Published in the proceedings of the Workshop on "Structure & Priors in Reinforcement Learning" at ICLR 2019

PROVABLY EFFICIENT RL WITH RICH OBSERVATIONS VIA LATENT STATE DECODING

Simon S. Du
CMU

Akshay Krishnamurthy
MSR, NY

Nan Jiang
UIUC

Alekh Agarwal
MSR, Redmond

Miroslav Dudík
MSR, NY

John Langford
MSY, NY

ABSTRACT

We study the exploration problem in episodic MDPs with rich observations generated from a small number of latent states. Under certain identifiability assumptions, we demonstrate how to estimate a mapping from the observations to latent states inductively through a sequence of regression and clustering steps—where previously decoded latent states provide labels for later regression problems—and use it to construct good exploration policies. We provide finite-sample guarantees on the quality of the learned state decoding function and exploration policies, and complement our theory with an empirical evaluation on a class of hard exploration problems. Our method exponentially improves over Q-learning with naïve exploration, even when Q-learning has cheating access to latent states.

1 INTRODUCTION

We study reinforcement learning (RL) in episodic environments with rich observations, such as images and texts. While many modern empirical RL algorithms are designed to handle such settings (see, e.g., Mnih et al., 2015), relatively few works focus on the question of strategic exploration in this literature (Ostrovski et al., 2017; Osband et al., 2016) and the sample efficiency of these techniques is not theoretically understood.

In this work we consider RL problems with *latent-state* structure. We consider recovering the latent-state structure explicitly by learning a *decoding function* (from a large set of candidates) that maps a rich observation to the corresponding latent state. We show that our algorithms are:

Provably sample-efficient: Under certain identifiability assumptions, we recover a mapping from the observations to underlying latent states as well as a good exploration policy using a number of samples which is polynomial in the number of latent states, horizon and the complexity of the decoding function class with no explicit dependence on the observation space size. Thus we significantly generalize beyond the works of Dann et al. (2018) who require deterministic dynamics and Azizzadenesheli et al. (2016) whose guarantees scale with the observation space size.

Computationally practical: Unlike many prior works in this vein, our algorithm is easy to implement and substantially outperforms naïve exploration in experiments, even when the baselines have cheating access to the latent states.

The main challenge in learning the decoding function is that the hidden states are never directly observed. Our key novelty is the use of a backward conditional probability vector (Equation 1) as a representation for latent state, and learning the decoding function via conditional probability estimation, which can be solved using least squares regression.

2 SETTING AND TASK DEFINITION

We begin by introducing some basic notation. We write $[h]$ to denote the set $\{1, \ldots, h\}$. For any finite set S, we write $U(S)$ to denote the uniform distribution over S. We write $\triangle_d$ for the simplex in $\mathbb{R}^d$. Finally, we write $\|\cdot\|$ and $\|\cdot\|_1$, respectively, for the Euclidean and the ℓ_1 norms of a vector.

2.1 BLOCK MARKOV DECISION PROCESS

In this paper we introduce and analyze a *block Markov decision process* or *BMDP*. It refers to an environment described by a finite, but unobservable *latent state space* $\mathcal{S}$, a finite *action space* $\mathcal{A}$, with $|\mathcal{A}| = K$, and a possibly infinite, but observable *context space* $\mathcal{X}$. The dynamics of a BMDP is described by the *initial state* $s_1 \in \mathcal{S}$ and two conditional probability functions: the *state-transition function* p and *context-emission function* q, defining conditional probabilities $p(s' \mid s, a)$ and $q(x \mid s)$ for all $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$, $x \in \mathcal{X}$.[1]

We consider episodic learning tasks with a finite horizon H. In each episode, the environment starts in the state s_1. In the step $h \in [H]$ of an episode, the environment generates a context $x_h \sim q(\cdot \mid s_h)$, the agent observes the context x_h (but not the state s_h), takes an action a_h, and the environment transitions to a new state $s_{h+1} \sim p(\cdot \mid s_h, a_h)$. The sequence $(s_1, x_1, a_1, \ldots, s_H, x_H, a_H, s_{H+1}, x_{H+1})$ generated in an episode is called a *trajectory*. We emphasize that a learning agent does not observe components s_h from the trajectory.

