Replicating Sutton's TD(λ) Algorithm

Carlos Souza
Georgia Institute of Technology
São Paulo, SP, Brazil
souza@gatech.edu

June 1st, 2019

Abstract

TD(λ) is a core algorithm of modern reinforcement learning. The purpose of this paper is to replicate results obtained in Richard Sutton's 1988 Learning to Predict by the Methods of Temporal Differences, in which he introduced temporal difference algorithms, theories and insightful experiments. We review Sutton's methods, replicating the experiment protocols he used to generate the overall same results he achieved in 1988 in the Random Walk Markov Reward Process. Finally, we present and discuss the results obtained, and propose a video animation to facilitate visualizing how differently TD(0) and TD(λ) propagates prediction levels throughout the learning process.

***K*eywords** Reinforcement learning · Temporal-difference (TD) learning · Markov reward process

1 Introduction

Temporal-difference (TD) learning is a core learning technique in modern reinforcement learning ([1], [2], [3], [4]). TD learning generates good estimates for expected returns by quickly bootstrapping from other expected-return estimates. TD(λ) is one of the most popular algorithms in TD learning. It was first introduced in Sutton's article *Learning to Predict by the Methods of Temporal Differences*, one of the most referenced articles in the field with over 5,000 citations. The purpose of this paper is to discuss the results obtained by replicating Sutton's TD(λ) algorithm, providing a complete description of the experiments, implementation details and outcomes.

1.1 Background

Our focus in this paper are *Markov Reward Processes*. MRPs are stochastic processes that can be described by $\langle \mathcal{S}, p, r, \gamma \rangle$, where $\mathcal{S}$ is the set of all possible states, $p(s'|s)$ is the transition probability from state $s \in \mathcal{S}$ to state $s' \in \mathcal{S}$, $r(s, s')$ is the reward obtained in the transition, and γ is the discount factor.

We will also focus on episodic MRPs, processes that contain *terminal states*. These states divide the sequences of state transitions into *episodes*. Once a terminal state is reached, the episode ends and the state is reset to its initial value. The return at time step t is the discounted sum of all the rewards observed after this time step until the end of the episode:

$$G_t = \sum_{i=1}^{T-t} \gamma^{i-1} R_{i+1} \tag{1}$$

where T is the time step of the terminal state.

Our goal is to learn the state-value function v that maps each state $s \in \mathcal{S}$ to the expected return:

$$v(s) = \mathbb{E}\{G_t \mid S_t = s\} \tag{2}$$

One of the simplest ways to solve a MRP is using Monte Carlo methods. These methods solve reinforcement learning problems by using experience from interactions with the environment to update their estimate V of v for the nonterminal state S_t ocurring in the experience. Monte Carlo methods wait until the end of the episode, until the full return following the visit is known, to use this value to update current estimate of the state-value function:

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t)) \tag{3}$$

where α is the learning-rate. The problem with Monte Carlo methods is that G_t is only known by the end of the episode, which means these methods must wait until the end of the episode to determine the increment to $V(S_t)$. TD methods, in other hand, need to wait only until the next step $t+1$. The simplest TD method, TD(0), uses the observed reward R_{t+1} and the current estimate $V(S_{t+1})$ from the next time step to update value-function estimate of current state:

$$V(S_t) \leftarrow V(S_t) + \alpha \underbrace{(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))}_{\delta\text{, known as TD error}} \tag{4}$$

$TD(0)$, also called *one-step TD*, is just an instance of a more general class of algorithms called TD(λ), where $\lambda = 0$. The general TD(λ) unify and generalize TD and Monte Carlo methods, producing methods that have Monte Carlo at one extreme ($\lambda = 1$) and one-step TD at the other ($\lambda = 0$). Its update rule is similar to the TD(0):

