Count-Based Exploration with the Successor Representation

Marlos C. Machado,[1] **Marc G. Bellemare,**[1] **Michael Bowling,**[2,3]
[1]Google AI, Brain Team, [2]University of Alberta, [3]DeepMind Alberta
{marlosm, bellemare, bowlingm}@google.com

Abstract

In this paper we introduce a simple approach for exploration in reinforcement learning (RL) that allows us to develop theoretically justified algorithms in the tabular case but that is also extendable to settings where function approximation is required. Our approach is based on the successor representation (SR), which was originally introduced as a representation defining state generalization by the similarity of successor states. Here we show that the norm of the SR, while it is being learned, can be used as a reward bonus to incentivize exploration. In order to better understand this transient behavior of the norm of the SR we introduce the substochastic successor representation (SSR) and we show that it implicitly counts the number of times each state (or feature) has been observed. We use this result to introduce an algorithm that performs as well as some theoretically sample-efficient approaches. Finally, we extend these ideas to a deep RL algorithm and show that it achieves state-of-the-art performance in Atari 2600 games when in a low sample-complexity regime.

1 Introduction

Reinforcement learning (RL) tackles sequential decision making problems by formulating them as tasks where an agent must learn how to act optimally through trial and error interactions with the environment. The goal in these problems is to maximize the (discounted) sum of the numerical reward signal observed at each time step. Because the actions taken by the agent influence not just the immediate reward but also the states and associated rewards in the future, sequential decision making problems require agents to deal with the trade-off between immediate and delayed rewards. Here we focus on the problem of exploration in RL, which aims to reduce the number of samples (i.e., interactions) an agent needs in order to learn to perform well in these tasks when the environment is initially unknown.

Surprisingly, the most common approach in the field is to select exploratory actions uniformly at random, with even high-profile success stories being obtained with this strategy (e.g., Tesauro 1995; Mnih et al. 2015). However, random exploration often fails in environments with sparse rewards, that is, environments where the agent observes a reward signal of value zero for the majority of states. In this paper we introduce an approach for exploration in RL based on the successor representation (SR; Dayan 1993). The SR is a representation that generalizes between states using the similarity between their successors, that is, the states that follow the current state given the agent's policy. The SR is defined for any problem, it can be learned with temporal-difference learning and, as we discuss below, it can be seen as implicitly estimating the transition dynamics of the environment.

The main contribution of this paper is to show that *the norm of the SR can be used as an exploration bonus*. We perform an extensive empirical evaluation to demonstrate this and we introduce the substochastic successor representation (SSR) to also understand, theoretically, the behavior of such a bonus. The SSR behaves similarly to the SR but it is more amenable to theoretical analyses. We show that the SSR implicitly counts state visitation, suggesting that the exploration bonus obtained from the SR, while it is being learned, might also be incorporating some notion of state visitation counts. We demonstrate this intuition empirically and we use this result to introduce algorithms that, in the tabular case, perform as well as traditional approaches with PAC-MDP guarantees. Finally, we extend the idea of using the norm of the SR as an exploration bonus to the function approximation case, designing a deep RL algorithm that achieves state-of-the-art performance in hard exploration Atari 2600 games when in a low sample-complexity regime. The proposed algorithm is also simpler than traditional baselines such as pseudo-count-based methods because it does not require domain-specific density models (Bellemare et al. 2016; Ostrovski et al. 2017).

2 Preliminaries

We consider an agent interacting with its environment in a sequential manner. Starting from a state $S_0 \in \mathcal{S}$, at each step the agent takes an action $A_t \in \mathcal{A}$, to which the environment responds with a state $S_{t+1} \in \mathcal{S}$ according to a transition function $p(s'|s,a) = \Pr(S_{t+1} = s'|S_t = s, A_t = a)$, and with a reward signal $R_{t+1} \in \mathbb{R}$, where $r(s,a)$ indicates the expected reward for a transition from state s under action a, that is, $r(s,a) \doteq \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$.

The value of a state s when following a policy π, $v_\pi(s)$, is defined to be the expected sum of discounted rewards from that state: $v_\pi(s) \doteq \mathbb{E}_\pi\Big[\sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \Big| S_t = s\Big]$, where γ is the discount factor. When the transition probability function p and the reward function r are known, we can compute $v_\pi(s)$ recursively by solving the system of equations below (Bellman 1957):

$$v_\pi(s) = \sum\nolimits_a \pi(a|s)\Big[r(s,a) + \gamma \sum\nolimits_{s'} p(s'|s,a) v_\pi(s')\Big].$$

These equations can also be written in matrix form with $\mathbf{v}_\pi, \mathbf{r} \in \mathbb{R}^{|\mathcal{S}|}$ and $P_\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$:

$$\mathbf{v}_\pi = \mathbf{r} + \gamma P_\pi \mathbf{v}_\pi = (I - \gamma P_\pi)^{-1}\mathbf{r}, \tag{1}$$

where P_π is the state to state transition probability function induced by π, that is, $P_\pi(s,s') = \sum_a \pi(a|s) p(s'|s,a)$.

Traditional model-based algorithms learn estimates of the matrix P_π and of the vector $\mathbf{r}$ and use them to estimate $\mathbf{v}_\pi$, for example by solving Equation 1. We use $\hat{P}_\pi$ and $\hat{\mathbf{r}}$ to denote empirical estimates of P_π and $\mathbf{r}$. Formally,

$$\hat{P}_\pi(s'|s) = \frac{n(s,s')}{n(s)}, \qquad \hat{\mathbf{r}}(s) = \frac{C(s,s')}{n(s)}, \tag{2}$$

where $\hat{\mathbf{r}}(i)$ denotes the i-th entry in the vector $\hat{\mathbf{r}}$, $n(s,s')$ is the number of times the transition $s \to s'$ was observed, $n(s) = \sum_{s' \in \mathcal{S}} n(s,s')$, and $C(s,s')$ is the sum of the rewards associated with the $n(s,s')$ transitions (we drop the action to simplify notation). However, model-based approaches are rarely successful in problems with large state spaces due to the difficulty in learning accurate models.

Because of the challenges in model learning, model-free solutions largely dominate the literature. In model-free RL, instead of estimating P_π and $\mathbf{r}$, we estimate $v_\pi(s)$ directly from samples. We often use TD learning (Sutton 1988) to update our estimates of $v_\pi(s)$, $\hat{v}(s)$, online:

$$\hat{v}(S_t) \leftarrow \hat{v}(S_t) + \alpha\big[R_{t+1} + \gamma \hat{v}(S_{t+1}) - \hat{v}(S_t)\big], \tag{3}$$

where α is the step-size parameter. Generalization is required in problems with large state spaces, where it is unfeasible to learn an individual value for each state. We do so by parametrizing $\hat{v}(s)$ with a set of weights θ. We write, given the weights θ, $\hat{v}(s;\theta) \approx v_\pi(s)$ and $\hat{q}(s,a;\theta) \approx q_\pi(s,a)$, where $q_\pi(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) v_\pi(s')$. Model-free methods have performed well in problems with large state spaces, mainly due to the use of neural networks as function approximators (e.g., Mnih et al. 2015).