So far, our description resembles that of a partially observable Markov decision process (POMDP). To finish the definition of BMDP, and distinguish it from a POMDP, we make the following assumption:

Assumption 2.1 (Block structure)**.** *Each context x uniquely determines its generating state s. That is, the context space $\mathcal{X}$ can be partitioned into disjoint blocks $\mathcal{X}_s$, each containing the support of the conditional distribution $q(\cdot \mid s)$.*

The block structure implies the existence of a *perfect decoding function* $f^* : \mathcal{X} \to \mathcal{S}$, which maps contexts into their generating states. To streamline our analysis, we make a standard assumption for episodic settings. We assume that $\mathcal{S}$ can be partitioned into disjoints sets $\mathcal{S}_h$, $h \in [H+1]$, such that $p(\cdot \mid s, a)$ is supported on $\mathcal{S}_{h+1}$ whenever $s \in \mathcal{S}_h$. We refer to h as the *level* and assume that it is observable as part of the context, so the context space is also partitioned into sets $\mathcal{X}_h$. We use notation $\mathcal{S}_{[h]} = \cup_{\ell \in [h]} \mathcal{S}_\ell$ for the set of states up to level h, and similarly define $\mathcal{X}_{[h]} = \cup_{\ell \in [h]} \mathcal{X}_\ell$. We assume that $|\mathcal{S}_h| \leq M$. We seek learning algorithms that scale polynomially in parameters M, K and H, but do not explicitly depend on $|\mathcal{X}|$, which might be infinite.

2.2 SOLUTION CONCEPT: COVER OF EXPLORATORY POLICIES

In this paper, we focus on the problem of exploration. Specifically, for each state $s \in \mathcal{S}$, we seek an agent strategy for reaching that state s. We formalize an agent strategy as an *h-step policy*, which is a map $\pi : \mathcal{X}_{[h]} \to \mathcal{A}$ specifying which action to take in each context up to step h. When executing an h-step policy π with $h < H$, an agent acts according to π for h steps and then arbitrarily until the end of the episode (e.g., according to a specific default policy).

For an h-step policy π, we write $\mathbb{P}^\pi$ to denote the probability distribution over h-step trajectories induced by π. We write $\mathbb{P}^\pi(\mathcal{E})$ for the probability of an event $\mathcal{E}$. For example, $\mathbb{P}^\pi(s)$ is the probability of reaching the state s when executing π.

We also consider randomized strategies, which we formalize as *policy mixtures*. An h-step *policy mixture* η is a distribution over h-step policies. When executing η, an agent randomly draws a policy $\pi \sim \eta$ at the beginning of the episode, and then follows π throughout the episode. The induced distribution over h-step trajectories is denoted $\mathbb{P}^\eta$.

Our algorithms create specific policies and policy mixtures via concatenation. Specifically, given an h-step policy π, we write $\pi \odot a$ for the $(h+1)$-step policy that executes π for h steps and chooses action a in step $h+1$. Similarly, if η is a policy mixture and ν a distribution over $\mathcal{A}$, we write $\eta \odot \nu$ for the policy mixture equivalent to first sampling and following a policy according to η and then independently sampling and following an action according to ν.

We finally introduce two key concepts related to exploration: *maximum reaching probability* and *policy cover*.

[1] For continuous context spaces, $q(\cdot \mid s)$ describes a density function relative to a suitable measure (e.g., Lebesgue measure).

Definition 2.1 (Maximum reaching probability.). *For any $s \in \mathcal{S}$, its* maximum reaching probability *$\mu(s)$ is $\mu(s) := \max_\pi \mathbb{P}^\pi(s)$, where the maximum is taken over all maps $\mathcal{X}_{[H]} \to \mathcal{A}$. The policy attaining the maximum for a given s is denoted π^*_s.*[2]

Without loss of generality, we assume that all the states are reachable, i.e., $\mu(s) > 0$ for all s. We write $\mu_{\min} = \min_{s\in\mathcal{S}} \mu(s)$ for the $\mu(s)$ value of the hardest-to-reach state. Given maximum reaching probabilities, we formalize the task of finding policies that reach states s as the task of finding an *ϵ–policy cover* in the following sense:

Definition 2.2 (Policy cover of the state space). *We say that a set of policies Π_h is an* ϵ–policy cover *of $\mathcal{S}_h$ if for all $s \in \mathcal{S}_h$ there exists an $(h-1)$-step policy $\pi \in \Pi_h$ such that $\mathbb{P}^\pi(s) \geq \mu(s) - \epsilon$. A set of policies Π is an* ϵ–policy cover *of $\mathcal{S}$ if it is an ϵ–policy cover of $\mathcal{S}_h$ for all $h \in [H+1]$.*

3 Embedding Approach

3.1 Embeddings and Function Approximation

In order to construct the decoding function f, we learn low-dimensional representations of contexts as well as latent states in a shared space, namely Δ_{MK}. We learn embedding functions $\mathbf{g} : \mathcal{X} \to \Delta_{MK}$ for contexts and $\boldsymbol{\phi} : \mathcal{S} \to \Delta_{MK}$ for states, with the goal that $\mathbf{g}(x)$ and $\boldsymbol{\phi}(s)$ should be close if and only if $x \in \mathcal{X}_s$. Such embedding functions always exist due to the block-structure: for any set of distinct vectors $\{\boldsymbol{\phi}(s)\}_{s\in\mathcal{S}}$, it suffices to define $\mathbf{g}(x) = \boldsymbol{\phi}(s)$ for $x \in \mathcal{X}_s$. We consider using function approximation to learn the embedding functions and we assume the realizability.

Assumption 3.1 (Realizability). *For any $h \in [H+1]$ and $\boldsymbol{\phi} : \mathcal{S}_h \to \triangle_{MK}$, there exists $\mathbf{g}_h \in \mathcal{G}$ such that $\mathbf{g}_h(x) = \boldsymbol{\phi}(s)$ for all $x \in \mathcal{X}_s$ and $s \in \mathcal{S}_h$.*

As we alluded to earlier, we learn context embeddings $\mathbf{g}_h$ by solving supervised learning problems. In fact, we only require the ability to solve least squares problems. Specifically, we assume access to an algorithm for solving vector-valued least-squares regression over the class $\mathcal{G}$. We refer to such an algorithm as the ERM oracle:

Definition 3.1 (ERM Oracle). *Let $\mathcal{G}$ be a function class that maps $\mathcal{X}$ to $\triangle_{MK}$. An* empirical risk minimization oracle (ERM oracle) *for $\mathcal{G}$ is any algorithm that takes as input a data set $D = \{(x_i, \mathbf{y}_i)\}_{i=1}^n$ with $x_i \in \mathcal{X}$, $\mathbf{y}_i \in \triangle_{MK}$, and computes $\operatorname{argmin}_{\mathbf{g}\in\mathcal{G}} \sum_{(x,\mathbf{y})\in D} \|\mathbf{g}(x) - \mathbf{y}\|^2$.*

3.2 Backward Probability Vectors and Separability

For any distribution $\mathbb{P}$ over trajectories, we define *backward probabilities* as the conditional probabilities of the form $\mathbb{P}(s_{h-1}, a_{h-1} \mid s_h)$. For any distribution ν over (s_{h-1}, a_{h-1}) , any $s \in \mathcal{S}_{h-1}$, $a \in \mathcal{A}$ and $s' \in \mathcal{S}_h$, the backward probability is defined as

$$b_\nu(s, a \mid s') = \frac{p(s' \mid s, a)\, \nu(s, a)}{\sum_{\tilde{s},\tilde{a}} p(s' \mid \tilde{s}, \tilde{a})\, \nu(\tilde{s}, \tilde{a})}. \tag{1}$$

For a given $s' \in \mathcal{S}_h$, we collect the probabilities $b_\nu(s, a \mid s')$ across all $s \in \mathcal{S}_{h-1}$, $a \in \mathcal{A}$ into the *backward probability vector* $\mathbf{b}_\nu(s') \in \triangle_{MK}$, padding with zeros if $|\mathcal{S}_{h-1}| < M$. Backward probability vectors are at the core of our approach, because they correspond to the state embeddings $\boldsymbol{\phi}(s)$ approximated by our algorithms. Our algorithms require that $\mathbf{b}_\nu(s')$ for different states $s' \in \mathcal{S}_h$ are sufficiently separated from one other for a suitable choice of ν:

Assumption 3.2 (γ-Separability). *There exists $\gamma > 0$ such that for any $h \in \{2, \ldots, H+1\}$ and any distinct $s', s'' \in \mathcal{S}_h$, the backward probability vectors with respect to the uniform distribution are separated by a margin of at least γ, i.e., $\|\mathbf{b}_\nu(s') - \mathbf{b}_\nu(s'')\|_1 \geq \gamma$, where $\nu = U(\mathcal{S}_{h-1} \times \mathcal{A})$.*

The key property that makes vectors $\mathbf{b}_\nu(s')$ algorithmically useful is that they arise as solutions to a specific least squares problem with respect to data generated by a policy whose marginal distribution over (s_{h-1}, a_{h-1}) matches ν. Let $\mathbf{e}_{(s,a)}$ denote the vector of the standard basis in $\mathbb{R}^{MK}$ corresponding to the coordinate indexed by $(s, a) \in \mathcal{S}_{h-1} \times \mathcal{A}$. Then the following statement holds:

[2] It suffices to consider maps $\mathcal{X}_{[h]} \to \mathcal{A}$ for $s \in \mathcal{S}_{h+1}$.

Theorem 3.1. *Let ν be a distribution supported on $\mathcal{S}_{h-1} \times \mathcal{A}$ and let $\tilde{\nu}$ be a distribution over (s, a, x') defined by sampling $(s, a) \sim \nu$, $s' \sim p(\cdot \mid s, a)$, and $x' \sim q(\cdot \mid s')$. Let $\mathbf{g}_h \in \operatorname{argmin}_{\mathbf{g} \in \mathcal{G}} \mathbb{E}_{\tilde{\nu}} \left[\left\| \mathbf{g}(x') - \mathbf{e}_{(s,a)} \right\|^2 \right]$. Then, under Assumption 3.1, every minimizer $\mathbf{g}_h$ satisfies $\mathbf{g}_h(x') = \mathbf{b}_\nu(s')$ for all $x' \in \mathcal{X}_{s'}$ and $s' \in \mathcal{S}_h$.*

4 Algorithm for Separable BMDPs

With the main components defined, we can now derive our algorithm for learning a policy cover in a separable BMDP.

The algorithm proceeds inductively, level by level. On each level h, we learn the following objects:

- The set of discovered latent states $\widehat{\mathcal{S}}_h \subseteq [M]$ and a decoding function $\hat{f}_h : \mathcal{X} \to \widehat{\mathcal{S}}_h$, which allows us to identify latent states at level h from observed contexts.
- The estimated transition probabilities $\hat{p}(\hat{s}_h \mid \hat{s}_{h-1}, a)$ across all $\hat{s}_{h-1} \in \widehat{\mathcal{S}}_{h-1}$, $a \in \mathcal{A}$, $\hat{s}_h \in \widehat{\mathcal{S}}_h$.
- A set of $(h-1)$-step policies $\Pi_h = \{\pi_{\hat{s}}\}_{\hat{s} \in \widehat{\mathcal{S}}_h}$.

Algorithm 1 constructs $\widehat{\mathcal{S}}_h$, $\hat{f}_h$, $\hat{p}$ and Π_h level by level. Given these objects up to level $h-1$, the construction for the next level h proceeds in the following three steps, annotated with the lines in Algorithm 1 where they appear:

(1) Regression step: learn $\hat{\mathbf{g}}_h$ (lines 7–9). We collect a dataset of trajectories by repeatedly executing a specific policy mixture η_h. We use $\hat{f}_{h-1}$ to identify $\hat{s}_{h-1}{=}\hat{f}_{h-1}(x_{h-1})$ on each trajectory, obtaining samples $(\hat{s}_{h-1}, a_{h-1}, x_h)$ from $\tilde{\nu}$ induced by η_h. The context embedding $\hat{\mathbf{g}}_h$ is then obtained by solving the empirical version of the regression problem in Theorem 3.1.

