r	Best α_{SF}	Best α_r	Best α_{SF}	Best α_r
SF-learning with ε-Greedy	0.1, 0.3, 0.5, 0.7, 0.9	0.1, 0.3, 0.5, 0.7, 0.9	0.7	0.5	0.1	0.1
SF-learning with Optimistic	0.1, 0.3, 0.5, 0.7, 0.9	0.1, 0.3, 0.5, 0.7, 0.9	0.5	0.9	0.3	0.9

B.2 Implementation Details of Chapter 3 Simulations

The presented experiments are conducted on finite MDPs and use a state representation function

$$\phi : s \mapsto \mathbf{\Phi}(s, :), \tag{B.1}$$

where $\boldsymbol{\Phi}$ is a $\mathcal{S} \times n$ matrix and $\boldsymbol{\Phi}(s,:)$ is a row with state index s. The feature dimension n is a fixed hyper parameter for each experiment.

B.2.1 Matrix Optimization in Column World

The column world experiment (Figure 3.5) learns a state representation using the full transition and reward tables. Assume that the transition table of the column world task is stored as a set of stochastic transition matrices $\{\boldsymbol{P}_a\}_{a\in\mathcal{A}}$ and the reward table as a set of reward vectors $\{\boldsymbol{r}_a\}_{a\in\mathcal{A}}$. The one-step reward prediction errors and linear SF prediction errors are minimized for the LSFM $\{\boldsymbol{F}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$ using the loss objective

$$\mathcal{L}_{\text{LSFM-mat}} = \sum_{a\in\mathcal{A}} ||\boldsymbol{\Phi}\boldsymbol{w}_a - \boldsymbol{r}_a||_2^2 + \alpha_\psi ||\boldsymbol{\Phi} + \gamma \boldsymbol{P}_a \boldsymbol{\Phi}\overline{\boldsymbol{F}} - \boldsymbol{\Phi}\boldsymbol{F}_a||_2^2. \tag{B.2}$$

For $\alpha_\psi = 1$, the loss objective $\mathcal{L}_{\text{LSFM-mat}}$ is optimized with respect to all free parameters $\{\boldsymbol{F}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$ and $\boldsymbol{\Phi}$. Similarly, a LAM $\{\boldsymbol{M}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$ is computed using the loss objective

$$\mathcal{L}_{\text{LAM-mat}} = \sum_{a\in\mathcal{A}} ||\boldsymbol{\Phi}\boldsymbol{w}_a - \boldsymbol{r}_a||_2^2 + ||\boldsymbol{\Phi}\boldsymbol{M}_a - \boldsymbol{P}_a\boldsymbol{\Phi}||_2^2. \tag{B.3}$$

This loss objective is optimized with respect to all free parameters $\{\boldsymbol{M}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$ and $\boldsymbol{\Phi}$. Both experiments used the Adam optimizer [56] with a learning rate of 0.1 and Tensorflow [1] default parameters. Optimization was initialized by sampling entries for $\boldsymbol{\Phi}$ uniformly from the interval $[0, 1]$. The LAM $\{\boldsymbol{M}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$ or LSFM $\{\boldsymbol{F}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$ was initialized using a least squares solution for the initialization of $\boldsymbol{\Phi}$.

B.2.2 Puddle-World Experiment

In the puddle world MDP transitions are probabilistic, because with a 5% chance, the agent does not move after selecting any action. The partition maps presented in Figures 3.6(b) and 3.6(c) were obtained by clustering latent state vectors using agglomerative clustering. A finite data set of transitions $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}_{i=1}^D$ was collected by selecting actions uniformly as random. Given

Hyper-Parameter	LAM	LSFM	Tested Values
Learning Rate	0.0005	0.0005	0.0001, 0.0005, 0.001, 0.005
α_ψ	-	0.01	0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0
α_p	1.0	-	0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0
α_N	0.1	0.0	0.0, 0.0001, 0.001, 0.01, 0.1
Feature Dimension	80	80	
Batch Size	50	50	
Number of Training Transitions	10000	10000	
Number of Gradient Steps	50000	50000	

Table B.3: Hyper-Parameter for Puddle-World Experiment

such a data set $\mathcal{D}$, the loss objective

$$\mathcal{L}_{\text{LAM}} = \underbrace{\sum_{i=1}^{D} \left(\boldsymbol{\phi}_{s_i}^\top \boldsymbol{w}_{a_i} - r_i\right)^2}_{=\mathcal{L}_r} + \alpha_p \underbrace{\sum_{i=1}^{D} \left|\left|\boldsymbol{\phi}_{s_i}^\top \boldsymbol{M}_a - \boldsymbol{\phi}_{s_i'}^\top\right|\right|_2^2}_{=\mathcal{L}_p} + \alpha_N \underbrace{\sum_{i=1}^{D} \left(\left|\left|\boldsymbol{\phi}_{s_i}\right|\right|_2^2 - 1\right)^2}_{=\mathcal{L}_N},$$

is used to approximate a reward-predictive state representation using a LAM. Optimization was initialized by each entry of the matrix $\boldsymbol{\Phi}$ uniformly at random and then finding a LAM $\{\boldsymbol{M}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$ for this initialized representation using least squares regression.