The ideas presented here are based on the successor representation (SR; Dayan 1993). The successor representation with respect to a policy π, Ψ_π, is defined as

$$\Psi_\pi(s,s') = \mathbb{E}_{\pi,p}\Big[\sum_{t=0}^{\infty} \gamma^t \mathbb{I}\{S_t = s'\} \Big| S_0 = s\Big],$$

where we assume the sum is convergent with $\mathbb{I}$ denoting the indicator function. This expectation can actually be estimated from samples with TD learning:

$$\hat{\Psi}(S_t, j) \leftarrow \hat{\Psi}(S_t, j) + \eta\Big(\mathbb{I}\{S_t = j\} + \gamma \hat{\Psi}(S_{t+1}, j) - \hat{\Psi}(S_t, j)\Big), \tag{4}$$

for all $j \in \mathcal{S}$ and η denoting the step-size. The SR also corresponds to the Neumann series of γP_π:

$$\Psi_\pi = \sum_{t=0}^{\infty} (\gamma P_\pi)^t = (I - \gamma P_\pi)^{-1}. \tag{5}$$

Notice that the SR is part of the solution when computing a value function: $\mathbf{v}_\pi = \Psi_\pi \mathbf{r}$ (Equation 1). We use $\hat{\Psi}_\pi$ to denote the SR computed through $\hat{P}_\pi$, the approximation of P_π.

Successor features (Barreto et al. 2017) generalize the successor representation to the function approximation setting. We use the definition for the uncontrolled case.

Definition 2.1 (Successor Features)**.** *For a given $0 \leq \gamma < 1$, policy π, and for a feature representation $\boldsymbol{\phi}(s) \in \mathbb{R}^d$, the successor features for a state s are:*

$$\boldsymbol{\psi}_\pi(s) = \mathbb{E}_{\pi,p}\left[\sum_{t=0}^{\infty} \gamma^t \boldsymbol{\phi}(S_t) \middle| S_0 = s\right].$$

Alternatively, in matrix form, we can write the successor features as $\Psi_\pi = \sum_{t=0}^{\infty} (\gamma P_\pi)^t \Phi = (I - \gamma P_\pi)^{-1}\Phi$, where $\Phi \in \mathbb{R}^{|\mathcal{S}|\times d}$ is a matrix encoding the feature representation of each state such that $\boldsymbol{\phi}(s) \in \mathbb{R}^d$. This definition reduces to the SR in the tabular case, where $\Phi = I$.

3 $||\Psi(s)||$ as an Exploration Bonus

Recent results have shown that the SR naturally captures the diffusion properties of the environment (e.g., Machado et al. 2018b; Wu, Tucker, and Nachum 2019). Inspired by these results, in this section we argue that the SR can be explicitly used to promote exploration. We show that the norm of the SR, while it is being learned, behaves as an exploration bonus that rewards agents for visiting states it has visited less often. We first demonstrate this behavior empirically, in the tabular case. We then introduce the substochastic successor representation to provide some theoretical intuition that justifies this idea. In subsequent sections we show how these ideas carry over to the function approximation setting.

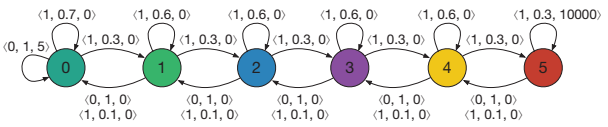
First Empirical Demonstration

To demonstrate the usefulness of the norm of the SR as an exploration bonus, we first compare the performance of traditional Sarsa (Rummery and Niranjan 1994; Sutton and Barto 1998) to Sarsa+SR, which incorporates the norm of the SR as an exploration bonus in the Sarsa update. The update equation for Sarsa+SR is

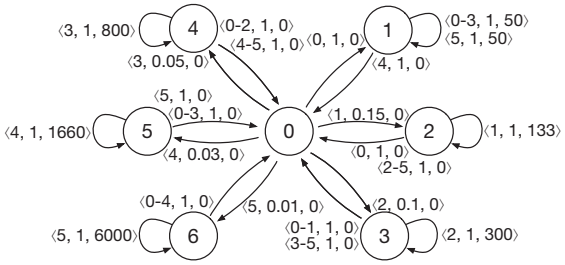
$$\hat{q}(S_t, A_t) \leftarrow \hat{q}(S_t, A_t) + \alpha\Big(R_t + \beta \frac{1}{||\hat{\Psi}(S_t)||_1} + \gamma \hat{q}(S_{t+1}, A_{t+1}) - \hat{q}(S_t, A_t)\Big), \tag{6}$$

where β is a scaling factor and, at each time step t, $\hat{\Psi}(S_t, \cdot)$ is updated before $\hat{q}(S_t, A_t)$ as per Equation 4.

We evaluated this algorithm in RiverSwim and SixArms (Strehl and Littman 2008), traditional domains in the PAC-MDP literature. In these domains, it is very likely that an agent will first observe a small reward generated in a



(a) RiverSwim



(b) SixArms

Figure 1: Domains used in the tabular case. The tuples in each transition denote ⟨action id, probability, reward⟩. In SIXARMS, the agent starts in state 0. In RIVERSWIM, the agent starts in either state 1 or 2 with equal probability.

state that is easy to get to. If the agent does not have a good exploration policy, it is likely to converge to a suboptimal behavior, never observing larger rewards available in states that are difficult to get to. See Figure 1 for more details.

We compared the performance of Sarsa and Sarsa+SR for 5,000 time steps when acting ϵ-greedily to maximize the discounted return ($\gamma = 0.95$). For Sarsa+SR, we swept over different values of α, η, γ_{SR}, β and ϵ, with $\alpha \in \{0.01, 0.05, 0.1, 0.25, 0.5\}$, $\eta \in \{0.01, 0.05, 0.1, 0.25, 0.5\}$, $\gamma_{SR} \in \{0.5, 0.8, 0.95, 0.99\}$, $\beta \in \{1, 10, 100, 1000, 10000\}$ and $\epsilon \in \{0.01, 0.05, 0.1\}$. For Sarsa, we swept over the parameters α and ϵ. For fairness, we looked at a finer granularity for these parameters, with $\alpha \in i \times 0.005$ for i ranging from 1 to 100, and with $\epsilon \in j \times 0.01$ for j ranging from 1 to 15. Table 6, at the end of the paper, summarizes the parameter settings that led to the best results for each algorithm in RIVERSWIM and SIXARMS. The performance of each algorithm, averaged over 100 runs, is available in Table 1.

