Lower PAC bound on Upper Confidence Bound-based Q-learning with examples

Jia-Shen Boon, Xiaomin Zhang
University of Wisconsin-Madison
{boon,xiaominz}@cs.wisc.edu

Abstract

Abstract Recently, there has been significant progress in understanding reinforcement learning in Markov decision processes (MDP). We focus on improving Q-learning and analyze its sample complexity. We investigate the performance of tabular Q-learning, Approximate Q-learning and UCB-based Q-learning. We also derive a lower PAC bound $\Omega(\frac{|\mathcal{S}|^2|\mathcal{A}|}{\epsilon^2}\ln\frac{|\mathcal{A}|}{\delta})$ of UCB-based Q-learning. Two tasks, CartPole and Pac-Man, are each solved using these three methods. Some results and discussion are presented at last. UCB-based learning does better in exploration but lose its advantage in exploitation, compared to its alternatives.

1 Introduction

Reinforcement learning seeks to answer the question, *"How should an agent act in its environment to maximize its expected return?"* [8]. We explore several reinforcement learning algorithms to solve two tasks: Pac-Man, and Cartpole.

A reinforcement learning task that is *Markovian* is known as a Markov Decision Process. A MDP is defined by a 5-tuple $\langle \mathcal{S}, A, T, R, \gamma \rangle$. $\mathcal{S}$ is the set of all possible states that the environment can be in; A is the set of all actions that the agent can take; $T : \mathcal{S} \times A \times \mathcal{S} \mapsto \mathbb{R}$ is the **transition model** that states the probability of going to some next state conditioned on the current state and action taken from the current state; R is the **reward model** which states the reward that environment gives to the agent, commonly defined as a function of a state and action; γ is the **discount factor**, a scalar which gives the weightage of future rewards relative to immediate rewards.

A known MDP can be solved by **dynamic programming** algorithms such as policy iteration. In general though, we do not know the MDP, especially the transition model T. In such cases, we turn to other approaches such as **temporal difference learning** or **Monte Carlo methods**.

The solution to a MDP is an optimal **policy** $\pi^\star : \mathcal{S} \mapsto A$ that tells the agent what action to take for every state. In practice, we compute an action-value function $Q : \mathcal{S} \times A \mapsto \mathbb{R}$ instead. Temporal difference methods are *bootstrapping* methods that update action values from other action values.

All RL control methods maintain two policies - a **behavior policy** and an **estimation policy** [8]. An **off-policy** method is one in which these two policies are not the same. Q-learning, as well as all extensions explored in here, is off-policy.

Upper confidence bounding (UCB) policies explore the **K-armed Bandit problem**. An agent in such a problem is a gambler that seeks to maximize payout on a slot machine with multiple arms/levers/actions. This problem is a specific case of the RL problem. As such UCB can be used as the behavior policy of a RL agent. This policy was first introduced by Lai and Robbins [5].

2 Previous Work

Q-learning, as well as its extensions into linear approximation and eligibility traces, was introduced by Watkins [9]. Koki.et al[6] proposed Discounted UCB1-tuned for Q-Learning, which we named it UCB-based Q-learning in our report. They soon investigated the usability of the algorithm in [7].

3 Preliminaries

We now introduce a classical reinforcement learning algorithm: **Q-learning**. Our work also includes approximate Q-learning and UCB-based Q-learning. Q-learning is a temporal difference learning (TD learning) method. It uses an action-value function Q to return the estimated total future rewards. The key is to compare the expected reward to the reward given by the environments. The update is calculated as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max_{a' \in A(s')} Q(s',a') - Q(s,a)],$$

where α is the learning rate, $\alpha \in (0,1]$, γ is the discount factor.

Approximate Q-learning is a modification of standard (or "tabular") Q-learning. The Q function, instead of being a table of state, action and Q value triples, is a linear function $Q(s,a) = \vec{w}^T \vec{f}(s,a)$, where $\vec{w}$ is a vector of weights, and $\vec{f}$ is a feature extractor function. The key of approximating here is to extract some significant features from the environment's state. The feature extractor allows us to hand-code prior knowledge into the agent, but if the feature extractor is insufficiently expressive, states that share features may actually be very different in Q value, and the Q-function can fail to generalize.