For the LSFM experiment, the matrix $\boldsymbol{\Phi}$ was also initialized using values sampled uniformly at random. The LSFM $\{\boldsymbol{F}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$ was set to zero matrices and vectors at initialization. Both loss objective functions were optimized using the Adam optimizer with Tensorflow's default parameters. Table B.3 lists the hyper-parameter that were found to work best for each model. Figures 3.6(h) and 3.6(i) are plotted by first evaluating an ε-greedy policy using the full transition and reward tables of the task. Then the state representation is used to find an approximation of the value functions for each ε setting using least-squares linear regression. Each curve then plots the maximum value prediction error.

B.2.3 Transfer Experiments

For the transfer experiment presented in Section 3.5, a training data set of 10000 transitions was collected from Task B. The LSFM was trained using the hyper-parameter listed in Table B.4.

Value-predictive state representations are learned using a modified version of Neural Fitted Q-iteration [87]. The Q-value function is computed with

$$Q(s, a; \boldsymbol{\Phi}, \{\boldsymbol{q}_a\}_{a\in\mathcal{A}}) = \boldsymbol{\phi}_s^\top \boldsymbol{q}_a, \tag{B.4}$$

Hyper-Parameter	LSFM	Tested Values
Learning Rate	0.001	0.0001, 0.0005, 0.001, 0.005
α_ψ	0.0001	0.001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0
α_N	0.0	0.0, 0.0001, 0.001, 0.01, 0.1
Feature Dimension	50	
Batch Size	50	
Number of Training Transitions	10000	
Number of Gradient Steps	50000	

Table B.4: Hyper-Parameter for LSFM on Task A

where the state features $\boldsymbol{\phi}_s$ are computed as shown in Equation (B.1). To find a value-predictive state representation, the loss function

$$\mathcal{L}_q(\boldsymbol{\Phi}, \{\boldsymbol{q}_a\}_{a\in\mathcal{A}}) = \sum_{i=1}^{D} (Q(s_i, a_i; \boldsymbol{\Phi}, \{\boldsymbol{q}_a\}_{a\in\mathcal{A}}) - y_{s_i,a_i,r_i,s_i'})^2 \tag{B.5}$$

is minimized using stochastic gradient descent on a transition data set $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}_{i=1}^{D}$. When differentiating the loss objective $\mathcal{L}_q$ with respect to its free parameters $\boldsymbol{\Phi}, \{\boldsymbol{q}_a\}_{a\in\mathcal{A}}$, the prediction target

$$y_{s,a,r,s'} = r + \gamma \max_{a'} Q(s', a'; \boldsymbol{\Phi}, \{\boldsymbol{q}_a\}_{a\in\mathcal{A}}) \tag{B.6}$$

is considered a constant and no gradient of $y_{s,a,r,s'}$ is computed. A value-predictive state representation is learned for Task A by optimizing over all free parameters $\boldsymbol{\Phi}, \{\boldsymbol{q}_a\}_{a\in\mathcal{A}}$. Table B.5 lists the used hyper-parameter. For Task A the best hyper-parameter setting was obtained by testing the learned state representation on Task A and using the model that produced the shortest episode length, averaged over 20 repeats.

On Task B, a previously learned state representation is evaluated using the same implementation of Fitted Q-iteration, but the previously learned state representation is re-used and considered a constant. At transfer, gradients of $\mathcal{L}_q$ are only computed with respect to the vector set $\{\boldsymbol{q}_a\}_{a\in\mathcal{A}}$ and the feature matrix $\boldsymbol{\Phi}$ is held constant.

The tabular model first compute a partial transition and reward table of Task B by averaging different transitions and reward using the given data set. If no transition is provided for a particular state and action pair, uniform transitions are assumed. If no reward is provided for a particular state and action pair, a reward value is sampled uniformly at random from the interval $[0, 1]$. Augmenting a partial transition and reward table is equivalent to providing the agent with a uniform prior over rewards and transitions. The tabular model's optimal policy is computed using value iteration.

Hyper-Parameter	Fitted Q-iteration, learning on Task A	Fitted Q-iteration, evaluation on Task B	Tested Values
Learning Rate	0.001	0.00001	0.00001, 0.0001, 0.001, 0.01
Feature Dimension	50	50	
Batch Size	50	50	
Training Transitions	10000	Varies	
Gradient Steps	50000	20000	

Table B.5: Hyper-Parameter used for Fitted Q-iteration

To plot the right panel in Figure 3.7(c), twenty different transition data sets of a certain fixed size were collected and the Fitted Q-iteration algorithm was used to approximate the optimal value function. For both tested state representations and data sets, a small enough learning rate was found to guarantee that Fitted Q-iteration converges. The found solutions were evaluated twenty times, and if all evaluation completed the navigation task within 22 time steps (which is close to optimal), then this data set is considered to be optimally solved. Note that all tested evaluation runs either complete within 22 time steps or hit the timeout threshold of 5000 time steps. Table B.5 lists the hyper-parameters used for Fitted Q-iteration to obtain the right panel in Figure 3.7(c). For transfer evaluation, the hyper-parameter setting was used that approximated the Q-values optimal in Task B with the lowest error.

B.2.4 Combination Lock Experiment

For the combination lock simulations presented in Figure 3.8, each Q-learning configuration was tested independently on each task with learning rates 0.1, 0.5, and 0.9. Because Q-values were initialized optimistically with a value of 1.0, each plot in Figure 3.8 uses a learning rate of 0.9.