Our results show that the proposed exploration bonus has a profound impact in the algorithm's performance. Sarsa obtains an average return of approximately 25,000 while Sarsa+SR obtains an approximate average return of 1.2 million. Notice that, in RIVERSWIM, the reward that is "easy to get" has value 5, implying that, different from Sarsa+SR, Sarsa almost never explores the state space well enough. We observe the same trend in SIXARMS.

Theoretical Justification

It is difficult to characterize the behavior of our proposed exploration bonus because it is updated at each time step with TD learning. It is hard to analyze the behavior of estimates obtained with TD learning in the interim. Also, at its fixed point, for a fixed policy, the ℓ_1-norm of the SR is $\sum \gamma^t 1 = 1/(1-\gamma)$ for all states, preventing us from using the fixed point of the SR to theoretically analyze the behavior of this exploration bonus. In this section we introduce the substochastic successor representation (SSR) to provide some theoretical intuition, in the prediction case, of why the norm of the SR is a good exploration bonus. The SSR behaves similarly to the SR but it is simpler to analyze.

Table 1: Comparison between Sarsa and Sarsa+SR. A 95% confidence interval is reported between parentheses.

	Sarsa	Sarsa + SR
RIVERSWIM	24,770 (196)	1,213,544 (540,454)
SIXARMS	247,977 (4,970)	1,052,934 (2,311,617)

Definition 3.1 (Substochastic Successor Representation)**.** *Let $\tilde{P}_\pi$ denote the substochastic matrix induced by the environment's dynamics and by the policy π such that $\tilde{P}_\pi(s'|s) = \frac{n(s,s')}{n(s)+1}$. For a given $0 \leq \gamma < 1$, the substochastic successor representation, $\tilde{\Psi}_\pi$, is defined as:*

$$\tilde{\Psi}_\pi = \sum_{t=0}^{\infty} \gamma^t \tilde{P}_\pi{}^t = (I - \gamma \tilde{P}_\pi)^{-1}.$$

The SSR only differs from the empirical SR in its incorporation of an additional "phantom" transition from each state, making it underestimate the real SR. Through algebraic manipulation we show that the SSR allows us to recover an estimate of the visit counts, $n(s)$. This result provides some intuition of why the exploration bonus we propose performs so well, as exploration bonuses based on state visitation counts are known to generate proper exploration.

As aforementioned, the SSR behaves similarly to the SR. When computing the norm of the SR, while it is being learned with TD learning, it is as if a reward of 1 was observed at each time step.[1] Thus, there is little variance in the target, with the predictions slowly approaching the true value of the SR. If pessimistically initialized, as traditionally done (i.e., initialized to zero when expecting positive rewards), the estimates of the SR approach the target from below. In this sense, the number of times a prediction has been updated in a given state is a good proxy to estimate how far this prediction is from its final target. From Definition 3.1 we can see that the SSR have similar properties. It underestimates the true target but slowly approaches it, converging to the true SR in the limit. The SSR simplifies the analysis by not taking bootstrapping into consideration.

The theorem below formalizes the idea that the norm of the SSR implicitly counts state visitation, shedding some light on the efficacy of the exploration bonus we propose.

Theorem 1. *Let $n(s)$ denote the number of times state s has been visited and let $\tilde{\Psi}_\pi$ denote the substochastic successor representation as in Definition 3.1. For a given $0 \leq \gamma < 1$,*

$$\frac{\gamma}{n(s)+1} - \frac{\gamma^2}{1-\gamma} \leq (1+\gamma) - ||\tilde{\Psi}_\pi(s)||_1 \leq \frac{\gamma}{n(s)+1}.$$

[1] In vector form, when estimating the SR with TD learning, the clause $\mathbb{I}\{S_t = j\}$, from Equation 4, is always true for one of the states, that is, an entry in the vector representing the SR. Thus, it is as if a reward of 1 was observed at each time step.

Proof of Theorem 1. Let $\hat{P}_\pi$ be the empirical transition matrix. We first rewrite $\tilde{P}_\pi$ in terms of $\hat{P}_\pi$:

$$\tilde{P}_\pi(s,s') = \frac{n(s,s')}{n(s)+1} = \frac{n(s)}{n(s)+1}\frac{n(s,s')}{n(s)}$$
$$= \frac{n(s)}{n(s)+1}\hat{P}_\pi(s,s') = \Big(1 - \frac{1}{n(s)+1}\Big)\hat{P}_\pi(s,s')$$

This expression can also be written in matrix form: $\tilde{P}_\pi = (I - N)\hat{P}_\pi$, where $N \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ denotes the diagonal matrix of augmented inverse counts. Expanding $\tilde{\Psi}_\pi$ we have:

$$\tilde{\Psi}_\pi \;=\; \sum_{t=0}^{\gamma}(\gamma\tilde{P}_\pi)^t \;=\; I + \gamma\tilde{P}_\pi + \gamma^2\tilde{P}_\pi{}^2\tilde{\Psi}_\pi.$$

The top eigenvector of a stochastic matrix is the all-ones vector, $\mathbf{1}$ (Meyn and Tweedie 2012). Using this fact and the definition of $\tilde{P}_\pi$ with respect to $\hat{P}_\pi$ we have:

$$\tilde{\Psi}_\pi\mathbf{1} \;=\; \big(I + \gamma(I-N)\hat{P}_\pi\big)\mathbf{1} + \gamma^2\tilde{P}_\pi{}^2\tilde{\Psi}_\pi\mathbf{1}$$
$$=\; (I+\gamma)\mathbf{1} - \gamma N\mathbf{1} + \gamma^2\tilde{P}_\pi{}^2\tilde{\Psi}_\pi\mathbf{1}. \qquad (7)$$

We can now bound the term $\gamma^2\tilde{P}_\pi{}^2\tilde{\Psi}_\pi\mathbf{1}$ using the fact that $\mathbf{1}$ is also the top eigenvector of the successor representation and has eigenvalue $\frac{1}{1-\gamma}$ (Machado et al. 2018b):

$$0 \le \gamma^2\tilde{P}_\pi{}^2\tilde{\Psi}_\pi\mathbf{1} \le \frac{\gamma^2}{1-\gamma}\mathbf{1}.$$