UCB-based Q-learning aims to balance the exploration and exploitation. We use UCB policy on Q-learning. UCB is a method based on selecting the highest UCB value instead of the estimated value $\bar{X}_i$. The definition of UCB is as follows:

$$UCB_i = \bar{X}_i + C\sqrt{\frac{\ln n}{n_i}},$$

where i is the number of levers, n is the total number of pulls and n_i is the number of pulls on lever i. C is a parameter that determines the agent's tendency to explore.

UCB1-tuned is an improvement over the basic UCB method. Here, the agent chooses level i that maximizes the UCB1-tuned value defined as:$UCB1_i = \bar{X}_i + C\sqrt{\frac{\ln n}{n_i} \min\{\frac{1}{4}, V_i + \sqrt{\frac{2 \ln n}{n_i}}\}}$, where V_i is the variance of the explored values. If we set 1 as success and 0 as failure, V_i is upper bounded by $\frac{1}{4}$.
Discounted UCB1-tuned is a further improvement over the above, using weighted value and weighted variance. $\bar{X}_{i,n_i} = \frac{\sum_{j=1}^{n_i} x_{i,j} r^{n_i - j}}{\sum_{j=1}^{n_i} r^{n_i-j}}$, where $x_{i,j}$ is the j^{th} data of the i^{th} lever and r is the damping factor, $r \in (0,1)$. $V_{i,n_i} = \frac{\sum_{j=1}^{n_i}(x_{i,j}^2 - \bar{X}_{i,n_i}^2) r^{n_i-j}}{\sum_{j=1}^{n_i} r^{n_i-j}}$. This weighted variance could be updated iteratively as $V_{i,n_i+1} = \frac{r(1-r^{n_i}) + x_{i,n_i+1}^2 (1-r)(V_{i,n_i} + \bar{X}_{i,n_i}^2)}{(1-r^{n_i+1})(V_{i,n_i} + \bar{X}_{i,n_i}^2)} - \bar{X}_{i,n_i+1}^2$.

4 Analysis of Sample Complexity

In this section, we present a lower bound of the UCB-based Q-learning. We can think of a MDP as a directed graph. The vertices are the states and the edges are the transitions.

UCB-based Q-learning: 1. Keep track of MDP by the UCB policy; 2. Solve the MDP based on Q value; 3.Take action from the π^*

To analyze the lower bound of the algorithm, let's look at some theorems.

Lemma 1. The following three metrics for bandits are equivalent.

1. Identify near-optimal action (ϵ optimal) with high probability $1-\delta$ in time $\tau(|\mathcal{A}|, \epsilon, \delta)$ which is polynomial in $|\mathcal{A}|$, $\frac{1}{\delta}$, $\frac{1}{\epsilon}$.

2. Nearly maximize reward (ϵ' - near) with high probability $1 - \delta'$ in time $\tau'(|\mathcal{A}|, \epsilon', \delta')$.
3. Select a non- near optimal action ϵ'' no more than $\tau''(|\mathcal{A}|, \epsilon'', \delta'')$ times with high probability $1 - \delta''$.

Theorem 1. 1. Once all edges are known, no more mistakes will be made.

2. If we loop without learning anything, then we can stop visiting the unknown states.
3. We execute a path to an unknown state-action pair for $O(|\mathcal{S}|^2|\mathcal{A}|)$ steps.

Lemma 2. Hoeffding's Inequality. Let $x_1, x_2, x_3, ..., x_n$ be i.i.d with mean μ. Let $\hat{\mu}$ be our estimation of $\mu_i = \sum_i \frac{x_i}{n}$. The following is a $100(1-\delta)\%$ confidence interval for μ_i is $[\hat{\mu} - \frac{z_\delta}{\sqrt{n}}, \hat{\mu} + \frac{z_\delta}{\sqrt{n}}]$, where $z_\delta = \sqrt{\frac{1}{2}\ln\frac{2}{\delta}}$.

