Theory and Application of Bonus-based Exploration in Reinforcement Learning

Bryan Chen

Acknowledgement

I would like to thank my advisor Jiantao Jiao and collaborator Paria Rashidinejad for their advice and feedback throughout this project.

Theory and Application of Bonus-based Exploration in Reinforcement Learning

by

Bryan Chen

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

EECS

in the

Graduate Division

of the

University of California, Berkeley

Spring 2021

Theory and Application of Bonus-based Exploration in Reinforcement Learning

by Bryan Chen

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

[illegible]

Professor Jiantao Jiao
Research Advisor

5/12/2021

(Date)

* * * * * * *

[illegible]

Professor Kannan Ramchandran
Second Reader

5/12/2021

(Date)

Abstract

Theory and Application of Bonus-based Exploration in Reinforcement Learning

by

Bryan Chen

Master of Science in EECS

University of California, Berkeley

Professor Jiantao Jiao, Chair

In this work, we strive to narrow the gap between theory and practice in bonus-based exploration, providing some new connections between the UCB algorithm and Random Network Distillation as well as some observations about pre- and post-projection reward bonuses. We propose an algorithm that reduces to UCB in the linear case and empirically evaluate the algorithm in challenging exploration environments. In the Randomised Chain and Maze environments, our algorithm consistently outperforms Random Network Distillation in reaching unseen states during training.

Contents

List of Figures

Acknowledgments

I would like to thank my advisor Jiantao Jiao and collaborator Paria Rashidinejad for their advice and feedback throughout this project.

Chapter 1

Introduction

Recently, deep reinforcement learning has been applied to a variety of problems with great success. However, even though there has been a flurry of work on theoretical results in reinforcement learning, the advancements are still quite far from the algorithms commonly used in practice. In particular, the analysis of one of the central questions of RL, the exploration-exploitation tradeoff, remains mostly limited to the linear function approximation case. Meanwhile, the plethora of algorithms used in practice with deep neural networks have little theoretical basis.

On tackling the exploration problem, one general idea that appears in both theory and practice is adding a bonus to the reward to encourage visiting unique states/actions. In theory, a common approach is to utilize the framework of *optimism in the face of uncertainty* to take actions. The intuition is to take actions that gives the largest plausible reward given the history; adding a bonus allows one to quantify the agent's uncertainty of the reward from an action. By contrast, in practice, reward bonuses are often used to augment the MDP, biasing the agent's learned model towards unseen actions.

Some practical reward bonuses [35], [9] attempt to estimate the visit counts of similar states/actions, motivated in a general sense by the provably efficient algorithms in the tabular RL case that use visit count bonuses for optimism [11],[33], [16], [6]. However, although demonstrating some success in practice, the motivation is unprincipled as the setting is actually the function approximation case. Thus, it is natural to ask whether we can devise algorithms that are inspired for this case.

The focus of this work is to demonstrate some connections between the algorithms used in practice and in theory for the function approximation case. We further propose an efficient algorithm utilizing these insights. In particular, our algorithm reduces to the celebrated UCB algorithm in the linear case [19], and works well in some empirical evaluations. On the Randomised Chain and Maze environments, our algorithm outperforms a popular practical algorithm, Random Network Distillation [12], in reaching unseen states during training.

Chapter 2

Related Work

Linear and nonlinear bandits: The stochastic linear (contextual) bandit case is well-studied [see [5], [14], [1], [28], [19]]. There are algorithms that can achieve near-optimal pseudoregret of $\tilde{O}(d\sqrt{T})$ [1] for the linear bandit case. Note that this case can be seen as a linear MDP with a one-length horizon. Next, the nonlinear bandit is much less well understood. The *Eluder dimension* was introduced by [29] as a measure of the complexity of a function class based on the degree of dependence among action rewards, allowing for regret bounds for general function approximation. However, [15] demonstrated that even for simple 2-layer neural networks, the Eluder dimension could be exponential in the input.

Tabular RL: There are many works on provably efficient algorithms in the tabular case. In particular, the model-based algorithms [11], [33], [16], [6] are all based on the optimism principle, using state-action counts to calculate a reward bonus. In the model-free case, [32], [17] are also provably efficient, with [17] utilizing state action counts for the bonus.

RL with function approximation: In the linear case, there has been some classical work proposing various algorithms without guarantees [see [23], [10], [7], [20]]. Recently, [18] demonstrated a provably efficient algorithm in this case. More generally, [37], [38] demonstrate regret bounds for algorithms with general function approximation in the deterministic case and [36] in the stochastic case, but these results depend on the Eluder dimension. There are also some results for policy-based methods such as [2], but we focus on value-based methods in our setting.