Plugging (7) into the definition of the SR we have (notice that $\Psi(s)\mathbf{1} = ||\Psi(s)||_1$):

$$(1+\gamma)\mathbf{1} - \tilde{\Psi}_\pi\mathbf{1} = \gamma N\mathbf{1} - \gamma^2\tilde{P}_\pi{}^2\tilde{\Psi}_\pi\mathbf{1} \;\le\; \gamma N\mathbf{1}.$$

When we also use the other bound on the quadratic term we conclude that, for any state s,

$$\frac{\gamma}{n(s)+1} - \frac{\gamma^2}{1-\gamma} \le (1+\gamma) - ||\tilde{\Psi}_\pi(s)||_1 \le \frac{\gamma}{n(s)+1}. \;\square$$

The relationship between the SSR and the SR. Theorem 1 shows that the SSR, obtained after a slight change to the SR, can be used to recover state visitation counts. The intuition behind this result is that the phantom transition, represented by the $+1$ in the denominator of the SSR, serves as a proxy for the uncertainty about that state by underestimating the SR. This is due to the fact that $\sum_{s'}\tilde{P}_\pi(s,s')$ gets closer to 1 each time state s is visited.

This result also suggests that the exploration bonus we propose, $||\Psi(\cdot)||^{-1}$, behaves as a proper exploration bonus. It rewards the agent for visiting a state for the first times and it becomes less effective as the agent continues to visit that state. Hence, our approach does not change the optimal policy because, in the limit, the norm of the SR (and the exploration bonus) converges to a constant value (see footnote 1):

$$\lim_{t\to\infty} ||\Psi(s)||_1 = \frac{1}{1-\gamma}. \qquad (8)$$

To validate the intuition that the SSR behaves similarly to the SR, we empirically evaluated the norm of the SR in the six states of RiverSwim as a function of the number of times those states were visited. Figure 2 depicts the average result over the 100 runs used to generate the results in Table 1. The plot shows that the norm of the SR does indeed grow as a function of state visitation counts. In this particular case, the norm of the SR grows at a rate between $1/n$, as suggested by the SSR, and $1/\sqrt{n}$, which is the rate at which TD often converges (Theorem 3.6; Dalal et al. 2018).

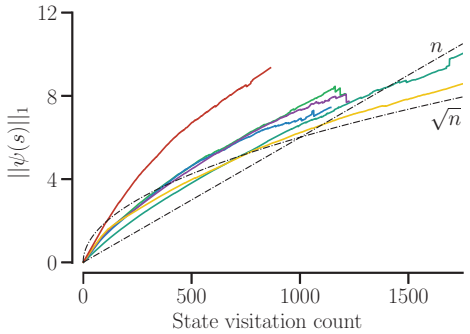


Figure 2: Empirical evaluation of the the norm of the SR as a function of state visitation count. Each curve denotes the evolution in one of the six states of RIVERSWIM. The colors match the colors of the states in Figure 1a. Reference functions $f_1(n) = c_1\sqrt{n}$ and $f_2(n) = c_2 n$ are depicted for comparison ($c_1 = 0.19$; $c_2 = 0.006$). See text for details.

Empirical validation of $||\tilde{\Psi}||_1$ as an exploration bonus. As a sanity check, we used the result in Theorem 1 to implement a simple model-based algorithm that penalizes the agent for visiting commonly visited states with the exploration bonus $\mathbf{r}_{\text{int}} = -||\tilde{\Psi}_\pi(s)||_1$. Our agent maximizes $r(s,a) + \beta r_{\text{int}}(s)$, where β is a scaling parameter. The shift $(1+\gamma)$ in the theorem has no effect in the agent's policy because it is the same across all states. In this algorithm the agent updates its transition model and reward model with Equation 2 and its SSR estimate as in Definition 3.1.

Table 2 depicts the performance of this algorithm, dubbed ESSR, as well as the performance of some algorithms with polynomial sample-complexity bounds. The goal with this evaluation is not to outperform these algorithms, but to evaluate how well ESSR performs when compared to algorithms that explicitly keep visitation counts to promote exploration. ESSR performs as well as R-MAX (Brafman and Tennenholtz 2002) and E^3 (Kearns and Singh 2002) on RIVERSWIM and it outperforms these algorithms on SIXARMS; while MBIE (Strehl and Littman 2008), which explicitly estimates confidence intervals over the expected return in each state, outperforms ESSR in these domains. These results clearly show that ESSR performs, on average, similarly to other algorithms with PAC-MDP guarantees, suggesting that the norm of the SSR is a promising exploration bonus.[2]

[2]The code used to generate all results in this section is available at: https://github.com/mcmachado/count_based_exploration_sr/tree/master/tabular.

Table 2: Comparison between ESSR, R-MAX, E^3, and MBIE. The numbers reported for R-MAX, E^3, and MBIE were extracted from the histograms presented by Strehl and Littman (2008). ESSR's performance is the average over 100 runs. A 95% confidence interval is reported between parentheses. All numbers are reported in millions (i.e., $\times 10^6$).

	E^3	R-MAX	MBIE	ESSR	
RIVERSWIM	3.0	3.0	3.3	3.1	(0.06)
SIXARMS	1.8	2.8	9.3	7.3	(1.2)

4 Counting Feature Activations with the SR

In large environments, where enumerating all states is not an option, directly using Sarsa+SR as described in the previous section is not viable. However, one of the reasons the results in the previous section are interesting is the fact that there is a natural extension of the SR to non-tabular settings, the successor features (Definition 2.1), and the fact that we can immediately use norms in the function approximation setting. In this section we show how one can extend the idea of using the norm of the SR as an exploration bonus to the function approximation setting, something one cannot easily do if relying on explicit state visitation counts. Because deep RL approaches often lead to state-of-the-art performance while also learning a representation from high-dimensional sensory inputs, in this section we introduce a deep RL algorithm that incorporates the ideas introduced. Our algorithm was also inspired by recent work that has shown that successor features can be learned jointly with the feature representation itself (Kulkarni et al. 2016; Machado et al. 2018b).