(2) Clustering step: learn $\widehat{\boldsymbol{\phi}}$ and $\hat{f}_h$ (lines 10–12). Thanks to Theorem 3.1, we expect that $\hat{\mathbf{g}}_h(x') \approx \mathbf{g}_h(x') = \mathbf{b}_\nu(s')$ for the distribution $\nu(\hat{s}_{h-1}, a_{h-1})$ induced by η_h.[3] Thus, all contexts x' generated by the same latent state s' have embedding vectors $\hat{\mathbf{g}}_h(x')$ close to each other and to $\mathbf{b}_\nu(s')$. Thanks to separability, we can therefore use clustering to identify all contexts generated by the same latent state, and this procedure is sample-efficient since the embeddings are low-dimensional vectors. Each cluster corresponds to some latent state s' and any vector $\hat{\mathbf{g}}_h(x')$ from that cluster can be used to define the state embedding $\widehat{\boldsymbol{\phi}}(s')$. The decoding function $\hat{f}_h$ is defined to map any context x' to the state s' whose embedding $\widehat{\boldsymbol{\phi}}(s')$ is the closest to $\hat{\mathbf{g}}_h(x')$.

(3) Dynamic programming: construct Π_h (lines 13–19). Finally, with the ability to identify states at level h via $\hat{f}_h$, we can use collected trajectories to learn an approximate transition model $\hat{p}(\hat{s}' \mid \hat{s}, a)$ up to level h. This allows us to use dynamic programming to find policies that (approximately) optimize the probability of reaching any specific state $s' \in \mathcal{S}_h$. The dynamic programming finds policies $\psi_{\hat{s}'}$ that act by directly observing decoded latent states. The policies $\pi_{\hat{s}'}$ are obtained by composing $\psi_{\hat{s}'}$ with the decoding functions $\{\hat{f}_\ell\}_{\ell \in [h-1]}$.

The next theorem guarantees that with a polynomial number of samples, Algorithm 1 finds a small ϵ–policy cover.[4]

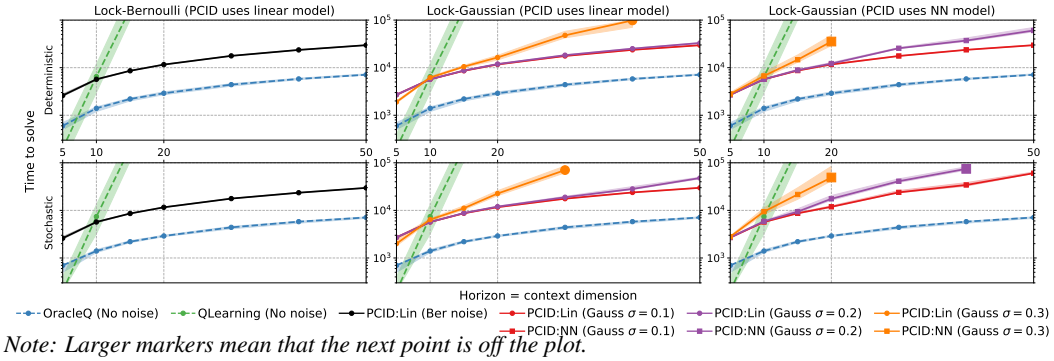
Theorem 4.1 (Sample Complexity of Algorithm 1)**.** *Fix any $\epsilon = O\Big(\frac{\mu_{\min}^3 \gamma}{M^4 K^3 H}\Big)$ and a failure probability $\delta > 0$. Set $N_{\mathrm{g}} = \tilde{\Omega}\Big(\frac{M^4 K^4 H \log |\mathcal{G}|}{\epsilon \mu_{\min}^3 \gamma^2}\Big)$, $N_\phi = \tilde{\Theta}\Big(\frac{MK}{\mu_{\min}}\Big)$, $N_{\mathrm{p}} = \tilde{\Omega}\Big(\frac{M^2 K H^2}{\mu_{\min} \epsilon^2}\Big)$, $\tau = \frac{\gamma}{30MK}$. Then with probability at least $1 - \delta$, Algorithm 1 returns an ϵ–policy cover of $\mathcal{S}$, with size at most MH.*[5]

In addition to dependence on the usual parameters like M, K, H and $1/\epsilon$, our sample complexity also scales inversely with the separability margin γ and the worst-case reaching probability $\mu_{\min}$. Compared with Azizzadenesheli et al. (2016), there is no explicit dependence on $|\mathcal{X}|$, although they make spectral assumptions instead of the explicit block structure.

