Comparing The Performance Of Online and Offline N-Step TD Methods On A Simple Random Walk Task

Holt Spalding

, October 26, 2018

1. Background

1.1. Brief Overview of Reinforcement Learning

The goal of any reinforcement learning task is to determine the actions an *agent* ought to take in its *environment* at any given time in order to maximize some cumulative reward. For the purposes of this paper, I will constrain this very broad definition of reinforcement learning to simply mean the process by which one solves a *Markov Decision Process.* A Markov Decision Process (MDP) is just one of many models of how an agent interacts with its environment over discrete time steps ($t = 1, 2, 3, ...$). MDPs provide us the mathematical framework by which we can solve many so-called "sequential decision making problems", problems in which an agent must adapt to a potentially stochastic environment and make decisions in order to achieve its goals. Formally, an MDP is defined in terms of three basic elements: a *state space* $\mathcal{S}$ which represents the set of all possible states of the environment, an *action space* $\mathcal{A}$, which represents the set of all possible actions the agent can take, and the set $\mathcal{R}$ which represents the set of all possible numerical rewards received as a result of performing some action. Given its current state (denoted S_t where t is the current time step), an MDP agent must decide what action A_t to take in order to maximize cumulative rewards over time. In this context, performing an action is analogous to deciding which state to transition to next. Time steps are incremented by one every time an action is performed. Therefore, from here forward we let R_{t+1} and S_{t+1} represent the reward received and state reached at time step $t + 1$ after an action A_t has been performed at time step t. An episode describes the sequence of state-transitions and rewards an agent encounters until it reaches some termination state or goal state.

We say that the policy of an MDP-modeled RL agent is the set of "rules" that it utilizes in order to make decisions about what action to take next. Over the course of the learning process, RL agents seek to find the optimal policy, that is, they seek to always perform the action which maximizes cumulative reward given their current state. RL agents find this optimal policy in part by estimating the value of each state (denoted $V_\pi(s)$, ie. the value of state s given the current policy π). In this context, the "value" of a state s is the expected reward to be returned under the current policy after reaching s and transitioning away from it. Since an agent does not always explicitly know where the reward lies in its environment, it can only make educated guesses about the value of each state based on experience. This falls under the domain of what is known as value function approximation, and it is the topic of this paper. In this paper, we will compare two different method of approximating V^*, the true state-value function of a given environment.

1.2. Monte Carlo and TD Methods

Monte Carlo and n-step temporal difference methods are some of the most common methods of value function approximation used in reinforcement learning. Agents that utilize Monte Carlo methods learn and update their state-value function by running an episode until termination, observing the returns received from each state[1], and then updating the value function on the basis of that experience. This updated value function can then be used to develop a better policy, but that lies out of the scope of this paper. In very basic Monte Carlo prediction, an agent estimates the value of a state by averaging all the returns it received from that state over all episodes.

Temporal difference or TD learning methods differ from Monte Carlo methods in that they don't require the agent wait till the end of an episode to update their state-value function; they can do this online. In n-step TD methods (n is an arbitrary hyperparamter), the value of a given state S_t is updated by summing the returns received n-steps into the future, along with a discounted value estimation of the state S_{n+1} under our current state-value

[1]not always *each* state depending on the algorithm, but put that aside for a moment

function. As you'll see in the next section, this new and improved value estimate of S_t can then help us update our old value estimate of S_t. Over the course of the learning process these updates should slow down, until the value function estimate forms a good approximation of the "true" state-value function of the environment.

1.3. Formalizing The Previous Section

To put things into formal terms, imagine an episode in which the state-reward sequence preceded as follows: $S_t, R_{t+1}, S_{t+1}, R_{t+2}, \ldots R_T, S_T$ where T represents the time step of the terminal state. A Monte Carlo method would update it's value approximation of the state S_t in the direction of discounted returns G_t defined as follows...

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T,$$
$$\text{where } \gamma \in [0,1] \text{ represents a discount factor.}$$

So, if cumulative returns following S_t were high in an episode, our agent would tend to raise its approximation of $V(S_t)$, and if returns were low or negative the approximation of $V(S_t)$ would diminish or remain stagnant.

Given the same state-reward sequence, the approximation of $V(S_t)$ of an agent using one-step TD methods would tend towards the following...

$$G_{t:t+1} = R_{t+1} + \gamma V_t(S_{t+1})$$

Here, $G_{t:t+1}$ is equal to the return observed immediately after an action was taken in S_t, plus a discounted estimation of the value of state S_{t+1} under the 'yet-to-be' updated value function. Assuming I've done my job in explaining this correctly, it should be obvious that updates to the state-value function with one-step TD methods can occur online as soon as R_{t+1} is observed. Two step TD methods are similar, their approximation of $V(S_t)$ tending towards the following...

$$G_{t:t+2} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V_{t+1}(S_{t+2})$$

This leads to the more general form for what is called "corrected n-step truncated return" defined as follows...

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

From this, we can derive our n-step TD update rule for our approximation of $V^*(S_t)$ as follows..

$$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha[G_{t:t+n} - V_{t+n-1}(S_t)],$$

where $\alpha \in (0, 1]$ is a hyperparameter representing our learning rate.