An overview of the neural network we used to learn the agent's value function while also learning the feature representation and the SR is depicted in Figure 3. The layers used to compute the state-action value function, $\hat{q}(S_t, \cdot)$, are structured as in DQN (Mnih et al. 2015), with the number of parameters (i.e., filter sizes, stride, and number of nodes) matching Oh et al.'s (2015) architecture, which is known to succeed in the auxiliary task detailed below of predicting the agent's next observation. We call the part of our architecture that predicts $\hat{q}(S_t, \cdot)$ $\text{DQN}_e^{\text{MMC}}$. It is trained to minimize

$$\mathcal{L}_{\text{TD}} = \mathbb{E}\Big[\big((1-\tau)\delta(s,a) + \tau\delta_{\text{MC}}(s,a)\big)^2\Big],$$

with $\delta(s,a)$ and $\delta_{\text{MC}}(s,a)$ being defined as

$$\delta(s,a) = R_t + \beta r_{\text{int}}(s;\theta^-) + \gamma \max_{a'} q(s',a';\theta^-) - q(s,a;\theta),$$

$$\delta_{\text{MC}}(s,a) = \sum_{t=0}^{\infty} \gamma^t \Big(r(S_t,A_t) + \beta r_{\text{int}}(S_t;\theta^-)\Big) - q(s,a;\theta).$$

This loss is known as the mixed Monte-Carlo return (MMC) and it has been used in the past by the algorithms that achieved succesful exploration in deep reinforcement learning (Bellemare et al. 2016; Ostrovski et al. 2017). The distinction between θ and θ^- is standard in the field, with θ^- denoting the parameters of the target network, which is updated less often for stability purposes (Mnih et al. 2015). As before, we use r_{int} to denote the exploration bonus obtained from the successor features of the internal representation, ϕ, which will be defined below. Moreover, to ensure all features are in the same range, we normalize the feature vector so that $||\phi(\cdot)||_1 = 1$. In Figure 3 we highlight with ϕ the layer in which we normalize its output. Notice that the features are always non-negative due to the use of ReLU gates.

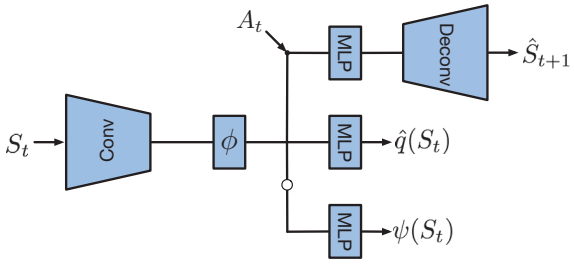


Figure 3: Neural network architecture used by our algorithm when learning to play Atari 2600 games.

The successor features, $\psi(S_t)$, at the bottom of the diagram, are obtained by minimizing the loss

$$\mathcal{L}_{\text{SR}} = \mathbb{E}\Big[\big(\phi(S_t;\theta^-) + \gamma\psi(S_{t+1};\theta^-) - \psi(S_t;\theta)\big)^2\Big].$$

Zero is a fixed point for the SR, which is particularly concerning in settings with sparse rewards. The agent might end up learning to set $\phi(\cdot) = \vec{0}$ to achieve zero loss. We address this problem by not propagating $\nabla\mathcal{L}_{\text{SR}}$ to ϕ (this is depicted in Figure 3 as an open circle stopping the gradient); and by creating an auxiliary task (Jaderberg et al. 2017) to encourage a representation to be learned before a non-zero reward is observed. As Machado et al. (2018b), we use the auxiliary task of predicting the next observation, learned with the architecture proposed by Oh et al. (2015), which is depicted as the top layers in Figure 3. The loss we minimize for this last part of the network is

$$\mathcal{L}_{\text{Recons}} = \big(\hat{S}_{t+1} - S_{t+1}\big)^2.$$

The overall loss minimized by the network is

$$\mathcal{L} = w_{\text{TD}}\mathcal{L}_{\text{TD}} + w_{\text{SR}}\mathcal{L}_{\text{SR}} + w_{\text{Recons}}\mathcal{L}_{\text{Recons}}.$$

The last step in describing our algorithm is to define $r_{\text{int}}(S_t;\theta^-)$, the intrinsic reward we use to encourage exploration. As in Sarsa+SR, we choose the exploration bonus to be the inverse of the ℓ_1-norm of the vector of successor features of the current state. That is,

$$r_{\text{int}}(S_t;\theta^-) = \frac{1}{||\psi(S_t;\theta^-)||_1},$$

where $\psi(S_t;\theta^-)$ denotes the successor features of state S_t parametrized by θ^-. The exploration bonus comes from the same intuition presented in the previous section (we observed in preliminary experiments not discussed here that DQN performs better when dealing with positive rewards).

Table 3: Performance of the proposed algorithm, $\text{DQN}_e^{\text{MMC}}$+SR, compared to various agents on the "hard exploration" subset of Atari 2600 games. The DQN results reported are from Machado et al. (2018a) while the $\text{DQN}_{\text{CTS}}^{\text{MMC}}$, $\text{DQN}_{\text{PIXELCNN}}^{\text{MMC}}$ and RND results were obtained through personal communication with the authors of the corresponding papers. Burda et al. did not evaluate RND in FREEWAY. When available, standard deviation is reported between parentheses. See text for details.

	DQN		$\text{DQN}_e^{\text{MMC}}$		$\text{DQN}_{\text{CTS}}^{\text{MMC}}$	$\text{DQN}_{\text{PIXELCNN}}^{\text{MMC}}$	RND		$\text{DQN}_e^{\text{MMC}}$+SR	
FREEWAY	32.4	(0.3)	29.5	(0.1)	29.2	29.4	-	-	29.4	(0.1)
GRAVITAR	118.5	(22.0)	1078.3	(254.1)	199.8	275.4	790.0	(122.9)	457.4	(120.3)
MONT. REV.	0.0	(0.0)	0.0	(0.0)	2941.9	1671.7	524.8	(314.0)	1395.4	(1121.8)
PRIVATE EYE	1447.4	(2,567.9)	113.4	(42.3)	32.8	14386.0	61.3	(53.7)	104.4	(50.4)
SOLARIS	783.4	(55.3)	2244.6	(378.8)	1147.1	2279.4	1270.3	(291.0)	1890.1	(163.1)
VENTURE	4.4	(5.4)	1220.1	(51.0)	0.0	856.2	953.7	(167.3)	1348.5	(56.5)

A complete description of the network architecture is available in Figure 4, which is at the end of the paper to allow the reader to first focus on the main concepts of the proposed idea. We initialize our network the same way Oh et al. (2015) does. We use Xavier initialization (Glorot and Bengio 2010) in all layers except the fully connected layers around the element-wise multiplication denoted by $\otimes$, which are initialized uniformly with values between -0.1 and 0.1.

5 Evaluation of Exploration in Deep RL

We evaluated our algorithm on the Arcade Learning Environment (Bellemare et al. 2013). Following Bellemare et al.'s (2016) taxonomy, we focused on the Atari 2600 games with sparse rewards that pose hard exploration problems. They are: FREEWAY, GRAVITAR, MONTEZUMA'S REVENGE, PRIVATE EYE, SOLARIS, and VENTURE.[3]

We used the evaluation protocol proposed by Machado et al. (2018a). The reported results are the average over 10 seeds after 100 million frames. We evaluated our agents in the stochastic setting (sticky actions, $\varsigma = 0.25$) using a frame skip of 5 with the full action set ($|\mathcal{A}| = 18$). The agent learns from raw pixels i.e., it uses the game screen as input.