Lemma 3. To choose the action with best estimate with high probability $1 - \delta$ within ϵ optimal. If there are $|\mathcal{A}|$ actions. Then we should learn each action to $\frac{\epsilon}{2}$ with probability $1 - \frac{\delta}{|\mathcal{A}|}$.

Theorem 2. If we want to learn each action to $\frac{\epsilon}{2}$ with probability $1 - \frac{\delta}{|\mathcal{A}|}$, then we need at least $2\frac{1}{\epsilon^2}\ln\frac{2|\mathcal{A}|}{\delta}$ samples for each action.

Theorem 3. The lower bound of the UCB-based Q-learning algorithm is $\Omega(\frac{|\mathcal{S}|^2|\mathcal{A}|}{\epsilon^2}\ln\frac{|\mathcal{A}|}{\delta})$.

5 Model and Experiment

We attempt to solve two reinforcement learning tasks: Pac-Man and Cartpole (see Figures 1a and 1b). We describe CartPole briefly since the Pac-Man arcade game is well-known.

The CartPole task models a rigid block (or "cart") attached to the end of a rigid pole. Intuitively, the agent (user) has to balance the pole by actuating the cart either to the right or left. The transition model is given by physics equations of dynamics with some noise. The agent receives a reward of 1 at every non-terminal state. A state is terminal if the cart's location is beyond some fixed interval or when the pole's angular position is beyond some fixed angle away from the vertical.

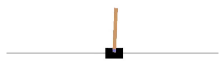
In both tasks, the respective libraries allow the agent to observe the high level state of the environment. The agent does not receive raw image data as input. For example, the state of the cartpole is a 4-tuple $\langle x, \frac{dx}{dt}, \theta, \frac{d\theta}{dt}\rangle$ where x is the linear position of the cart, and θ is the angular position of the pole. This state is exposed by the environment's API.

Pac-Man UCB/tabular Q-learning - The Pac-Man problem lends itself well to UCB/tabular Q-learning since the state space is discrete and can thus be learnt by the Q-agent without modification.
Pac-Man approximate Q-learning - The features extracted from the board are whether there is adjacent food, the distance from the closet dot, the number of adjacent ghosts, and a bias term[1].
CartPole UCB/tabular Q-learning - Since tabular Q-learning requires a discrete state space to prevent the table size from growing arbitrarily, we discretize the otherwise continuous state space. A grid in a 1D space is defined by the range $[x_{\text{low}}, x_{\text{high}}]$ and n cells. Any real number x then maps to one of the cells in the grid (up to a clipping operation), given by $\frac{n(x - x_{\text{low}})}{x_{\text{high}} - x_{\text{low}}}$, clipped between 0 and $n-1$. A 4D state is similarly discretized by a 4D grid. We find the range of the grid by running a random policy for several episodes and taking the limits of the explored state space as the limits of the grid range.
CartPole approximate Q-learning - The features extracted are the state vector concatenated with the action (0 or 1 corresponding to the left or right action) and a bias term. We also experiment with a 2D polynomial kernel to raise the expressiveness of the state vector.

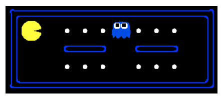
6 Results

We implement the tabular Q-learning agent, approximate Q-learning agent, and UCB Q-learning agent. The Pac-Man environment comes from the Berkley Pac-Man series [1]; the CartPole environment comes from OpenAI Gym, a public repository of RL environments [2].