[3] Theorem 3.1 uses distributions ν and $\tilde{\nu}$ over true states s_{h-1}, but its analog also holds for distributions over $\hat{s}_{h-1}$, as long as decoding is approximately correct at the previous level.

[4] The $\tilde{O}(\cdot)$, $\tilde{\Omega}(\cdot)$, and $\tilde{\Theta}(\cdot)$ notation suppresses factors that are polynomial in $\log M$, $\log K$, $\log H$ and $\log(1/\delta)$.

[5] We refer readers to https://arxiv.org/abs/1901.09018 for the full proof.



Note: Larger markers mean that the next point is off the plot.

Figure 1: Time-to-solve against problem difficulty for the Lock environment with two observation processes and function classes. Left: *Lock-Bernoulli* environment. Center: *Lock-Gaussian* with linear functions, Right: *Lock-Gaussian* with neural networks. Top row: deterministic latent transitions. Bottom row: stochastic transitions with switching probability 0.1. ORACLEQ and QLEARNING are cheating and operate directly on latent states.

5 EXPERIMENTS

The environments. All environments share the same latent structure, and are a form of "combination lock," with H levels, 3 states per level, and 4 actions. Non-zero reward is only achievable from states $s_{1,h}$ and $s_{2,h}$. From $s_{1,h}$ and $s_{2,h}$ one action leads with probability $1-\alpha$ to $s_{1,h+1}$ and with probability α to $s_{2,h+1}$, another has the flipped behavior, and the remaining two lead to $s_{3,h+1}$. All actions from $s_{3,h}$ lead to $s_{3,h+1}$. The "good" actions are randomly assigned for every state. From $s_{1,H}$ and $s_{2,H}$, two actions receive Ber$(1/2)$ reward; all others provide zero reward. The start state is $s_{1,1}$. We consider deterministic variant ($\alpha = 0$) and stochastic variant ($\alpha = 0.1$).

We also consider two observation processes, which we use only for our algorithm, while the baselines operate directly on the latent state space. In *Lock-Bernoulli*, the observation space is $\{0,1\}^{H+3}$ where the first 3 coordinates are reserved for one-hot encoding of the state and the last H coordinates are drawn iid from Ber$(1/2)$. In *Lock-Gaussian*, the observation space is $\mathbb{R}^{H+3}$. As before the first 3 coordinates are reserved for one-hot encoding of the state, but this encoding is corrupted with Gaussian noise. Formally, if the agent is at state $s_{i,h}$ the observation is $\mathbf{e}_i + \mathbf{v} \in \mathbb{R}^{3+H}$, where $\mathbf{e}_i$ is one of the first three standard basis vectors and $\mathbf{v}$ has $\mathcal{N}(0,\sigma^2)$ entries. We consider $\sigma \in \{0.1, 0.2, 0.3\}$.

Baselines, hyperparameters. We compare our algorithms against two *tabular* approaches that cheat by directly accessing the latent states. The first, ORACLEQ, is the Optimistic Q-Learning algorithm of Jin et al. (2018). The second, QLEARNING, is tabular Q-learning with ϵ-greedy exploration. This algorithm serves as a baseline: any algorithm with strategic exploration should vastly outperform QLEARNING, even though it is cheating.

Results. The results are in Figure 1 in a log-linear plot. First, QLEARNING works well for small horizon problems but cannot solve problems with $H \geq 15$ within 100K episodes, which is not surprising.[6] The performance curve for QLEARNING is linear, revealing an exponential sample complexity, and demonstrating that these environments cannot be solved with naïve exploration. As a second observation, ORACLEQ performs extremely well, and as we verify in Appendix **??** demonstrates a linear scaling with H.[7]

In *Lock-Bernoulli*, PCID is roughly a factor of 5 worse than the skyline ORACLEQ for all values of H, but the curves have similar behavior. Of course PCID is an exponential improvement over QLEARNING with ϵ-greedy exploration here. In *Lock-Gaussian* with linear functions, the results are similar for the low-noise setting, but the performance of PCID degrades as the noise level increases. For example, with noise level $\sigma = 0.3$, it fails to solve the stochastic problem with $H = 40$ in 100K episodes. On the other hand, the performance is still quite good, and the scaling represents a dramatic improvement over QLEARNING.