The above shows that our new value estimate of the state S_t should be equal to our current value estimate of S_t plus the "error" of our previous value estimate of S_t weighted by our learning rate. The second term of the equation, $[G_{t:t+n} - V_{t+n-1}(S_t)]$ is known as "TD-error", since it represents the difference between our previous inferior value estimate of S_t and the better, newest value estimate of S_t, $G_{t:t+n}$. It should hopefully be fairly intuitive at this point why this update rule would result in incremental improvements state-value function estimates.

You may be asking yourself, what is the update rule for an agent employing n-step TD methods if episodes always end in under n time steps. When this occurs, our so-called bootstrapping term $(\gamma^n V_{t+n-1}(S_{t+n}))$ is dropped from our calculation of $G_{t:t+n}$, and we only utilize discounted cumulative returns in our calculation of $G_{t:t+n}$, just like Monte Carlo methods. This is why we say Monte Carlo methods are a special case of n-step TD learning methods. More specifically, they can be classified as an infinite-step TD method, since our bootstrapping term is "never reached". Sutton & Barto further prove that Monte Carlo error $(G_t - V(S_t))$ can be written in terms of the sum of TD errors (generally denoted $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ for TD(0)) when our value estimates are not updated from time step to time step, but instead updated at the end of each episode. The proof of that for interested parties can be found below...

$$\begin{aligned}
G_t - V(S_t) &= R_{t+1} + \gamma G_{t+1} - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1}) && \text{(from (3.9))} \\
&= \delta_t + \gamma\big(G_{t+1} - V(S_{t+1})\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\big(G_{t+2} - V(S_{t+2})\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}\big(G_T - V(S_T)\big) \\
&= \delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2} + \cdots + \gamma^{T-t-1}\delta_{T-1} + \gamma^{T-t}(0 - 0) \\
&= \sum_{k=t}^{T-1} \gamma^{k-t}\delta_k. && (6.6)
\end{aligned}$$

The question then remains, how do "online" TD methods, which update their value functions over the course of an episode, compare with "offline" TD methods which, like Monte Carlo, wait until the end of an episode in order to update their value functions (the main difference being, the bootstrapping terms in offline methods are always "one episode behind"[2]). In the following short demonstration I will empirically show how the two methods differ and why online methods are generally faster and algorithmically superior.

2. My Experiment

I have adapted an experiment created by Sutton & Barto (1998) to show the drastic performance difference between online and offline n-step TD methods on a random walk task. The task can be visualized in the following diagram...

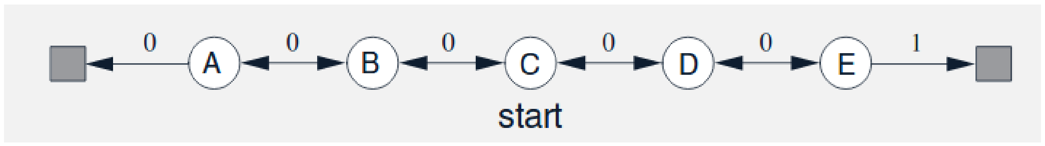

Figure 2: page 102 of Sutton & Barto (1998)

In this form of the task, there are 7 states, A,B,C,D,E, and two terminal states on either side of the environment. The agent begins in the center of the graph at state C, and is allowed to randomly walk around the environment until it reaches a terminal state. The direction of the arrows denote acceptable actions/state-transitions the agent can take. The reward for reaching the far right state is 1, and the reward for reaching any other state is 0. The Bellman equation which we need not delve into, tells us that the true value v^* of each state from left to right is $0, \frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}$, and 1 respectively. And this makes intuitive sense. The more left you move, the further you get from the reward/goal state, and therefore more leftward states should be valued less than more rightward states. If we were to let an online TD agent and an offline TD agent randomly walk through this environment for a given number

[2]They become significantly more episodes behind as the number episodes increases

of cycles, which would you expect to achieve better approximations of these true values?

3. Results

As Sutton Barto (1998) have done in Example 7.1 of their textbook[1], I have extended this task to 19 states (21 including terminal states) and observed the performance difference between online and offline n-step TD agents with different settings of both n and α, the learning rate. All agents had a discount factor of 1. Also, as Sutton & Barto have done, I gave the leftmost terminal state a reward of -1. Each agent performed 200 independent trials, 10 episodes each, and then the results of those 200 trials were averaged and graphed. Performance was measured in terms of root mean square error, a measure of how well a model can predict a true set of values. In this case, I measured how well these different agents were at approximating the true values of each state as defined above (for clarity, the true value of each state spans -1 to 1 split into uniform intervals, with the leftmost state having value -1 and the rightmost state having value 1). In the below graphs, an agent receiving RMS error of 0 would represent the agent which most accurately approximates the true value of each state (impossible to achieve in practice), and an agent receiving RMS error of 1 would represent the agent which least accurately represents the true value of each state. It should be fairly obvious from Figures 3 and 4 below that online TD methods outperformed offline methods in every instance. The RMS error of every online n-step agent at every level of alpha formed a lower bound on the RMS error of every offline n-step agent. This shows that online agents can perform superior value approximation in less time than offline methods can on an this random walk task.