To find a reward-predictive state representation, the LSFM loss objective $\mathcal{L}_{\text{LSFM-mat}}$ (Equation (B.2)) was optimized 100000 iterations using Tensorflow's Adam optimizer with a learning rate of 0.005 and $\alpha_\psi = 0.01$.

B.3 Implementation Details for Chapter 4 Simulations

B.3.1 State Abstractions in Tabular Tasks

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

$$\forall a \in \mathcal{A},\ \forall i, j \in \mathcal{S},\ \boldsymbol{M}_a(i,j) = p(i,a,j), \tag{B.7}$$

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

$$\forall a \in \mathcal{A},\ \forall i \in \mathcal{S},\ \boldsymbol{r}_a(i) = \mathbb{E}_{s'}[r(s,a,s')]. \tag{B.8}$$

Using a weighting function ω, the latent transitions and reward "aggregate" 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

$$w(s_\phi, a) = \sum_{s:\phi(s)=s_\phi} \omega(s)\boldsymbol{r}_a(s). \tag{B.9}$$

The transition function between latent states m is constructed as

$$m(s_\phi, a, s'_\phi) = \sum_{s:\phi(s)=s_\phi} \sum_{s':\phi(s')=s'_\phi} \omega(s)p(s,a,s'). \tag{B.10}$$

Here, 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 4.2.1 and 4.2.2 and Chapter 5 assume a uniform weighting function which averages across state partitions. Similar to Equations (B.7) and (B.8), 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

$$\forall a \in \mathcal{A},\ \forall i, j \in \mathcal{S}_\phi,\ \boldsymbol{M}_a(i,j) = m(i,a,j) \text{ and } \boldsymbol{w}_a(i) = w(i,a). \tag{B.11}$$

Note that function m defines transition probabilities between latent states in the same way the transition function p defines transition probabilities between states [67]. 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 [106, 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.

B.3.2 Hyper-Parameters Used for Learning Experiments

Table B.6: Learning rates used for Q-learning (Algorithm 1) and SF-learning (Algorithm 2) in Section 4.2.3. Both algorithms are initialized optimistically, because this leads to faster convergence as described in Section 2.2.

	Tested α_{SF}	Tested α_r	Tested α	Best α_{SF}	Best α_r	Best α
SF-learning with SF transfer, orange curve in Fig. 4.6	0.1, 0.5, 0.9	0.1, 0.5, 0.9	-	0.5	0.9	-
SF-learning baseline, blue curve in Fig. 4.6	0.1, 0.5, 0.9	0.1, 0.5, 0.9	-	0.5	0.9	-
Reward predictive with SF, green curve in Fig. 4.6	0.1, 0.5, 0.9	0.1, 0.5, 0.9	-	0.5	0.9	-
Q-Learning in Fig. 4.6 Including abstraction agents	-	-	0.1,0.5,0.9	-	-	0.9

Table B.7: Hyper-parameters tested for mixture model in Section 4.2.3.

	Tested α	Tested β	Best α_{SF}	Best α_r
Reward maximizing, orange curve in Fig. 4.6(a)	$10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}$	1, 10^2, 10^4	10^{-3}	10^2
Reward predictive green curve in Fig. 4.6(a)	$10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}$	1, 10^2, 10^4	10^{-5}	10^2
Reward predictive with SF-learning, green curve in Fig. 4.6(b)	10^{-9}, 10^{-5}, 10^{-1}	1, 10^2, 10^4	10^{-9}	10^2

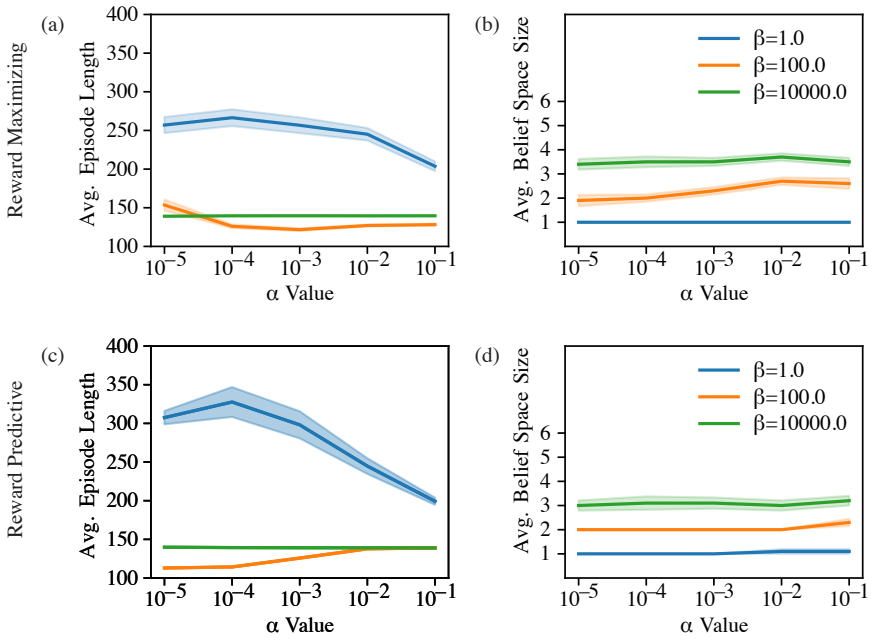


Figure B.1: Model parameters α and β control the belief space size of the non-parametric Bayesian model presented in Section 4.2.3. (a): Avg. episode length of the reward-maximizing model. (b) Avg. belief space size of the reward-maximizing model. (c) Avg. episode length of the reward-predictive model. (d) Avg. belief space size of the reward-predictive model.