[6] We actually ran QLEARNING for 1M episodes and found it solves $H = 15$ with 170K episodes.

[7] This is incomparable with the result in Jin et al. (2018) since we are not measuring regret here.

REFERENCES

Kamyar Azizzadenesheli, Alessandro Lazaric, and Animashree Anandkumar. Reinforcement learning of POMDPs using spectral methods. In *Conference on Learning Theory*, 2016.

Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 2002.

Christoph Dann, Nan Jiang, Akshay Krishnamurthy, Alekh Agarwal, John Langford, and Robert E Schapire. On oracle-efficient PAC reinforcement learning with rich observations. In *Advances in Neural Information Processing Systems*, 2018.

Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 2010.

Nan Jiang, Akshay Krishnamurthy, Alekh Agarwal, John Langford, and Robert E. Schapire. Contextual decision processes with low Bellman rank are PAC-learnable. In *International Conference on Machine Learning*, 2017.

Chi Jin, Zeyuan Allen-Zhu, Sebastien Bubeck, and Michael I Jordan. Is Q-learning provably efficient? In *Advances in Neural Information Processing Systems*, 2018.

Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The Malmo Platform for artificial intelligence experimentation. In *International Joint Conference on Artificial Intelligence*, 2016.

Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine learning*, 2002.

Akshay Krishnamurthy, Alekh Agarwal, and John Langford. PAC reinforcement learning with rich observations. In *Advances in Neural Information Processing Systems*, 2016.

Tor Lattimore and Marcus Hutter. PAC bounds for discounted MDPs. In *International Conference on Algorithmic Learning Theory*, 2012.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

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

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. In *Advances in Neural Information Processing Systems*, 2016.

Georg Ostrovski, Marc G Bellemare, Aaron van den Oord, and Rémi Munos. Count-based exploration with neural density models. In *International Conference on Machine Learning*, 2017.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, 2017.

David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, and Thomas Degris. The predictron: End-to-end learning and planning. In *International Conference on Machine Learning*, 2017.

Algorithm 1 PCID (**P**olicy **C**over via **I**nductive **D**ecoding)

Input:
  N_{g}: sample size for learning context embeddings
  N_{ϕ}: sample size for learning state embeddings
  N_{p}: sample size for estimating transition probabilities
  $\tau > 0$: a clustering threshold for learning latent states

Output: policy cover $\Pi = \Pi_1 \cup \cdots \cup \Pi_{H+1}$
Let $\widehat{\mathcal{S}}_1 = \{s_1\}$. Let $\hat{f}_1(x) = s_1$ for all $x \in \mathcal{X}$.
Let $\Pi_1 = \{\pi_0\}$ where π_0 is the trivial 0-step policy.
Initialize $\hat{p}$ to an empty mapping.
for $h = 2, \ldots, H+1$ **do**
  Let $\eta_h = U(\Pi_{h-1}) \odot U(\mathcal{A})$
  Execute η_h for N_{g} times. $D_{\mathrm{g}} = \{\hat{s}^i_{h-1}, a^i_{h-1}, x^i_h\}_{i=1}^{N_{\mathrm{g}}}$
  for $\hat{s}_{h-1} = \hat{f}_{h-1}(x_{h-1})$.
  Learn $\hat{\mathbf{g}}_h$ by calling ERM oracle on input D_{g}:
  $\hat{\mathbf{g}}_h = \mathrm{argmin}_{\mathbf{g} \in \mathcal{G}} \sum_{(\hat{s},a,x') \in D_{\mathrm{g}}} \left\| \mathbf{g}(x') - \mathbf{e}_{(\hat{s},a)} \right\|^2$.
  Execute η_h for N_{ϕ} times. $\mathcal{Z} = \{\widehat{\mathbf{z}}_i = \hat{\mathbf{g}}_h(x^i_h)\}_{i=1}^{N_{\phi}}$.
  Learn $\widehat{\mathcal{S}}_h$ and the state embedding map $\widehat{\boldsymbol{\phi}}_h : \widehat{\mathcal{S}}_h \to \mathcal{Z}$
  by clustering $\mathcal{Z}$ with threshold τ (see Algorithm 2).
  Define $\hat{f}_h(x') = \mathrm{argmin}_{\hat{s} \in \widehat{\mathcal{S}}_h} \left\| \widehat{\boldsymbol{\phi}}(\hat{s}) - \hat{\mathbf{g}}_h(x') \right\|_1$.
  Execute η_h for N_{p} times. $D_{\mathrm{p}} = \{\hat{s}^i_{h-1}, a^i_{h-1}, \hat{s}^i_h\}_{i=1}^{N_{\mathrm{p}}}$
  for $\hat{s}_{h-1} = \hat{f}_{h-1}(x_{h-1})$, $\hat{s}_h = \hat{f}_h(x_h)$.
  Define $\hat{p}(\hat{s}_h \mid \hat{s}_{h-1}, a_{h-1})$
  equal to empirical conditional probabilities in D_{p}.
  for $\hat{s}' \in \widehat{\mathcal{S}}_h$ **do**
    Run Algorithm 3 with inputs $\hat{p}$ and $\hat{s}'$
  to obtain $(h-1)$-step policy $\psi_{\hat{s}'} : \widehat{\mathcal{S}}_{[h-1]} \to \mathcal{A}$.
    Set $\pi_{\hat{s}'}(x_\ell) = \psi_{\hat{s}'}(\hat{f}_\ell(x_\ell))$, $\ell \in [h-1]$, $x_\ell \in \mathcal{X}_\ell$.
  Let $\Pi_h = (\pi_{\hat{s}})_{\hat{s} \in \widehat{\mathcal{S}}_h}$.