Practical bonus-based exploration: Recently in deep RL, there have been many practical methods proposed that add a bonus term to the reward. They attain great sample efficiency on hard exploration environments from the Arcade Learning Environment [8]. Most belong in the category of adding an intrinsic reward: count-based rewards that estimate how often similar states have been visited [9], [35] and next-state prediction rewards that use predictions about the transition to estimate the uncertainty of the state action pair [26], [12], [31], [25]. In particular, our method is closely related to [12], which is known for achieving amazing performance on Montezuma's Revenge. Many of these techniques are evaluated under a common setting by [34], which finds that although they can reproduce the performance gains on the hard exploration environments, the methods often fail to outperform epsilon greedy on easy environments. Note that all of these bonuses have little to no theoretical guarantees or even grounded in provable ideas. The idea of entropy regularization for policy-based methods [21] can be viewed as a bonus and has some convergence guarantees in simple cases [13], but [3] finds that its usefulness may be in aiding optimization rather than overcoming aleatoric uncertainty.

Chapter 3

The Linear Bandits Connection

In this section we highlight some of the theoretical motivations behind our approach. To this end, we analyze the simple case of linear bandits and demonstrate some connections to the practical ideas.

3.1 Problem Setup

We consider the stochastic bandits setting [19]. Let $\theta^* \in \Theta$ be the true model parameter that determines the bandit instance, $a \in A$ be the action in the action space, $\phi(a) \in \mathbb{R}^d$ be a feature mapping, the noise $\epsilon \sim \mathcal{N}(0,1)$, and $r(a, \theta^*)$ be the reward function. Then at each timestep t, the agent selects an action a_t and receives the reward $r(a_t, \theta^*) + \epsilon$. The goal of the agent is to minimize the pseuodoregret

$$\sum_{t=1}^{T} \arg\max_{a \in A} \mathbb{E}[r(a, \theta^*) - r(a_t, \theta^*)]$$

In this section we discuss the linear bandits case, but our practical algorithms are for the nonlinear case. Here, we consider the stochastic linear case, where $r(a, \theta^*) = \langle \theta^*, \phi(a) \rangle$.

The celebrated UCB algorithm [19] allows us to achieve a near-optimal pseudo-regret. The idea is to act on the principle of optimism in the face of uncertainty, or to take actions that are as good as plausible based on the evidence so far. In the linear case, the algorithm centers on the least squares estimate and creates an ellipsoidal confidence set around it that contains the "true" model parameters with high probability. The optimistic action then takes the form of an added bonus to the least squares estimate.

Algorithm 1 LinUCB algorithm

```
$X \in \mathbb{R}^{T \times d}$ is a design matrix with observations in each row
$r \in \mathbb{R}^T$ is a vector of the rewards over time
$\hat{\theta} \in \mathbb{R}^d$ is the weights of the estimator of the reward.
for t = 1, 2, ... T do
    Observe context $x_{t,a}$
    Choose arm $a_t = \arg\max_{a \in A}(\hat{\theta}^T x_{t,a} + 2\sqrt{\beta x_{t,a}^T (X^T X + \lambda I)^{-1} x_{t,a}})$ and observe reward $r_t$
    Update X and r with the observed context and reward
    $\hat{\theta} \leftarrow (X^T X + \lambda I)^{-1} X^T r$
end for
```

3.2 Connection to RND

In Random Network Distillation (RND), a popular practical method, [12] considered using the prediction error of a random network as a bonus. A neural network f_{RND} parameterized by θ is trained to predict the output of a frozen, randomly initialized neural network f with the same architecture:

$$b_{RND}(s) = ||f_{RND}(s;\theta) - f(s)||_2^2$$

It turns out that in the linear case, there is a close connection between RND and the linear UCB discussed previously. To be precise, we will show that a linear version of the RND bonus where the target is perturbed by Gaussian noise has the same form as the linear UCB bonus:

Theorem 3.2.1. *Let $X \in \mathbb{R}^{n \times m}$ be a design matrix with each row as a feature, $v \in \mathbb{R}^m$ be the frozen, randomly initialized teacher weights, $\theta \in \mathbb{R}^m$ be the RND predictor, and $\vec{\epsilon} \sim \mathcal{N}(\vec{0}, \sigma^2 I)$. Then, assuming that the targets have added Gaussian noise ϵ, the form of the RND bonus for a incoming feature ϕ is given by $\sigma^2 \phi^T (X^T X)^{-1} \phi$.*