Table B.8: Hyper-Parameter used in guitar example in Section 4.2.4. For each agent the SF-learning algorithm (Algorithm 2) was used and initialized optimistically as described in Section 2.2.

	Tested α_{SF}	Tested α_r	Best α_{SF}	Best α_r
SF-learning Baseline	0.1, 0.5, 0.9	0.1, 0.5, 0.9	0.1	0.9
SF-learning with SF transfer	0.1, 0.5, 0.9	0.1, 0.5, 0.9	0.9	0.9
Reward predictive with SF-learning	0.1, 0.5, 0.9	0.1, 0.5, 0.9	0.9	0.9

B.4 Implementation Details of Chapter 5 Simulations

B.4.1 Computing LSFMs

The weight matrices and vectors $\{\boldsymbol{F}_a, \boldsymbol{w}_a\}_{a\in\mathcal{A}}$ of a LSFM are estimated using linear regression. Specifically, given a transition data set $\mathcal{D}$, the weight vector $\boldsymbol{w}_a$ is estimated such that

$$\boldsymbol{w}_a = \arg\min_{\boldsymbol{v}\in\mathbb{R}^n} \sum_{(s,a,r,s')\in\mathcal{D}} (\boldsymbol{e}_{c(s)}^\top \boldsymbol{v} - r)^2. \tag{B.12}$$

Note that the clustering algorithm always constructs a representation mapping states to one-hot vectors using a cluster function c. To compute the SF matrices $\{\boldsymbol{F}_a\}_{a\in\mathcal{A}}$ the latent transition matrices of a LAM are first estimated $\{\boldsymbol{M}_a\}_{a\in\mathcal{A}}$ such that

$$\boldsymbol{M}_a = \arg\min_{\boldsymbol{M}\in\mathbb{R}^{n\times n}} \sum_{(s,a,r,s')\in\mathcal{D}} (\boldsymbol{e}_{c(s)}^\top \boldsymbol{M} - \boldsymbol{e}_{c(s')}^\top)^2. \tag{B.13}$$

By Lemma 4, the latent transition matrices $\{\boldsymbol{M}_a\}_{a\in\mathcal{A}}$ of a LAM are related to the SF matrices $\{\boldsymbol{F}_a\}_{a\in\mathcal{A}}$ with the identity

$$\boldsymbol{F}_a = \boldsymbol{I} + \gamma \boldsymbol{M}_a \overline{\boldsymbol{F}}, \tag{B.14}$$

where γ is the discount factor used by the LSFM. Furthermore, we have

$$\overline{\boldsymbol{F}}^{-1} = \boldsymbol{I} - \gamma \overline{\boldsymbol{M}}. \tag{B.15}$$

Because the state representation used by the clustering algorithm maps each state to a one-hot vector, the matrices $\{\boldsymbol{M}_a\}_{a\in\mathcal{A}}$ are stochastic. Consequently, Equation (B.15) and (B.14) are used to compute $\{\boldsymbol{F}_a\}_{a\in\mathcal{A}}$ given latent transition matrices $\{\boldsymbol{M}_a\}_{a\in\mathcal{A}}$.

B.4.2 Hyper-Parameter for Simulation Experiments

Table B.9: Hyper-parameters used in the image puddle world example (Section 5.2.1). The first layer receives the gray-scale image as input. The last layer does not have an activation function when the network is used to approximate one-step rewards, SFs, or Q-values (in case one of the DQN agent are used). When learning a state representation, then the last layer uses a soft-max activation function. In this case, the loss objective to train the representation network (Algorithm 3, line 11) is replaced using a cross-entropy loss. The ERM routine trains the network using the Adam optimizer [56] 5 epochs to approximate one-step rewards or SFs. The final representation network is trained for 10 epochs. The presented simulations use $\varepsilon_r = 0.5$ and $\varepsilon_\psi = 0.15$. The cluster operation is implemented using K-Means clustering implementation in SciKit learn [83] for $k = 500$. The found centroids are then merged using agglomerative clustering (UPGMC algorithm) with the L1 norm [114, 77].

Input Layer	2D Convolution	5×5 filter, 8 channels, ReLU activation fn.
	2D Convolution	3×3 filter, 8 channels
	ReLU Activation	
	2D Convolution	5×5 filter, 8 channels
	ReLU Activation	
	Linear Layer	200 outputs
	ReLU Activation	
	Linear Layer	200 outputs
	ReLU Activation	
Output Layer	Linear Layer	number of outputs equal number of cluster

Table B.10: Hyper-parameters used in the MNIST combination lock example (Section 5.2.2). The first layer receives the gray-scale image as input. The last layer does not have an activation function when the network is used to approximate one-step rewards, SFs, or Q-values (in case one of the DQN agent are used). When learning a state representation, then the last layer uses a soft-max activation function. In this case, the loss objective to train the representation network (Algorithm 3, line 11) is replaced using a cross-entropy loss. For the MNIST combination lock task the Adam optimizer [56] was used using tensorflow default parameters [1]. The ERM routine always trained the network for 80 epochs. For one-step reward and SF prediction a separate network was trained for each action. The presented simulations use $\varepsilon_r = 0.5$ and $\varepsilon_\psi = 0.15$. The cluster operation is implemented using K-Means clustering implementation in SciKit learn [83] for $k = 110$. The found centroids are then merged using agglomerative clustering (UPGMC algorithm) [114, 77] with the L1 norm.