Our results were obtained with the algorithm described in Section 4. We set $\beta = 0.05$ after a rough sweep over values in the game MONTEZUMA'S REVENGE. We annealed ϵ in DQN's ϵ-greedy exploration over the first million steps, starting at 1.0 and stopping at 0.1 as done by Bellemare et al. (2016). We trained the network with RMSprop with a step-size of 0.00025, an ϵ value of 0.01, and a decay of 0.95, which are the standard parameters for training DQN (Mnih et al. 2015). The discount factor, γ, is set to 0.99, and $w_{\text{TD}} = 1$, $w_{\text{SR}} = 1000$, $w_{\text{Recons}} = 0.001$. The weights w_{TD}, w_{SR}, and w_{Recons} were set so that the loss functions would be roughly at the same scale. All other parameters are the same as those used by Mnih et al. (2015) and Oh et al. (2015).

Overall Performance and Baselines

Table 3 summarizes the results after 100 million frames. The performance of other algorithms is also provided for reference. Notice we are reporting learning performance for all algorithms instead of the maximum scores achieved by the algorithm. We use the superscript $^{\text{MMC}}$ to distinguish between the algorithms that use MMC from those that do not. When comparing our algorithm, $\text{DQN}_e^{\text{MMC}}$+SR, to DQN we can see how much our approach improves over the most traditional baseline. By comparing our algorithm's performance to $\text{DQN}_{\text{CTS}}^{\text{MMC}}$ (Bellemare et al. 2016) and $\text{DQN}_{\text{PIXELCNN}}^{\text{MMC}}$ (Ostrovski et al. 2017) we compare our algorithm to established baselines for exploration that are closer to our method. By comparing our algorithm's performance to Random Network Distillation (RND; Burda et al. 2019) we compare our algorithm to the most recent paper in the field with state-of-the-art performance. Finally, we also evaluate the impact of the proposed exploration bonus by comparing our algorithm to $\text{DQN}_e^{\text{MMC}}$, which uses the same number of parameters our network uses (i.e., filter sizes, stride, and number of nodes), but without the additional modules (next state prediction and successor representation) and without the intrinsic reward bonus. We do this by setting $w_{\text{SR}} = w_{\text{Recons}} = \beta = 0$.

We can clearly see that our algorithm achieves scores much higher than those achieved by DQN, which struggles in games that pose hard exploration problems. When comparing our algorithm to $\text{DQN}_{\text{CTS}}^{\text{MMC}}$ and $\text{DQN}_{\text{PIXELCNN}}^{\text{MMC}}$ we observe that, on average, $\text{DQN}_e^{\text{MMC}}$+SR at least matches the performance of these algorithms while being simpler by not requiring a density model. Instead, our algorithm requires the SR, which is domain-independent as it is already defined for every problem since it is a component of the value function estimates, as discussed in Section 2. Finally, $\text{DQN}_e^{\text{MMC}}$+SR also outperforms RND (Burda et al. 2019) when it is trained for 100 million frames.[4] Importantly, RND, when trained for 2 billion frames, is currently considered to be the state-of-the-art approach for exploration in Atari 2600 games. Recently Taiga et al. (2019) evaluated several exploration algorithms, including those we use as baselines, and they have shown that, in these games, their performance at 100 million frames is predictive of their performance at one billion frames.

Finally, the comparison between $\text{DQN}_e^{\text{MMC}}$+SR and $\text{DQN}_e^{\text{MMC}}$ shows that the provided exploration bonus has a big impact in the game MONTEZUMA'S REVENGE, which is

[3] The code used to generate the reported results is available at: https://github.com/mcmachado/count_based_exploration_sr/tree/master/function_approximation.

[4] $\text{DQN}_e^{\text{MMC}}$+SR outperforms $\text{DQN}_{\text{CTS}}^{\text{MMC}}$ in five out of six games, it outperforms RND in four out of five games, and its performance is comparable to $\text{DQN}_{\text{PIXELCNN}}^{\text{MMC}}$'s performance.

probably known as the hardest game among those we used in our evaluation, and the only game where agents do not learn how to achieve scores greater than zero with random exploration. Interestingly, the change in architecture and the use of MMC leads to a big improvement in games such as GRAVITAR and VENTURE, which we cannot fully explain. However, because the change in architecture does not have any effect in MONTEZUMA'S REVENGE, it seems that the proposed exploration bonus is essential in games with very sparse rewards.

Evaluating the Impact of the Auxiliary Task

While the results depicted in Table 3 allow us to see the benefit of using an exploration bonus derived from the SR, they do not inform us about the impact of the auxiliary task in the results. The experiments in this section aim at addressing this issue. We focus on MONTEZUMA'S REVENGE because it is the game where the problem of exploration is maximized, with most algorithms not being able to do anything without an exploration bonus.

The first question we asked was whether the *auxiliary task was necessary* in our algorithm. We evaluated this by dropping the reconstruction module from the network to test whether the initial random noise generated by the SR is enough to drive representation learning. It is not. When dropping the auxiliary task, the average performance of this baseline over 4 seeds in MONT. REVENGE after 100 million frames was 100 points (σ^2 = 200; min: 0, max: 400). As comparison, our algorithm obtains 1395.4 points (σ^2 = 1121.8, min: 400, max: 2500). These results suggest that auxiliary tasks are necessary for our method to perform well.

We also evaluated whether the *auxiliary task was sufficient* to generate the results we observed. To do so we dropped the SR module and set $\beta = 0.0$ to evaluate whether our exploration bonus was actually improving the agent's performance or whether the auxiliary task was doing it. The exploration bonus seems to be essential. When dropping the exploration bonus and the SR module, the average performance of this baseline over 4 seeds in MONTEZUMA'S REVENGE after 100 million frames was 398.5 points (σ^2 = 230.1; min: 0, max: 400). Again, clearly, the auxiliary task is not a sufficient condition for the performance we report. The reported results use the same parameters as before.

Thus, while it is hard to completely disentangle the impact of the different components of DQN$_e^{\text{MMC}}$+SR (e.g., the exploration bonus, learning the SR, the auxiliary task), the comparisons in Table 3 and the results in this section suggest that the exploration bonus we introduced is essential for our approach to achieve state-of-the-art performance.

Evaluating the Impact of Using Different P-Norms

To further understand the different nuances behind the idea that the norm of the successor representation can be used to generate an exploration bonus, we also asked the question of whether this is true only for the ℓ_1-norm of the SR. It is often said, for example, that the ℓ_2-norm is smoother, and thus might be more amenable to the training of neural networks.