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))\, e(S_t) \tag{5}$$

however, it is applied to *all states* accordingly to their *eligibility* $e(S_t)$. The eligibility of a state is the degree to which it has been visited in the past. It is a short-term memory vector that is bumped in the component correspondent to the state recently visited, then fading away in proportion to λ. Eligibility can be updated incrementaly as:

$$e(s) \leftarrow \begin{cases} \gamma\lambda\, e(s) + 1 & \text{if } s = \text{current state} \\ \gamma\lambda\, e(s) & \text{otherwise} \end{cases} \tag{6}$$

We can use equations (5) and (6) to devise TD(λ) algorithm:

Algorithm 1 Value-state prediction with TD(λ)

```
Initialize state-value function $V$ arbitrarily
**for** $episode \leftarrow 1,\, M$ **do**
    Initialize $S$
    Initialize $e$ to 0 for all $s \in \mathcal{S}$
    **for** $t \leftarrow 1,\, T$ **do**                                  ▷ For each time step $t$ in the $episode$
        Observe reward $R_{t+1}$ and next state $S_{t+1}$
        $e(S_t) \leftarrow e(S_t) + 1$
        $\delta \leftarrow R_{t+1} + \gamma V(S_{t+1}) - V(S)$
        $V(s) \leftarrow V(s) + \alpha\, \delta\, e(s)$ for all $s \in \mathcal{S}$
        $e(s) \leftarrow \gamma\, \lambda\, e(s)$ for all $s \in \mathcal{S}$
        $S_t \leftarrow S_{t+1}$
    **end for**
**end for**
```

1.2 Random Walk

Sutton (1988) illustrates TD methods with a random-walk example, one of the simplest dynamic systems. A bounded random walk is a Markov Reward Process. The episodes always start in the middle state D. At each time step, the walk moves to the neighbor state, either left or right, with equal probability. The episode ends when walk reaches either extreme states A or G. If termination happens by reaching the extreme right (G), environment provides a reward of +1; all the other rewards are zero.

A typical episode terminating in the extreme right might be the following state-reward sequence: $D, 0, C, 0, D, 0, E, 0, F, 0, G, 1$. An example of a walk ending in the extreme left might be: $D, 0, C, 0, B, 0, A, 0$.

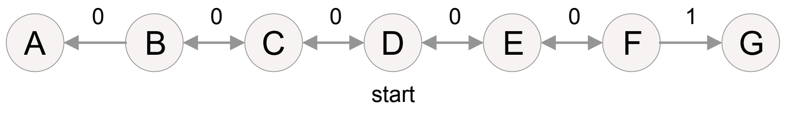


Figure 1: A bounded random walk generator used in Sutton's original paper. All episodes start in state D. At each time step, walk has equal probability of either moving left or right. Episode ends when walk reaches either A or G. In case of episode terminating in G, +1 reward is received; otherwise, reward is zero.

This MRP is undiscounted ($\gamma = 1$); therefore, the true value of each state is the probability of terminating on the extreme right by starting from that specific state. Thus, the true values from all states, from A to G respectively, are

$$v_{true} = \left[0, \frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}, 1\right] \tag{7}$$

We applied TD(λ) algorithm to effectively predict these true values, replicating Sutton's original paper methods and results. The following sections explain in details how we implemented the experiments, their results, and compares them with the previously published outcomes.

2 Methods

To replicate Sutton's results, we followed the exact protocols described in the article. First, we generated the data to be used by all learning procedures, by simulating 100 training sets, each containing 10 episodes (or sequences). Then, we implemented the experiments using this data. To measure how good the predictions were, we used the root mean squared (RMS) error between the predictions and the true values.

While implementing the experiments, we changed how the algorithm was written. Sutton developed all the math in his paper and the algorithms considering that a prediction P_t at time step t is function of i) a vector x_t of observations available at this time step (representing thus the state), and ii) a vector of modifiable weights, w. This means he expressed all learning procedures and equations as rules for updating w. However, the same learning procedures and equations can also be expressed as rules for updating state-value function V directly, as $w = [V(S_1), V(S_2), V(S_3), ...]$, i.e. the vector in which each element is the state-value function $V(s)$ for each $s \in \mathcal{S}$. As the underlying math is exactly the same, we arrived at the same results and conclusions, as we will show in the next section.