Proof. As in RND, we train θ on the observations X to match the teacher, but we add noise to the final labels, i.e. $Xv + \vec{\epsilon}$. The problem and the least squared solution are as follows:

$$\min_\theta ||X\theta - (Xv + \vec{\epsilon})||^2 \tag{3.1}$$

$$\hat{\theta} = (X^T X)^{-1} X^T (Xv + \vec{\epsilon}) \tag{3.2}$$

The bonus value for an incoming observation ϕ is the squared prediction error of the learned $f_{RND}(\phi, \theta)$. We analyze the expected bonus value b_{RND}:

$$b_{RND}(\phi) \triangleq ((\hat{\theta} - v)^T \phi)^2 \tag{3.3}$$

$$
\begin{aligned}
\mathbb{E}[b_{RND}(\phi)] &= \mathbb{E}[\phi^T(X^TX)^{-1}X^T\epsilon\epsilon^TX(X^TX)^{-1}\phi] && (3.4)\\
&= \phi^T(X^TX)^{-1}X^T\mathbb{E}[\epsilon\epsilon^T]X(X^TX)^{-1}\phi && (3.5)\\
&= \sigma^2\phi^T(X^TX)^{-1}\phi && (3.6)
\end{aligned}
$$

□

Next, for intuition, note that the linear UCB bonus can be derived as the closed form of the *width* function of a particular confidence set (similar ideas are found in [36]):

Lemma 3.2.2. *Let the width w of a function family $\mathcal{F}$ at a point z be defined as $w(\mathcal{F}, z) = \max_{f_1,f_2\in\mathcal{F}} f_1(z) - f_2(z)$. Then, if $\mathcal{F}_l$ is given by all linear functions satisfying $\{f : \sum_i (f(\phi_i) - \hat{f}(\phi_i))^2 \leq \beta\}$ for some linear function $\hat{f}$, then $w(\mathcal{F}_l, z) = 2\sqrt{\beta z^T(X^TX)^{-1}z}$.*

Under the interpretation from this lemma, the forms of the bonuses imply that β and σ take on similar roles. In the UCB bonus, β is a measure of the error tolerance we want to incorporate to the estimator to capture the true reward function. In the RND interpretation, the uncertainty is captured by the standard deviation of the added Gaussian noise. When $\beta = 0$, then there is no uncertainty, so in the RND interpretation we would have $\sigma = 0$, $\hat{\theta} = v$, and the bonus is 0. Otherwise, the added noise represents the uncertainty we are giving the estimator.

Chapter 4

Approximate RND

We introduce an algorithm that reduces to UCB in the linear case and utilizes the connection to RND for a practical implementation.

4.1 Motivation

Before we establish the algorithm, we highlight an important difference between practical "intrinsic reward" style bonuses and UCB-type bonuses. In the former, it is a "pre-projection" bonus in the sense that the learner uses the bonus to influence how it learns the model; in the latter, the bonus is used to adjust a learned model to ensure certain properties during exploration. In this section we argue that only using intrinsic rewards in the linear bandit case could suffer from linear regret, suggesting that optimism should be used in exploration.

We begin by introducing the general algorithm in the linear case that uses intrinsic reward bonuses:

Algorithm 2 Pre-proj bonus algorithm

```
$X \in \mathbb{R}^{T \times d}$ is a design matrix with observations in each row
$r \in \mathbb{R}^{T}$ is a vector of the rewards over time
$\hat{\theta} \in \mathbb{R}^{d}$ is the weights of the estimator of the reward.
$b \in \mathbb{R}^{T}$ is a vector of the bonuses over time
for t = 1, 2, ... T do
    Observe context $x_{t,a}$
    Choose arm $a_t = argmax_{a \in A} \hat{\theta}^T x_{t,a}$ and observe reward $r_t$
    Update X, r, and b with observed context, reward, and computed bonus
    $\hat{\theta} \leftarrow (X^T X + \lambda I)^{-1} X^T (r + b)$
end for
```

Now, we argue that this type of algorithm cannot be optimal:

Theorem 4.1.1. *In Algorithm 2, for all possible bonus added, there exists an instance on which the algorithm suffers linear regret.*

Proof. We prove this by counterexample: Let $\theta^* = \begin{bmatrix} c \\ 2c \end{bmatrix}, 0 < c < 1/2$. The actions are $a \in \{0, 1\}$ and the corresponding context vectors are fixed for all t, $x_0 := x_{t,0} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, x_1 := x_{t,1} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. In the following, we show that if the first action is 0, regardless of the bonus, following the greedy algorithm will never explore the other action.