[1]http://ai.berkeley.edu



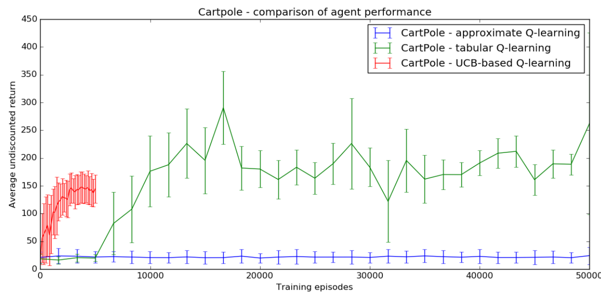
(a) CartPole



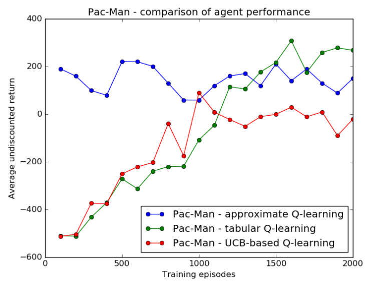
(b) Pac-Man

Figure 1: (a). Visualization of the CartPole task. The brown thin rectangle is the pole while the black rectangle is the cart. (b).Visualization of the Pac-Man task

The evaluation of CartPole uses parameters N_{check}, $N_{\text{episodes/check}}$ and N_{train}. The agent trains in the environment for N_{train} episodes. At N_{check} checkpoints, we evaluate the estimation policy by getting the agent to follow its estimation policy in the environment for $N_{\text{episodes/check}}$ episodes. The mean undiscounted return at each checkpoint indicates the agent's learning performance. This evaluation process is inspired by RLPy [4]; training and evaluation are separated because the agents are off-policy.



(a) Results on the CartPole task. Half-width of error bars are the standard deviation of $N_{\text{episodes/check}}$ episodes in each checkpoint.



(b) Results of the three agents on the Pac-Man task

CartPole task - Results are shown in Figure 2a. The approximate Q-agent fails to learn; the tabular Q-agent's performance peaks after 15000 iterations; the UCB-based Q-function peaks in less than 4000 iterations. The standard deviation of UCB-based Q-learning is smaller than the other methods. Since UCB-based Q-learning converges quickly (and is also much slower per episode than the other two methods), we only plot the first 5000 iterations.
Pac-Man task - Results are shown in Figure 2b. The approximate Q-agent learns quickly, within the first checkpoint; the tabular Q-agent learns more slowly but reaches similar performance to the approximate variant; the UCB-based Q-agent learns slightly more quickly than the tabular variant. Because of the coding environment of Pac-Man, we cannot check the rewards without ϵ greedy. In this way, the standard deviation doesn't make sense so that we do not plot them or compare them.

As aforementioned, approximate Q-learning heavily depends on the feature extractor. Good features can make learning much easier while bad features can make learning strictly harder. Tabular Q-learning is a good approach in general. UCB-based Q-learning does well in initial exploratory learning stage, at the expensive of heavier compute per state transition.

7 Conclusion and Discussion

We derive a lower bound of $\Omega(\frac{|\mathcal{S}|^2|\mathcal{A}|}{\epsilon^2}\ln\frac{|\mathcal{A}|}{\delta})$ for UCB-based Q-learning. Approximate Q-learning can be excellent if the features are chosen well. UCB-based Q-learning does a good job in less iterations and Q-learning does well in general.

For the future work, we will investigate carefully on sample complexity using both Pac-Man and CartPole. This asks us to be really familiar with the environment of these examples. We may also consider the problem within a fixed horizon MDPs[3]. We will also try to simplify and accelerate the UCB-based Q-learning in order to make it faster.

References

[1] Berkeley cs188. `http://ai.berkeley.edu/`. Accessed: 2016-05-13.

[2] Openai gym. `https://gym.openai.com`. Accessed: 2016-05-13.

[3] Christoph Dann and Emma Brunskill. Sample complexity of episodic fixed-horizon reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2800–2808, 2015.

[4] Alborz Geramifard, Robert H Klein, Christoph Dann, William Dabney, and Jonathan P How. RLPy: The Reinforcement Learning Library for Education and Research. `http://acl.mit.edu/RLPy`, 2013.

[5] Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

[6] Koki Saito, Akira Notsu, and Katsuhiro Honda. Discounted ucb1-tuned for q-learning. In *Soft Computing and Intelligent Systems (SCIS), 2014 Joint 7th International Conference on and Advanced Intelligent Systems (ISIS), 15th International Symposium on*, pages 966–970. IEEE, 2014.