Algorithm 2 Clustering to Find Latent-state Embeddings.

Input: Data points $\mathcal{Z} = \{\mathbf{z}_i\}_{i=1}^n$ and threshold $\tau > 0$.
Output: Cluster indices $\widehat{\mathcal{S}}$ and centers $\widehat{\boldsymbol{\phi}} : \widehat{\mathcal{S}} \to \mathcal{Z}$.
Let $\widehat{\mathcal{S}} = \emptyset$, $k = 0$ (number of clusters).
while $\mathcal{Z} \neq \emptyset$ **do**
  Pick any $\mathbf{z} \in \mathcal{Z}$ (a new cluster center).
  Let $\mathcal{Z}' = \{\mathbf{z}' \in \mathcal{Z} : \ \|\mathbf{z} - \mathbf{z}'\|_1 \leq \tau\}$.
  Add cluster: $k \leftarrow k+1$, $\widehat{\mathcal{S}} \leftarrow \widehat{\mathcal{S}} \cup \{k\}$, $\widehat{\boldsymbol{\phi}}(k) = \mathbf{z}$.
  Remove the newly covered points: $\mathcal{Z} \leftarrow \mathcal{Z} \setminus \mathcal{Z}'$.

We refer readers to

Algorithm 3 Dynamic Programming for Reaching a State

Input: target state $\hat{s}^* \in \widehat{\mathcal{S}}_h$,
    transition probabilities $\hat{p}(\hat{s}' \mid \hat{s}, a)$
      for all $\hat{s} \in \widehat{\mathcal{S}}_\ell$, $a \in \mathcal{A}$, $\hat{s}' \in \widehat{\mathcal{S}}_{\ell+1}$, $\ell \in [h-1]$.
Output: policy $\psi : \widehat{\mathcal{S}}_{[h-1]} \to \mathcal{A}$ maximizing $\hat{\mathbb{P}}^{\psi}(\hat{s}^*)$.
Let $v(\hat{s}^*) = 1$ and let $v(\hat{s}) = 0$ for all other $\hat{s} \in \widehat{\mathcal{S}}_h$.
for $\ell = h-1, h-2, \ldots, 1$ **do**
    for $\hat{s} \in \widehat{\mathcal{S}}_\ell$ **do**
        $\psi(\hat{s}) = \max_{a \in \mathcal{A}} \Big[\sum_{\hat{s}' \in \widehat{\mathcal{S}}_{\ell+1}} v(\hat{s}')\, \hat{p}(\hat{s}' \mid \hat{s}, a) \Big]$.
        $v(\hat{s}) = \sum_{\hat{s}' \in \mathcal{S}_{\ell+1}} v(\hat{s})\, \hat{p}(\hat{s}' \mid \hat{s}, a = \psi(\hat{s}))$.