First, we show that given the first action is 0, the action afterwards is still 0. Let $M := x_0 x_0^T + \lambda I = \begin{bmatrix} 1+\lambda & 0 \\ 0 & \lambda \end{bmatrix}$ and least squares fitted to rewards is $\theta_r = M^{-1} x_0 r_1 = \begin{bmatrix} r_1/(\lambda+1) \\ 0 \end{bmatrix}$. Similarly, least squares fitted to bonus is $\theta_b = M^{-1} x_0 b_1 = \begin{bmatrix} b_1/(\lambda+1) \\ 0 \end{bmatrix}$. On each timestep we take the action:

$$\arg\max_{a \in \{0,1\}} \theta_r^T x_a + \theta_b^T x_a \tag{4.1}$$

Thus action 0 gives $\frac{r_1+b_1}{\lambda+1}$ and 1 gives 0, so the agent chooses 0. Now, assume that at a given timestep t, action 0 is chosen for all previous timesteps. $M_t = X^T X + \lambda I = \begin{bmatrix} t+\lambda & 0 \\ 0 & \lambda \end{bmatrix}$ so that

$$\theta_r = M^{-1} \sum_{\tau=1}^{t} x_\tau r_\tau = \begin{bmatrix} \sum_i r_i/(\lambda+t) \\ 0 \end{bmatrix} \tag{4.2}$$

$$\theta_b = M^{-1} \sum_{\tau=1}^{t} x_\tau b_\tau = \begin{bmatrix} \sum_i b_i/(\lambda+t) \\ 0 \end{bmatrix} \tag{4.3}$$

$$\tag{4.4}$$

Clearly, action 1 will never be chosen. Thus the regret is given by

$$R(T) = E[\sum_{t=1}^{T} x_*^T \theta^* - x_{a_t}^T \theta^*] \tag{4.5}$$

$$= T \cdot (2c - c) = Tc \tag{4.6}$$

In comparison, the UCB algorithm will choose action 1 when:

$$\frac{\sum_{\tau=1}^{t} r_\tau}{\lambda+t} + \alpha \sqrt{\frac{1}{\lambda+t}} < 0 + \sqrt{1/\lambda} \tag{4.7}$$

Note that if $\lambda = 1$ and $\alpha = 1$, it will choose action 1 after the first step. □

Note that while the intrinsic reward cannot introduce provable exploration in the linear case, in the deep RL setting it is widely used, possibly due to making the optimization landscape easier [27], [3]. This is part of the wide gap between theory and practice, and we find that our bonus can be used as an intrinsic reward effectively as well.

4.2 Algorithm

We first present an algorithm in the nonlinear bandit setting, which can be seen as a simplification of the standard deep RL setting.

Intuition. We would like to utilize a value estimate in order to help the agent make better decisions. Further, the first order approximation of the true reward nearby a model estimate θ_t is approximately linear in the gradient and θ^*. Thus, we may be able to benefit from the ideas in the provably efficient algorithms in the linear case. As with other value-based methods, the algorithm maintains an estimate of the reward at each timestep based on the historical actions and rewards it has observed. The key idea is that it uses the gradient with respect to the current model θ_t as the feature space ϕ for each step. Then, the standard linear UCB bonus is added to help explore when selecting actions.

Reduction to linear case. If we assume that the reward parameterizations are linear in the actions, then our algorithm exactly recovers the linear UCB algorithm. This is due to the fact that the gradient features are simply the actions themselves in this simple case. However, note that the algorithm is different than the setting where it is linear in the feature transformation of the actions $\phi(a)$. This is because the features ϕ_t are changing with each step as we update the model.

Algorithm 3 Nonlinear Bandit Algorithm

```
$\theta_0 \in \mathbb{R}^m$ is the initial parameterization of our model
$\Sigma_0 = \lambda I$ is the initial covariance of the features
for t = 1, 2, ... T do
    Set $\phi_t(\cdot)$ to be $\nabla_\theta r(\cdot, \theta)|_{\theta=\theta_t}$
    $a_t \leftarrow \arg\max_{a \in A_t} r(a, \theta_t) + \beta\sqrt{\phi_t(a)^T \Sigma_t^{-1} \phi_t(a)}$
    Pull $a_t$ to receieve reward $r_t$
    $\theta_t \approx \arg\min_\theta \sum_{s=1}^{t} (r_s - r(a_s, \theta))^2$
    $\Sigma_t \leftarrow \Sigma_t + \phi_t(a_i)\phi_t(a_i)^T$
end for
```