Input Layer	2D Convolution	5×5 filter, 32 channels
	ReLU Activation	
	2D Convolution	5×5 filter, 32 channels, no bias term
	Batch Normalization	
	ReLU Activation	
	Max-Pooling	2×2 filter
	Dropout	Keep probability of 0.25
	2D Convolution	3×3 filter, 64 channels
	ReLU Activation	
	2D Convolution	3×3 filter, 64 channels, no bias term
	Batch Normalization	
	ReLU Activation	
	Max-Pooling	2×2 filter
	Dropout	Keep probability of 0.25
	Linear Layer	2048 outputs, no bias term
	Batch Normalization	
	ReLU Activation	
	Linear Layer	2048 outputs, no bias term
	Batch Normalization	
	ReLU Activation	
	Linear Layer	2048 outputs, no bias term
	Batch Normalization	
	ReLU Activation	
	Dropout	Keep probability of 0.25
Output Layer	Linear Layer	number of outputs equal number of cluster

Table B.11: Hyper-parameters used for the online learning simulations presented in Figure 5.9. Each combination of test values was tested and the combination with the shortest average episode length (the best parameter setting) is used in Figure 5.9. In all experiments, the neural networks were trained using the Adam optimizer [56] and only the learning rates are optimized leaving the other parameters to Tensorflow's [1] defaults. An ε-greedy policy is used to select actions and the ε parameter is decreased linearly from 1 to 0 for the first n exploration episodes.

	Hard-Coded Rep. Agent		Reward-Predictive Rep. Agent		DQN (both versions)	
Hyper-parameter	Tested	Best	Tested	Best	Tested	Best
Learning Rate	0.1, 0.5, 0.9	0.9	0.1, 0.5, 0.9	0.9	10^{-4}, $5 \cdot 10^{-4}$, 10^{-3}	10^{-4}
Replay Buffer Size	50, 100, 200	200	50, 100, 200	100	50, 100, 200	200
Batch Size	32	32	32	32	32	32
Exploration Episodes	2	2	8	8	2	2

Table B.12: SF-learning hyper-parameter used in the MNIST combination lock transfer example (Section 5.2.2). For each agent the SF-learning algorithm (Algorithm 2) and all weights are initialized to zero (unless SF weights are transferred).

	Tested α_{SF}	Tested α_r	Best α_{SF}	Best α_r
SF-learning Baseline	0.1, 0.5, 0.9	0.1, 0.5, 0.9	0.5	0.5
SF-learning with SF transfer	0.1, 0.5, 0.9	0.1, 0.5, 0.9	0.1	0.9
Reward predictive with SF-learning	0.1, 0.5, 0.9	0.1, 0.5, 0.9	0.1	0.9

Appendix C

Supplemental Files

Audio File S1: Sound version of the optimal policy in the scale task 1.

https://doi.org/10.1371/journal.pcbi.1008317.s007

Audio File S2: Sound version of the optimal policy in the scale task 2.

https://doi.org/10.1371/journal.pcbi.1008317.s008

Audio File S3: Sound version of the SF transfer algorithm's policy after learning for 25 episodes in scale task 2.

https://doi.org/10.1371/journal.pcbi.1008317.s009

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[2] David Abel, David Hershkowitz, and Michael Littman. Near optimal behavior via approximate state abstraction. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 2915–2923, 2016.

[3] David Abel, Dilip Arumugam, Lucas Lehnert, and Michael Littman. State abstractions for lifelong reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 10–19, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL `http://proceedings.mlr.press/v80/abel18a.html`.

[4] David Abel, Dilip Arumugam, Kavosh Asadi, Yuu Jinnai, Michael L. Littman, and Lawson L.S. Wong. State abstraction as compression in apprenticeship learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

[5] R Ellen Ambrose, Brad E Pfeiffer, and David J Foster. Reverse replay of hippocampal place cells is uniquely modulated by changing reward. *Neuron*, 91(5):1124–1136, 2016.

[6] Kavosh Asadi, Dipendra Misra, and Michael Littman. Lipschitz continuity in model-based reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 264–273, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL `http://proceedings.mlr.press/v80/asadi18a.html`.

[7] Hisham E Atallah, Michael J Frank, and Randall C O'Reilly. Hippocampus, cortex, and basal ganglia: Insights from computational models of complementary learning systems. *Neurobiology of learning and memory*, 82(3):253–267, 2004.

[8] Mohammad Gheshlaghi Azar, Ian Osband, and Rémi Munos. Minimax regret bounds for reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 263–272. JMLR.org, 2017.

[9] David Badre and Michael J Frank. Mechanisms of hierarchical reinforcement learning in cortico–striatal circuits 2: Evidence from fmri. *Cerebral cortex*, 22(3):527–536, 2011.