[7] Koki Saito, Akira Notsu, Seiki Ubukata, and Katsuhiro Honda. Performance investigation of ucb policy in q-learning. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 777–780. IEEE, 2015.

[8] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[9] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

8 Supplementary material

Our code is available online[2].

Proof sketch for Lemma 1.
$1 \Rightarrow 2$: Set $\epsilon = \epsilon', \delta = \delta'$, given some algorithm ALG1 algorithm that satisfies Lemma 1.1, we can just run ALG1 for $\tau(|\mathcal{A}|, \epsilon, \delta)$ times to know a ϵ' action and do this action forever in the future.
$2 \Rightarrow 3$: Assume we have an algorithm named ALG2 satisfying Lemma 1.2. Set $\epsilon' = \epsilon', \delta'' = \delta'$ and just run $ALG2$. We allow any mistakes in the first $\tau'(|\mathcal{A}|, \epsilon', \delta')$ detecting actions. So the mistake is 1 at most. And we require $\epsilon'/2$ in the remaining ($\tau''(|\mathcal{A}|, \epsilon', \delta')$ - $\tau'(|\mathcal{A}|, \epsilon', \delta')$) actions, then 3 holds as long as

$$\epsilon' \geq \frac{1 * m + \frac{1}{2}\epsilon' * (\tau''(|\mathcal{A}|, \epsilon', \delta') - \tau'(|\mathcal{A}|, \epsilon', \delta'))}{\tau''(|\mathcal{A}|, \epsilon', \delta')}.$$

That is to say,

$$\tau''(|\mathcal{A}|, \epsilon'), \delta' \geq \tau'(|\mathcal{A}|, \epsilon', \delta')(\frac{2}{\epsilon'} - 1)$$

$3 \Rightarrow 1$: What we have now is algorithm satisfying 3. We call it ALG3. We can run ALG3 and get a perstep near optimal payoff. Assume e_i is the suboptimality of the plurality action, then $\epsilon'' \leq e_i * \frac{1}{|\mathcal{A}|}, e_i \leq \epsilon \Leftrightarrow \epsilon'' = \epsilon * \frac{1}{|\mathcal{A}|}$.

Proof sketch for Theorem 1.

1. This is obvious from Lemma 1.
2. This is obvious from Lemma 1.
3. Consider the worst case, whenever we want to get to a new state, we need to go through the already known states again. For each state, we need $O(|\mathcal{S}||\mathcal{A}|)$ steps. Totally we have $|\mathcal{S}|$ states. Thus, there are $O(|\mathcal{S}|^2|\mathcal{A}|)$ steps.

Proof sketch for Lemma 3. The probability of an action to be wrong(not ϵ optimal) is $\frac{\delta}{|\mathcal{A}|}$. Using Union Bound, we know that the probability of at least one action being wrong is $k * \frac{\delta}{k} = \delta$. Therefore, the probability of all the actions being right is $1 - \delta$.

Proof sketch for Theorem 2. From the Hoeffding's inequality and Lemma 3, noting c as the number of samples, then we need

$$\frac{\sqrt{\frac{1}{2}\ln(\frac{2|\mathcal{A}|}{\delta})}}{\sqrt{c}} \leq \frac{\epsilon}{2}$$

Therefore,

$$c \geq \frac{1}{\epsilon^2} \ln \frac{2|\mathcal{A}|}{\delta}.$$

[2]`https://github.com/boonjiashen/Berkeley_reinforcement_learning`

Proof sketch for Theorem 3. From theorem 1 and theorem 2, we know that the lower bound of the UCS built-in Q-learning algorithm is $(c_1|\mathcal{S}|^2|\mathcal{A}|) * (\frac{1}{\epsilon^2} \ln \frac{2|\mathcal{A}|}{\delta}) = \Omega(\frac{|\mathcal{S}|^2|\mathcal{A}|}{\epsilon^2} \ln \frac{|\mathcal{A}|}{\delta})$