For the function approximation case, DQN$_e^{\text{MMC}}$+SR when using the ℓ_2-norm has a performance comparable to DQN$_e^{\text{MMC}}$+SR when using the ℓ_1-norm of the SR to generate its exploration bonus (ϕ is normalized with the respective norm). The actual performance of both approaches in the Atari 2600 games we used in the previous experiment is available in Table 4. We followed the same evaluation protocol described before, averaging the performance of DQN$_e^{\text{MMC}}$+SR with the ℓ_2-norm over 10 runs. The parameter β is the only parameter not shared by both algorithms. While $\beta = 0.025$ when using the ℓ_2-norm of the SR, $\beta = 0.05$ when using the ℓ_1-norm of the SR.

Table 4: Performance of the proposed algorithm, DQN$_e^{\text{MMC}}$+SR, when using the ℓ_1-norm and ℓ_2-norm of the SR to generate the exploration bonus. Standard deviation is reported between parentheses. See text for details.

	ℓ_1-norm		ℓ_2-norm	
FREEWAY	29.4	(0.1)	29.5	(0.1)
GRAVITAR	457.4	(120.3)	430.3	(109.4)
MONT. REV.	1395.4	(1121.8)	1778.6	(903.6)
PRIVATE EYE	104.4	(50.4)	99.1	(1.8)
SOLARIS	1890.1	(163.1)	2155.7	(398.3)
VENTURE	1348.5	(56.5)	1241.8	(236.0)

Table 5: Performance of Sarsa+SR when using the ℓ_1-norm and ℓ_2-norm of the SR to generate the exploration bonus. A 95% confidence interval is reported between parentheses.

	ℓ_1-norm	ℓ_2-norm
RIVERSWIM	1,213,544 (540,454)	1,192,052 (507,179)
SIXARMS	1,052,934 (2,311,617)	819,927 (2,132,003)

We also revisited the results presented in Section 3 to evaluate, in the tabular case, the impact of the different norms in Sarsa+SR. We swept over all the parameters, as previously described. The results reported for Sarsa+SR when using the ℓ_2-norm of the SR are the average over 100 runs. The actual numbers are available in Table 5. As before, it seems that it does not make much difference which norm of the SR we use (ℓ_1-norm or the ℓ_2-norm). The fact that these results are so close might suggest that the idea of using the norm of the SR for exploration is quite general, with the p-norm of the SR being effective for more than one value of p. Recall that $||x||_1 \leq \sqrt{2}||x||_2$ for any finite vector x.

6 Related Work

There are multiple algorithms in the tabular, model-based case, with guarantees about their performance in terms of regret bounds (e.g., Osband, Roy, and Wen 2016) or sample-complexity (e.g., Brafman and Tennenholtz 2002; Kearns and Singh 2002; Strehl and Littman 2008). RIVERSWIM and SIXARMS are domains traditionally used when evaluating these algorithms. In this paper we introduced a model-free algorithm that performs particularly well in these domains. We also introduced a model-based algorithm that performs as well as some of these algorithms with theoretical guarantees. Among these algorithms, R-MAX is the closest approach to ours. As R-MAX, the algorithm presented in Section 3 augments the state-space with an imaginary state and

Table 6: Parameter settings that led to the reported performance in RIVERSWIM and SIXARMS.

	RIVERSWIM					SIXARMS				
Algorithm	α	η	γ_{SR}	β	ϵ	α	η	γ_{SR}	β	ϵ
Sarsa	0.005	-	-	-	0.01	0.465	-	-	-	0.03
Sarsa+SR (w/ ℓ_1-norm)	0.25	0.01	0.95	100	0.1	0.1	0.01	0.99	100	0.01
Sarsa+SR (w/ ℓ_2-norm)	0.25	0.01	0.99	100	0.1	0.1	0.01	0.99	10	0.01

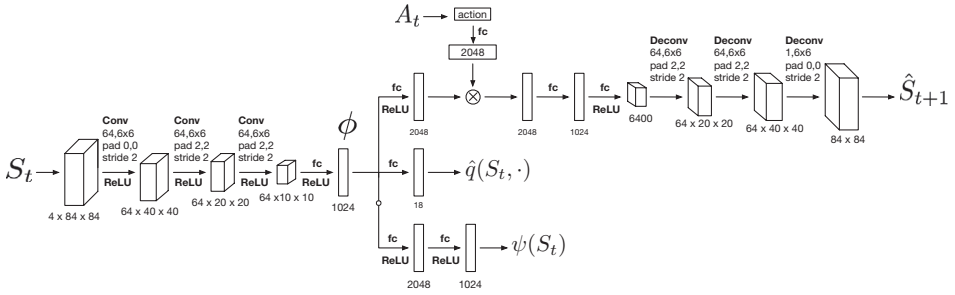


Figure 4: Neural network architecture used by our algorithm when learning to play Atari 2600 games.

encourages the agent to visit that state, implicitly reducing the algorithm's uncertainty. However, R-MAX deletes the transition to this imaginary state once a state has been visited a given number of times. Ours, on the other hand, lets the probability of visiting this imaginary state vanish with additional visitations. Importantly, notice that it is not clear how to apply traditional algorithms such as R-MAX and E^3 to large domains where function approximation is required.

Conversely, there are not many model-free approaches with proven sample-complexity bounds (e.g., Strehl et al. 2006), but there are multiple model-free algorithms for exploration that actually work in large domains (e.g., Bellemare et al. 2016; Ostrovski et al. 2017; Plappert et al. 2018; Burda et al. 2019). Among these algorithms, the use of pseudo-counts through density models is the closest to ours (Bellemare et al. 2016; Ostrovski et al. 2017). Inspired by those papers we used the mixed Monte-Carlo return as a target in the update rule. In Section 5 we showed that our algorithm at least matches these approaches while being simpler by not requiring a density model. Importantly, Martin et al. (2017) had already shown that counting activations of fixed, handcrafted features in Atari 2600 games leads to good exploration behavior. Nevertheless, by using the SR we are not only counting *learned* features but we are also implicitly capturing the induced transition dynamics.

7 Conclusion

RL algorithms tend to have high sample complexity, which often prevents them from being used in the real-world. Poor exploration strategies is one of the reasons for this high sample-complexity. Despite all of its shortcomings, uniform random exploration is, to date, the most commonly used approach for exploration. This is mainly due to the fact that most approaches for tackling the exploration problem still rely on domain-specific knowledge (e.g., density models, handcrafted features), or on having an agent learn a perfect model of the environment. In this paper we introduced a general method for exploration in RL that implicitly counts state (or feature) visitation in order to guide the exploration process. It is compatible with representation learning and the idea can also be adapted to be applied to large domains.