[10] André Barreto, Rémi Munos, Tom Schaul, and David Silver. Successor features for transfer in reinforcement learning. *CoRR*, abs/1606.05312, 2016. URL `http://arxiv.org/abs/1606.05312`.

[11] André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4055–4065, 2017.

[12] Andre Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Zidek, and Remi Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 501–510, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL `http://proceedings.mlr.press/v80/barreto18a.html`.

[13] André Barreto, Shaobo Hou, Diana Borsa, David Silver, and Doina Precup. Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, 117(48):30079–30087, 2020.

[14] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 4509–4517, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

[15] Timothy EJ Behrens, Timothy H Muller, James CR Whittington, Shirley Mark, Alon B Baram, Kimberly L Stachenfeld, and Zeb Kurth-Nelson. What is a cognitive map? organizing knowledge for flexible behavior. *Neuron*, 100(2):490–509, 2018.

[16] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019. ISSN 0027-8424. doi: 10.1073/pnas.1903070116. URL `https://www.pnas.org/content/116/32/15849`.

[17] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *CoRR*, abs/1207.4708, 2012. URL `http://arxiv.org/abs/1207.4708`.

[18] Richard E Bellman. *Adaptive Control Processes: A Guided Tour*, volume 2045. Princeton University Press, 1961.

[19] Dimitri P Bertsekas. Dynamic programming and optimal control 3rd edition, volume ii. *Belmont, MA: Athena Scientific*, 2011.

[20] Dimitri P Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

[21] Diana Borsa, André Barreto, John Quan, Daniel Mankowitz, Rémi Munos, Hado van Hasselt, David Silver, and Tom Schaul. Universal successor features approximators. *arXiv preprint arXiv:1812.07626*, 2018.

[22] Justin A Boyan and Andrew W Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in neural information processing systems*, pages 369–376, 1995.

[23] Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231, 2002.

[24] Alyssa A. Carey, Youki Tanaka, and Matthijs A. A. van der Meer. Reward revaluation biases hippocampal replay content away from the preferred outcome. *Nature Neuroscience*, 22(9):1450–1459, 2019. doi: 10.1038/s41593-019-0464-6. URL `https://doi.org/10.1038/s41593-019-0464-6`.

[25] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016.

[26] Anne Gabrielle Eva Collins and Michael Joshua Frank. Neural signature of hierarchically structured expectations predicts clustering and transfer of rule sets in reinforcement learning. *Cognition*, 152:160–169, 2016.

[27] Anne GE Collins and Michael J Frank. Opponent actor learning (opal): Modeling interactive effects of striatal dopamine on reinforcement learning and choice incentive. *Psychological review*, 121(3):337, 2014.

[28] Gheorghe Comanici, Doina Precup, and Prakash Panangaden. Basis refinement strategies for linear value function approximation in MDPs. In *Advances in Neural Information Processing Systems*, pages 2899–2907, 2015.

[29] Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993.

[30] Carlo D'Eramo, Davide Tateo, Andrea Bonarini, Marcello Restelli, and Jan Peters. Sharing knowledge in multi-task deep reinforcement learning. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rkgpv2VFvr`.

[31] Carlos Diuk, Andre Cohen, and Michael L. Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 240–247, New York, NY, USA, 2008. Association

for Computing Machinery. ISBN 9781605582054. doi: 10.1145/1390156.1390187. URL https://doi.org/10.1145/1390156.1390187.

[32] Simon S Du, Akshay Krishnamurthy, Nan Jiang, Alekh Agarwal, Miroslav Dudík, and John Langford. Provably efficient rl with rich observations via latent state decoding. *arXiv preprint arXiv:1901.09018*, 2019.

[33] Eyal Even-Dar and Yishay Mansour. Approximate equivalence of Markov decision processes. In *Learning Theory and Kernel Machines*, pages 581–594. Springer, 2003.

[34] Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite markov decision processes. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 162–169. AUAI Press, 2004.

[35] Norm Ferns, Prakash Panangaden, and Doina Precup. Bisimulation metrics for continuous Markov decision processes. *SIAM Journal on Computing*, 40(6):1662–1714, 2011.

[36] Norman Ferns and Doina Precup. Bisimulation metrics are optimal value functions. In *UAI*, pages 210–219. Citeseer, 2014.

[37] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[38] Vincent François-Lavet, Yoshua Bengio, Doina Precup, and Joelle Pineau. Combined reinforcement learning via abstract representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3582–3589, 2019.

[39] Michael J Frank and David Badre. Mechanisms of hierarchical reinforcement learning in corticostriatal circuits 1: computational analysis. *Cerebral cortex*, 22(3):509–526, 2012.

[40] Nicholas T Franklin and Michael J Frank. Compositional clustering in task structure learning. *PLoS computational biology*, 14(4):e1006116, 2018.

[41] Nicholas T Franklin and Michael J Frank. Generalizing to generalize: when (and when not) to be compositional in task structure learning. *bioRxiv*, 2019. doi: 10.1101/547406.

[42] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adverserial inverse reinforcement learning. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rkHywl-A-`.

[43] Mona M Garvert, Raymond J Dolan, and Timothy EJ Behrens. A map of abstract relational knowledge in the human hippocampal–entorhinal cortex. *Elife*, 6:e17086, 2017.