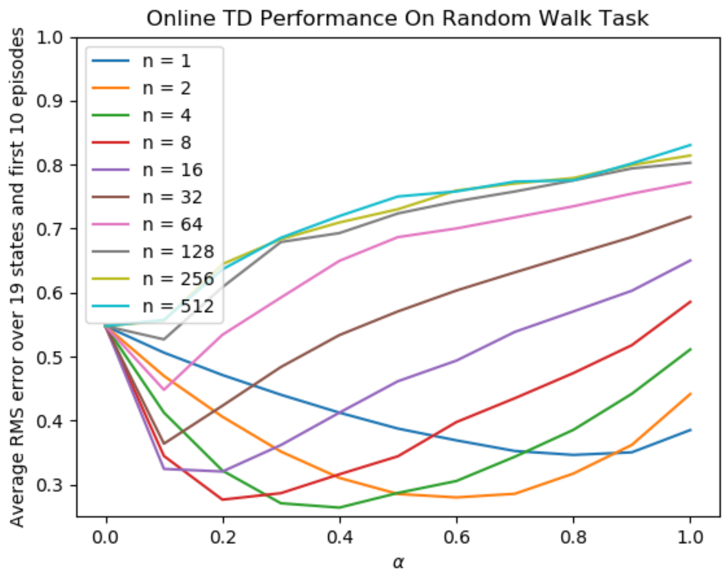

Figure 3:

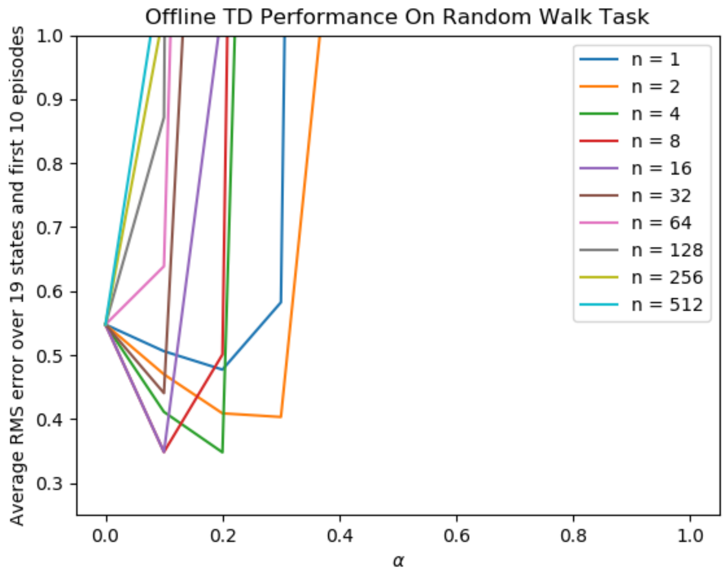

Figure 4:

4. Brief Discussion

In this demonstration, I hope to have empirically shown that online TD methods are generally superior to offline TD methods in terms of time it takes them to produce an accurate value function approximations. It can be inferred from this evidence that online TD methods also converge faster to their optimal policy. The literature I've come across would seem to support this, but I have not yet found any research that has definitively proven this on a theoretical basis. In the future I would like to know 1) the exact conditions under which online and offline methods perform differently and 2) the theoretical basis for why this is the case. It is my assumption that there must be some tasks in which both online and offline methods perform the same, though these tasks are probably impractical and uninteresting (maybe a two state random walk?).

Intuitively, it makes a lot of sense why online methods would outperform offline methods on this random walk task, the reason being that you've lost the compounding effect of your bootstrapping term in making updates in the offline method. The lack of this online bootstapping term would naturally result in smaller updates and therefore slower time to convergence. Another question I had was whether or not offline TD learning and Monte Carlo differ at all. Some people seem to use them interchangeably, but my assumption is that Monte Carlo methods always look to the end of an episode when calculating cumulative reward whereas offline TD-methods only look n-steps steps ahead. If this is the case, then offline TD methods *are* Monte Carlo methods at their best, and impractical theoretical models their worst.

I would like to learn more about is how Monte Carlo methods and online TD methods can be combined, and also the potential benefits of Monte Carlo methods since up until this point I have only tried to find reasons why MC is worse than TD. One paper I've found thus far which has addressed this was published by Gelly & Silver (2007). Silver, if I'm not mistaken, is a DeepMind researcher who has been one of the leaders in developing RL agents that can play Go. In their 2007 paper, Gelly & Silver show how training an agent offline based on prior knowledge can greatly accelerate performance online, specifically in the game of Go. Most RL researchers seem to take this approach: offline first, online second. I wonder if there are or ever will be practical circumstances under which this paradigm may change.

5. References

1. Sutton, R. S., & Barto, A. G. (1998). Reinforcement learning: An introduction. MIT press.

2.Gelly, S., Silver, D. (2007, June). Combining online and offline knowledge in UCT. In Proceedings of the 24th international conference on Machine learning (pp. 273-280). ACM.