Deep RL

In order to devise a practical algorithm for the deep RL setting, we modify the well-known DQN algorithm [22]. The underlying algorithm itself is not too different: we maintain an estimate of a state-action value (Q) function through gradient descent, and add the same bonus when selecting actions at each step. Additionally, in order to work well in practice, we make use of the following details:

Approximate RND. In modern neural networks, overparameterization seems to be key to the amazing generalization performance [4]. However, utilizing our method would mean a feature space that grows linearly in the number of parameters, which could be a very large amount even for simple problems. Indeed, it would be unfeasible to invert (or even calculate) the covariance matrix for the bonus as it scales at least quadratically in the number of parameters. To deal with this, we use our insight in section 3, that RND has the same form as the UCB in the linear case. Thus, we replace our bonus with a prediction error bonus, where the networks are linear in the gradient features. This approach would be feasible as it scales only linearly in the number of parameters by using gradient descent to update the network.

Representation Size. As in [12], the prediction task is actually to output an embedding instead of a scalar. It fits with our earlier interpretation as the MSE in this case would simply be averaging the prediction error among multiple instances of the bonus, providing an estimate of the expected value.

Normalization. Following [12], we keep track of the running standard deviation of the bonus and divide rewards by it. This is due to the fact that the scale of the reward could be quite different among settings and it would be difficult to choose hyperparameters that work well.

Chapter 5

Results

In order to verify that our algorithm works well in practice, we test it on a few environments known for hard exploration. The environments used are the same as in [27] and we utilize their code for the baselines and experiments as well. Any deviations from their setting are noted in the descriptions.

5.1 Randomised Chain

We start with the simple environment of a randomised chain introduced in [24]. Like in [27], we use a chain of length 100. The agent starts the episode in the pictured state, and interacts with the given MDP for N+9 steps, after which the episode ends and the agent is reset. At each step, the agent has two actions available to it, moving left or right. Before the training begins, the environment is randomly initialized so that the action that moves left or right at each step is chosen randomly and fixed throughout the agent interactions. The agent receives a reward of 0.001 for going left in the state labeled 1, a reward of 1 for going right in the last state furthest to the right, and no reward otherwise. Thus, the optimal policy is to pick the action that goes right at each timestep. See Figure 5.1 for the representation of the chain we use. We also follow [24], [27] and use a thermometer encoding for the state.

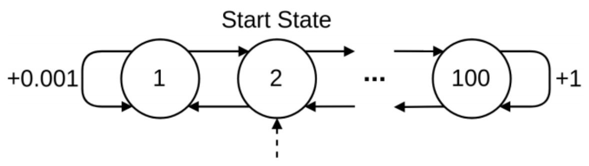

Figure 5.1: The Randomised Chain Environment

Experimental Details

Please refer to [27] for the default setup of the algorithm. We have the additional hyperparameters as follows: For RND, we use a representation size of 64, the network architectures are the same as the value networks, the networks are updated once using every 32 samples, and the optimizer is the same setting as the main networks (RMSProp with 5e-4). For our bonus, we use a representation size of 64, the networks are updated once using every 2 samples, and the learning rate is 1e-2.

To determine the sensitive hyperparameter β of scaling the bonus, we do sweeps: for the RND baseline, we do a search using 3 seeds each across $\beta = \{0.001, 0.01, 0.1, 1, 10\}$ and use $\beta = 0.01$. Similarly, we do the same search for our bonus across the same values, and we use $\beta = 0.001$ for intrinsic reward and $\beta = 0.01$ for the action selection. These values are then used for 6 seeds for the results shown.

Experimental Results

We compare the use of our bonus to a baseline RND intrinsic reward implementation. To do this, we implement our bonus both for the action optimism and as an intrinsic reward. Figure 5.2 demonstrates the unique states visited during the agent exploration per episode. It shows that our bonus, used in either way, works much better than the baseline RND in practice to encourage exploring unseen states.