[44] Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G Bellemare. DeepMDP: Learning continuous latent space models for representation learning. In *International Conference on Machine Learning*, pages 2170–2179, 2019.

[45] Samuel J Gershman and David M Blei. A tutorial on bayesian nonparametric models. *Journal of Mathematical Psychology*, 56(1):1–12, 2012.

[46] Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147(1):163–223, 2003.

[47] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. JMLR Workshop and Conference Proceedings. URL `http://proceedings.mlr.press/v9/glorot10a.html`.

[48] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[49] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored mdps. *Journal of Artificial Intelligence Research*, 19:399–468, 2003.

[50] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[51] Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(Apr):1563–1600, 2010.

[52] Chi Jin, Zeyuan Allen-Zhu, Sebastien Bubeck, and Michael I Jordan. Is Q-learning provably efficient? In *Advances in Neural Information Processing Systems*, pages 4863–4873, 2018.

[53] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[54] Ken Kansky, Tom Silver, David A Mély, Mohamed Eldawy, Miguel Lázaro-Gredilla, Xinghua Lou, Nimrod Dorfman, Szymon Sidor, Scott Phoenix, and Dileep George. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. *arXiv preprint arXiv:1706.04317*, 2017.

[55] Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *arXiv preprint arXiv:2012.13490*, 2020.

[56] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

[57] Jon Kleinberg. An impossibility theorem for clustering. *Advances in neural information processing systems*, pages 463–470, 2003.

[58] Donald E Knuth. *The Art of Computer Programming, Volume 4, Fascicle 3: Generating All Combinations and Partitions*. Addison-Wesley, 2005.

[59] George Konidaris, Sarah Osentoski, and Philip Thomas. Value function approximation in reinforcement learning using the Fourier basis. *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages pages 380–385, August 2011.

[60] Mirko Křivánek and Jaroslav Morávek. Np-hard problems in hierarchical-tree clustering. *Acta informatica*, 23(3):311–323, 1986.

[61] Tejas D Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J Gershman. Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*, 2016.

[62] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

[63] Lucas Lehnert and Michael L Littman. Successor features combine elements of model-free and model-based reinforcement learning. *Journal of Machine Learning Research*, 21(196):1–53, 2020.

[64] Lucas Lehnert, Stefanie Tellex, and Michael L Littman. Advantages and limitations of using successor features for transfer in reinforcement learning. *arXiv preprint arXiv:1708.00102*, 2017.

[65] Lucas Lehnert, Michael L Littman, and Michael J Frank. Reward-predictive representations generalize across tasks in reinforcement learning. *PLoS computational biology*, 16(10):e1008317, 2020.

[66] Felix Leibfried, Nate Kushman, and Katja Hofmann. A deep learning approach for joint video frame and reward prediction in atari games. *arXiv preprint arXiv:1611.07078*, 2016.

[67] Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for MDPs. In *ISAIM*, 2006.

[68] Michael L Littman, Anthony R Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *Machine Learning Proceedings 1995*, pages 362–370. Elsevier, 1995.

[69] Yunzhe Liu, Raymond J Dolan, Zeb Kurth-Nelson, and Timothy EJ Behrens. Human replay spontaneously reorganizes experience. *Cell*, 178(3):640–652, 2019.

[70] Tamas Madarasz and Tim Behrens. Better transfer learning with inferred successor maps. In *Advances in Neural Information Processing Systems*, pages 9029–9040, 2019.

[71] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is np-hard. In Sandip Das and Ryuhei Uehara, editors, *WALCOM: Algorithms and Computation*, pages 274–285, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-00202-1.

[72] Marcelo G Mattar and Nathaniel D Daw. Prioritized memory access explains planning and hippocampal replay. *Nature neuroscience*, 21(11):1609–1617, 2018.

[73] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[74] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. MIT Press, 2018.

[75] Ida Momennejad, Evan M Russek, Jin H Cheong, Matthew M Botvinick, ND Daw, and Samuel J Gershman. The successor representation in human reinforcement learning. *Nature Human Behaviour*, 1(9):680, 2017.

[76] P Read Montague, Peter Dayan, and Terrence J Sejnowski. A framework for mesencephalic dopamine systems based on predictive hebbian learning. *Journal of neuroscience*, 16(5):1936–1947, 1996.

[77] Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms, 2011.

[78] Helen M Nasser, Donna J Calu, Geoffrey Schoenbaum, and Melissa J Sharpe. The dopamine prediction error: contributions to associative models of reward learning. *Frontiers in psychology*, 8:244, 2017.

[79] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems*, pages 2863–2871, 2015.

[80] Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. *arXiv preprint arXiv:1707.03497*, 2017.

[81] Ian Osband, Daniel Russo, and Benjamin Van Roy. (More) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems*, pages 3003–3011, 2013.

[82] Ronald Parr, Lihong Li, Gavin Taylor, Christopher Painter-Wakefield, and Michael L Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 752–759. ACM, 2008.

[83] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[84] Sébastien Racanière, Théophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomenech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li,

et al. Imagination-augmented agents for deep reinforcement learning. In *Advances in neural information processing systems*, pages 5690–5701, 2017.

[85] Balaraman Ravindran and Andrew G Barto. Approximate homomorphisms: A framework for non-exact minimization in markov decision processes. 2004.