This result opens up multiple possibilities for future work. Based on the results presented in Section 3, for example, we conjecture that the substochastic successor representation can be actually used to generate algorithms with PAC-MDP bounds (e.g., one could replace explicit state visitation counts by the norm of the SSR in traditional algorithms). Investigating to what extent different auxiliary tasks impact the algorithm's performance, and whether simpler tasks such as predicting feature activations or parts of the input (Jaderberg et al. 2017) are effective is also worth studying. Finally, it might be interesting to further investigate the connection between representation learning and exploration, since it is also known that better representations can lead to faster exploration (Jiang et al. 2017).

Acknowledgements

The authors would like to thank Jesse Farebrother for the initial implementation of DQN used in this paper, Georg Ostrovski for the discussions and for providing us the exact results we report for $\text{DQN}^{\text{MMC}}_{\text{CTS}}$ and $\text{DQN}^{\text{MMC}}_{\text{PIXELCNN}}$, and Yuri Burda for providing us the data we used to compute the performance we report for RND in Atari 2600 games. We would also like to thank Carles Gelada, George Tucker and Or Sheffet for useful discussions, as well as the anonymous reviewers for their feedback. This work was supported by grants from Alberta Innovates Technology Futures and the Alberta Machine Intelligence Institute (Amii). Computing resources were provided by Compute Canada through CalculQuébec. Marlos C. Machado performed this work while at the University of Alberta.

References

Barreto, A.; Dabney, W.; Munos, R.; Hunt, J.; Schaul, T.; Silver, D.; and van Hasselt, H. 2017. Successor Features for Transfer in Reinforcement Learning. In *Advances in Neural Information Processing Systems (NIPS)*, 4058–4068.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research* 47:253–279.

Bellemare, M. G.; Srinivasan, S.; Ostrovski, G.; Schaul, T.; Saxton, D.; and Munos, R. 2016. Unifying Count-Based Exploration and Intrinsic Motivation. In *Advances in Neural Information Processing Systems (NIPS)*, 1471–1479.

Bellman, R. E. 1957. *Dynamic Programming*. Princeton, NJ: Princeton University Press.

Brafman, R. I., and Tennenholtz, M. 2002. R-MAX - A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. *Journal of Machine Learning Research* 3:213–231.

Burda, Y.; Edwards, H.; Storkey, A.; and Klimov, O. 2019. Exploration by Random Network Distillation. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Dalal, G.; Szörényi, B.; Thoppe, G.; and Mannor, S. 2018. Finite Sample Analyses for TD(0) With Function Approximation. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, 6144–6160.

Dayan, P. 1993. Improving Generalization for Temporal Difference Learning: The Successor Representation. *Neural Computation* 5(4):613–624.

Glorot, X., and Bengio, Y. 2010. Understanding the Difficulty of Training Deep Feedforward Neural Networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 249–256.

Jaderberg, M.; Mnih, V.; Czarnecki, W. M.; Schaul, T.; Leibo, J. Z.; Silver, D.; and Kavukcuoglu, K. 2017. Reinforcement Learning with Unsupervised Auxiliary Tasks. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Jiang, N.; Krishnamurthy, A.; Agarwal, A.; Langford, J.; and Schapire, R. E. 2017. Contextual Decision Processes with Low Bellman Rank are PAC-Learnable. In *Proceedings of the International Conference on Machine Learning (ICML)*, 1704–1713.

Kearns, M. J., and Singh, S. P. 2002. Near-Optimal Reinforcement Learning in Polynomial Time. *Machine Learning* 49(2-3):209–232.

Kulkarni, T. D.; Saeedi, A.; Gautam, S.; and Gershman, S. J. 2016. Deep Successor Reinforcement Learning. *CoRR* abs/1606.02396.

Machado, M. C.; Bellemare, M. G.; Talvitie, E.; Veness, J.; Hausknecht, M.; and Bowling, M. 2018a. Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents. *Journal of Artificial Intelligence Research* 61:523–562.

Machado, M. C.; Rosenbaum, C.; Guo, X.; Liu, M.; Tesauro, G.; and Campbell, M. 2018b. Eigenoption Discovery through the Deep Successor Representation. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Martin, J.; Sasikumar, S. N.; Everitt, T.; and Hutter, M. 2017. Count-Based Exploration in Feature Space for Reinforcement Learning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2471–2478.

Meyn, S. P., and Tweedie, R. L. 2012. *Markov Chains and Stochastic Stability*. Springer-Verlag.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level Control through Deep Reinforcement Learning. *Nature* 518:529–533.

Oh, J.; Guo, X.; Lee, H.; Lewis, R. L.; and Singh, S. P. 2015. Action-Conditional Video Prediction using Deep Networks in Atari Games. In *Advances in Neural Information Processing Systems (NIPS)*, 2863–2871.

Osband, I.; Roy, B. V.; and Wen, Z. 2016. Generalization and Exploration via Randomized Value Functions. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2377–2386.

Ostrovski, G.; Bellemare, M. G.; van den Oord, A.; and Munos, R. 2017. Count-Based Exploration with Neural Density Models. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2721–2730.

Plappert, M.; Houthooft, R.; Dhariwal, P.; Sidor, S.; Chen, R. Y.; Chen, X.; Asfour, T.; Abbeel, P.; and Andrychowicz, M. 2018. Parameter Space Noise for Exploration. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Rummery, G. A., and Niranjan, M. 1994. On-line Q-Learning using Connectionist Systems. CUED/F-INFENG/TR 166, Cambridge University Engineering Dept.

Strehl, A. L., and Littman, M. L. 2008. An Analysis of Model-based Interval Estimation for Markov Decision Processes. *Journal of Computer and System Sciences* 74(8):1309–1331.

Strehl, A. L.; Li, L.; Wiewiora, E.; Langford, J.; and Littman, M. L. 2006. PAC Model-Free Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 881–888.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT Press.

Sutton, R. S. 1988. Learning to Predict by the Methods of Temporal Differences. *Machine Learning* 3:9–44.

Taiga, A. A.; Fedus, W.; Machado, M. C.; Courville, A.; and Bellemare, M. G. 2019. Benchmarking Bonus-Based Exploration Methods on the Arcade Learning Environment. *CoRR* abs/1908.02388.

Tesauro, G. 1995. Temporal Difference Learning and TD-Gammon. *Communications of the ACM* 38(3):58–68.

Wu, Y.; Tucker, G.; and Nachum, O. 2019. The Laplacian in RL: Learning Representations with Efficient Approximations. In *Proceedings of the International Conference on Learning Representations (ICLR)*.