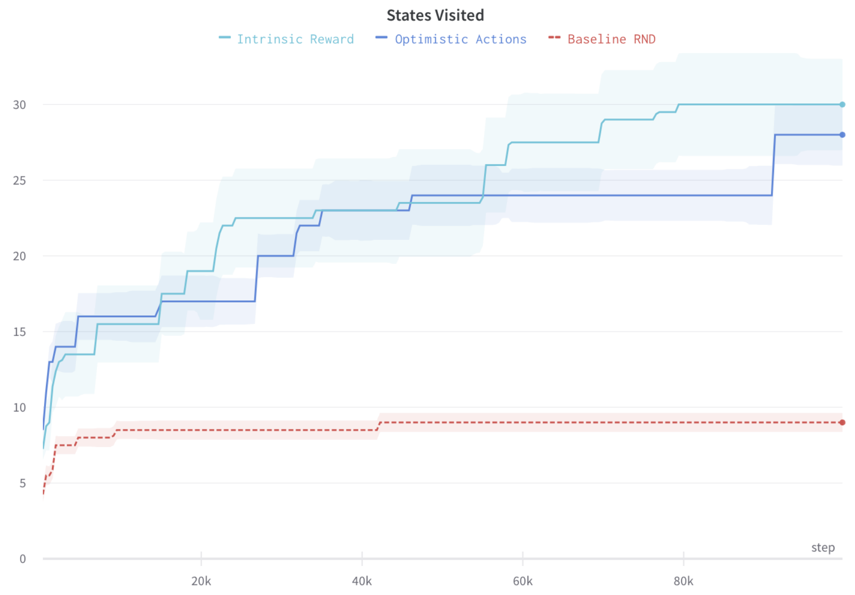

Figure 5.2: Comparison on the 100Chain environment. The median unique states visited is plotted, with standard error shaded in.

5.2 Maze

Next, we move to a more difficult environment, the Maze. In particular, it is a 2-dimensional gridworld where the agent can move up, down, left, or right. The objective is to navigate the maze to reach the marked goal, getting a reward of +10 at the goal and 0 otherwise. The horizon is 250, and the states are grayscale, 24 x 24 images in [0, 1]. Like in the chain environment, the effects of actions are randomised at the beginning of training. Figure 5.3 shows an example state in the environment.

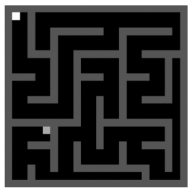

Figure 5.3: The Maze environment.

Experimental Details

Please refer to [27] for the default setup of the algorithm. The number of episodes is changed to 200k, and we use a target update interval of 200. Note that for the RND baseline-specific hyperparameters it is the same as [27], but in our setting it has stronger results. We have the additional hyperparameters as follows: For our bonus, we use a representation size of 128, the networks are updated once using every 32 samples, and the learning rate is 1e-2.

To determine the sensitive hyperparameter β of scaling the bonus, we do the same sweep as before, choosing $\beta = 0.1$ for the action selection. The values are used for 6 seeds for the results shown.

Experimental Results

Again, we compare the use of our bonus to a baseline RND intrinsic reward implementation. Here, we only use our bonus for optimistic action selection. Figure 5.4 demonstrates the unique states that are visited over episodes. Utilizing our bonus demonstrates increases exploration of unseen states, eventually reaching almost all of them (340 total).

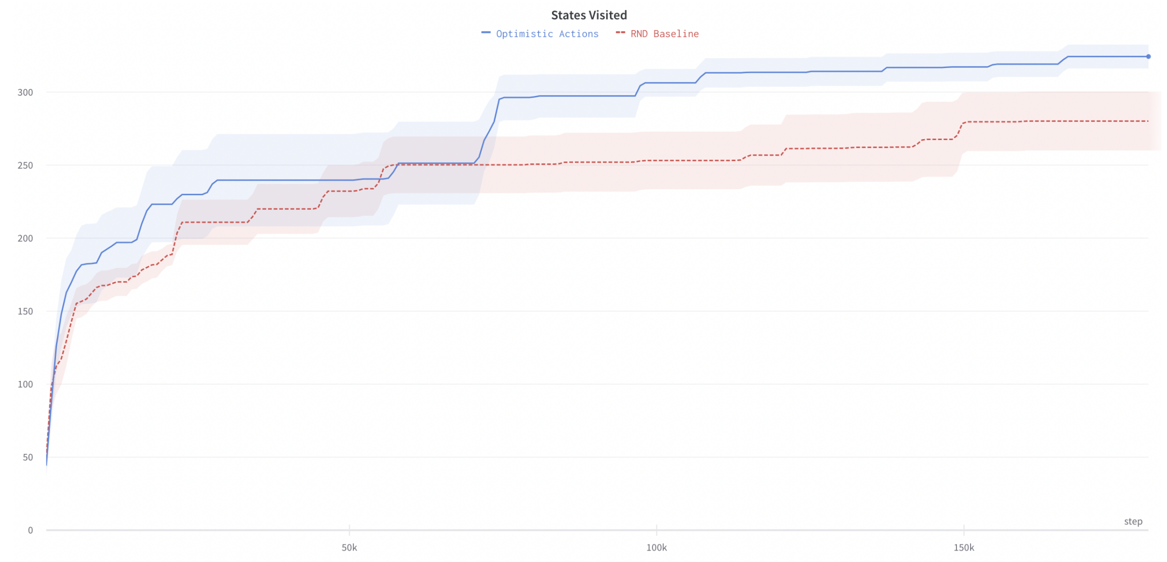

Figure 5.4: Comparison on the Maze environment. The median unique states visited is plotted, with standard error shaded in.