[86] A David Redish, Steve Jensen, Adam Johnson, and Zeb Kurth-Nelson. Reconciling reinforcement learning models with behavioral extinction and renewal: implications for addiction, relapse, and problem gambling. *Psychological review*, 114(3):784, 2007.

[87] Martin Riedmiller. Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer, 2005.

[88] Sherry Shanshan Ruan, Gheorghe Comanici, Prakash Panangaden, and Doina Precup. Representation discovery for mdps using bisimulation metrics. In *AAAI*, pages 3578–3584, 2015.

[89] Evan M Russek, Ida Momennejad, Matthew M Botvinick, Samuel J Gershman, and Nathaniel D Daw. Predictive representations can link model-based reinforcement learning to model-free mechanisms. *PLoS computational biology*, 13(9):e1005768, 2017.

[90] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

[91] Anna C Schapiro, Nicholas B Turk-Browne, Kenneth A Norman, and Matthew M Botvinick. Statistical learning of temporal community structure in the hippocampus. *Hippocampus*, 26 (1):3–8, 2016.

[92] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

[93] Nicolas W Schuck and Yael Niv. Sequential replay of nonspatial task states in the human hippocampus. *Science*, 364(6447):eaaw5181, 2019.

[94] Nicolas W Schuck, Ming Bo Cai, Robert C Wilson, and Yael Niv. Human orbitofrontal cortex represents a cognitive map of state space. *Neuron*, 91(6):1402–1412, 2016.

[95] Melissa J Sharpe, Chun Yun Chang, Melissa A Liu, Hannah M Batchelor, Lauren E Mueller, Joshua L Jones, Yael Niv, and Geoffrey Schoenbaum. Dopamine transients are sufficient and necessary for acquisition of model-based associations. *Nature Neuroscience*, 20(5):735, 2017.

[96] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529 (7587):484–489, 2016.

[97] David Silver, Hado Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, et al. The predictron: End-to-end learning and planning. In *International Conference on Machine Learning*, pages 3191–3199. PMLR, 2017.

[98] Zhao Song, Ronald E Parr, Xuejun Liao, and Lawrence Carin. Linear feature encoding for reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4224–4232, 2016.

[99] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[100] Kimberly L Stachenfeld, Matthew M Botvinick, and Samuel J Gershman. The hippocampus as a predictive map. *Nature Neuroscience*, 20(11):1643, 2017.

[101] Alexander L Strehl, Lihong Li, and Michael L Littman. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research*, 10(Nov):2413–2444, 2009.

[102] Richard S. Sutton. Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3(1):9–44, August 1988.

[103] Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the seventh international conference on machine learning*, pages 216–224, 1990.

[104] Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, pages 1038–1044, 1996.

[105] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book. MIT Press, Cambridge, MA, 1 edition, 1998.

[106] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[107] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[108] Richard S. Sutton, Csaba Szepesvári, Alborz Geramifard, and Michael Bowling. Dyna-style planning with linear function approximation and prioritized sweeping. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, 2008.

[109] Erik Talvitie. Self-correcting models for model-based reinforcement learning. In *AAAI*, pages 2597–2603, 2017.

[110] Erik Talvitie. Learning the reward function for a misspecified model. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4838–4847, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL `http://proceedings.mlr.press/v80/talvitie18a.html`.

[111] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

[112] Yee Whye Teh, Michael I Jordan, Matthew J Beal, and David M Blei. Hierarchical dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581, 2006. doi: 10.1198/016214506000000302. URL `https://doi.org/10.1198/016214506000000302`.

[113] Vladimir Vapnik. Principles of risk minimization for learning theory. In *Advances in neural information processing systems*, pages 831–838, 1992.

[114] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van

der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[115] Christopher J.C.H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[116] Théophane Weber, Sébastien Racanière, David P Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. *arXiv preprint arXiv:1707.06203*, 2017.

[117] James Whittington, Timothy Muller, Shirely Mark, Caswell Barry, and Tim Behrens. Generalisation of structural knowledge in the hippocampal-entorhinal system. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8484–8495. Curran Associates, Inc., 2018.

[118] James CR Whittington, Timothy H Muller, Shirley Mark, Guifen Chen, Caswell Barry, Neil Burgess, and Timothy EJ Behrens. The tolman-eichenbaum machine: Unifying space and relational memory through generalisation in the hippocampal formation. *bioRxiv*, page 770495, 2019.

[119] Robert C Wilson, Yuji K Takahashi, Geoffrey Schoenbaum, and Yael Niv. Orbitofrontal cortex as a cognitive map of task space. *Neuron*, 81(2):267–279, 2014.

[120] Sam Witty, Jun Ki Lee, Emma Tosch, Akanksha Atrey, Michael Littman, and David Jensen. Measuring and characterizing generalization in deep reinforcement learning. *arXiv preprint arXiv:1812.02868*, 2018.

[121] Hengshuai Yao and Csaba Szepesvári. Approximate policy iteration with linear action models. In *AAAI*, 2012.

[122] Amy Zhang, Shagun Sodhani, Khimya Khetarpal, and Joelle Pineau. Learning robust state abstractions for hidden-parameter block mdps, 2021.

[123] Jingwei Zhang, Jost Tobias Springenberg, Joschka Boedecker, and Wolfram Burgard. Deep reinforcement learning with successor features for navigation across similar environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2371–2378. IEEE, 2017.