Chapter 6

Conclusion

In this work, we strive to narrow the gap between theory and practice in bonus-based exploration, providing some new connections between the UCB algorithm [19] and RND [12] as well as some observations about pre- and post-projection reward bonuses. We propose an algorithm that reduces to UCB in the linear case [19] and demonstrate empirically that it outperforms Random Network Distillation [12] in challenging exploration environments. For future work, we ask whether similar algorithms can provably converge to local optima and at what rates. Other interesting directions are to scale the algorithm to challenging environments like Montezuma's Revenge, or to try different algorithms based on other linear bandit ideas such as Thompson sampling [30].

Bibliography

[1] Yasin Abbasi-Yadkori, Dávid Pál, and Csaba Szepesvári. "Improved Algorithms for Linear Stochastic Bandits." In: *NIPS*. Vol. 11. 2011, pp. 2312–2320.

[2] Alekh Agarwal et al. "On the theory of policy gradient methods: Optimality, approximation, and distribution shift". In: *arXiv preprint arXiv:1908.00261* (2019).

[3] Zafarali Ahmed et al. "Understanding the impact of entropy on policy optimization". In: *CoRR* abs/1811.11214 (2018). arXiv: `1811.11214`. URL: `http://arxiv.org/abs/1811.11214`.

[4] Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. "Learning and Generalization in Overparameterized Neural Networks, Going Beyond Two Layers". In: *CoRR* abs/1811.04918 (2018). arXiv: `1811.04918`. URL: `http://arxiv.org/abs/1811.04918`.

[5] Peter Auer. "Using confidence bounds for exploitation-exploration trade-offs". In: *Journal of Machine Learning Research* 3.Nov (2002), pp. 397–422.

[6] Mohammad Gheshlaghi Azar, Ian Osband, and Rémi Munos. "Minimax regret bounds for reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 263–272.

[7] Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. "Efficient exploration through bayesian deep q-networks". In: *2018 Information Theory and Applications Workshop (ITA)*. IEEE. 2018, pp. 1–9.

[8] Marc G. Bellemare et al. "The Arcade Learning Environment: An Evaluation Platform for General Agents". In: *CoRR* abs/1207.4708 (2012). arXiv: `1207.4708`. URL: `http://arxiv.org/abs/1207.4708`.

[9] Marc G. Bellemare et al. "Unifying Count-Based Exploration and Intrinsic Motivation". In: *CoRR* abs/1606.01868 (2016). arXiv: `1606.01868`. URL: `http://arxiv.org/abs/1606.01868`.

[10] Steven J Bradtke and Andrew G Barto. "Linear least-squares algorithms for temporal difference learning". In: *Machine learning* 22.1 (1996), pp. 33–57.

[11] Ronen I Brafman and Moshe Tennenholtz. "R-max-a general polynomial time algorithm for near-optimal reinforcement learning". In: *Journal of Machine Learning Research* 3.Oct (2002), pp. 213–231.

[12] Yuri Burda et al. "Exploration by Random Network Distillation". In: *CoRR* abs/1810.12894 (2018). arXiv: `1810.12894`. URL: `http://arxiv.org/abs/1810.12894`.

[13] Shicong Cen et al. *Fast Global Convergence of Natural Policy Gradient Methods with Entropy Regularization*. 2021. arXiv: `2007.06558 [stat.ML]`.

[14] Varsha Dani, Thomas P Hayes, and Sham M Kakade. "Stochastic linear optimization under bandit feedback". In: (2008).

[15] Kefan Dong, Jiaqi Yang, and Tengyu Ma. "Provable Model-based Nonlinear Bandit and Reinforcement Learning: Shelve Optimism, Embrace Virtual Curvature". In: *arXiv preprint arXiv:2102.04168* (2021).

[16] Thomas Jaksch, Ronald Ortner, and Peter Auer. "Near-optimal Regret Bounds for Reinforcement Learning." In: *Journal of Machine Learning Research* 11.4 (2010).

[17] Chi Jin et al. "Is Q-learning provably efficient?" In: *arXiv preprint arXiv:1807.03765* (2018).

[18] Chi Jin et al. "Provably efficient reinforcement learning with linear function approximation". In: *Conference on Learning Theory*. PMLR. 2020, pp. 2137–2143.

[19] Lihong Li et al. "A Contextual-Bandit Approach to Personalized News Article Recommendation". In: *CoRR* abs/1003.0146 (2010). arXiv: `1003.0146`. URL: `http://arxiv.org/abs/1003.0146`.

[20] Francisco S Melo and M Isabel Ribeiro. "Q-learning with linear function approximation". In: *International Conference on Computational Learning Theory*. Springer. 2007, pp. 308–322.

[21] Volodymyr Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *CoRR* abs/1602.01783 (2016). arXiv: `1602.01783`. URL: `http://arxiv.org/abs/1602.01783`.

[22] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013). arXiv: `1312.5602`. URL: `http://arxiv.org/abs/1312.5602`.

[23] Ian Osband, Benjamin Van Roy, and Zheng Wen. "Generalization and exploration via randomized value functions". In: *International Conference on Machine Learning*. PMLR. 2016, pp. 2377–2386.

[24] Ian Osband et al. "Deep Exploration via Bootstrapped DQN". In: *CoRR* abs/1602.04621 (2016). arXiv: `1602.04621`. URL: `http://arxiv.org/abs/1602.04621`.

[25] Pierre-Yves Oudeyer, Frdric Kaplan, and Verena V. Hafner. "Intrinsic Motivation Systems for Autonomous Mental Development". In: *IEEE Transactions on Evolutionary Computation* 11.2 (2007), pp. 265–286. DOI: `10.1109/TEVC.2006.890271`.

[26] Deepak Pathak et al. "Curiosity-driven Exploration by Self-supervised Prediction". In: *CoRR* abs/1705.05363 (2017). arXiv: `1705.05363`. URL: `http://arxiv.org/abs/1705.05363`.

[27] Tabish Rashid et al. “Optimistic Exploration even with a Pessimistic Initialisation”. In: *CoRR* abs/2002.12174 (2020). arXiv: `2002.12174`. URL: `https://arxiv.org/abs/2002.12174`.

[28] Paat Rusmevichientong and John N Tsitsiklis. “Linearly parameterized bandits”. In: *Mathematics of Operations Research* 35.2 (2010), pp. 395–411.

[29] Daniel Russo and Benjamin Van Roy. “Eluder Dimension and the Sample Complexity of Optimistic Exploration.” In: *NIPS*. Citeseer. 2013, pp. 2256–2264.

[30] Daniel Russo et al. “A tutorial on thompson sampling”. In: *arXiv preprint arXiv:1707.02038* (2017).

[31] Bradly C. Stadie, Sergey Levine, and Pieter Abbeel. “Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models”. In: *CoRR* abs/1507.00814 (2015). arXiv: `1507.00814`. URL: `http://arxiv.org/abs/1507.00814`.

[32] Alexander L Strehl, Lihong Li, and Michael L Littman. “Reinforcement Learning in Finite MDPs: PAC Analysis.” In: *Journal of Machine Learning Research* 10.11 (2009).

[33] Alexander L Strehl and Michael L Littman. “An analysis of model-based interval estimation for Markov decision processes”. In: *Journal of Computer and System Sciences* 74.8 (2008), pp. 1309–1331.

[34] Adrien Ali Taıga et al. “Benchmarking Bonus-Based Exploration Methods on the Arcade Learning Environment”. In: *CoRR* abs/1908.02388 (2019). arXiv: `1908.02388`. URL: `http://arxiv.org/abs/1908.02388`.

[35] Haoran Tang et al. “#Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning”. In: *CoRR* abs/1611.04717 (2016). arXiv: `1611.04717`. URL: `http://arxiv.org/abs/1611.04717`.

[36] Ruosong Wang, Russ R Salakhutdinov, and Lin Yang. “Reinforcement learning with general value function approximation: Provably efficient approach via bounded eluder dimension”. In: *Advances in Neural Information Processing Systems* 33 (2020).

[37] Zheng Wen and Benjamin Van Roy. “Efficient exploration and value function generalization in deterministic systems”. In: *Advances in Neural Information Processing Systems* (2013).

[38] Zheng Wen and Benjamin Van Roy. “Efficient reinforcement learning in deterministic systems with value function generalization”. In: *Mathematics of Operations Research* 42.3 (2017), pp. 762–782.