The first experiment assessed how different values of λ produced different predictions with different corresponding RMS errors. We used $\lambda = 0, 0.1, 0.3, 0.5, 0.7, 0.9$ and 1. There are 2 special cases in these values: 0 and 1. TD(0) is the *one-step* TD, which updates the value-function using the reward from only one step ahead. TD(1) is an instance of the Monte Carlo methods, which updates value-function only after the episode is complete, using all the rewards until the end of the episode. This case of $\lambda = 1$ is also referred as Widrow-Hoff in the original paper.

In this experiment, the state-value function was not updated neither online (i.e. after each time step, as shown in the algorithm 1) nor by the end of each episode. Instead, the function was only updated after a full loop over all episodes in the training set. Also, following the paper, we implemented the *repeated presentations* training paradigm in this experiment: we executed the training loop for each training set repeatedly, using the same data, until the state-value function did not change significantly. Here the paper is unclear about: i) the learning rate (α) used, ii) how to assess significant changes in the state-value function throughout the simulation, and iii) the minimum threshold difference between different estimates of state-value functions, below which the simulation ends. We used $\alpha = 0.01$ (as the paper mentions it used a small learning rate to ensure conversion), RMS error as a way to assess with a single scalar the difference between two estimates of value-functions, and $\theta = 10^{-4}$ as threshold to stop the simulation: whenever the RMS error between two estimates of different value-functions fall below this, the simulation ends.

In the second experiment we simulated different values of α and different values of λ, to understand the impact of the learning rate in TD. Three important changes were made in comparison to the first experiment. First, the training loop was executed only once for each training set. Second, the state-value function was updated by the end of each episode. Third, different than the first experiment, where the state-value function was initialized randomly, now the state-value function was initialized at constant value of 0.5 for all states. We varied α from 0 to 0.6 in steps of 0.05, and γ from 0 to 1 in steps of 0.1.

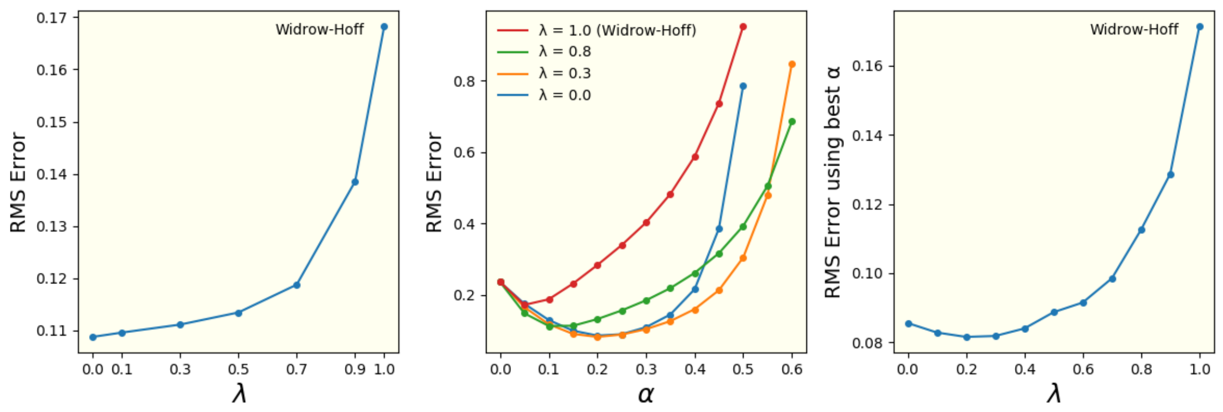


Figure 2: Replication of Figures 3, 4 and 5 from Sutton's original paper. In the left, average error on random walk problem under repeated presentations. In the center, average error on random walk problem after 10 episodes for different values of α and γ. In the right, average error at best α on random walk problem for different values of γ.

3 Results and Discussion

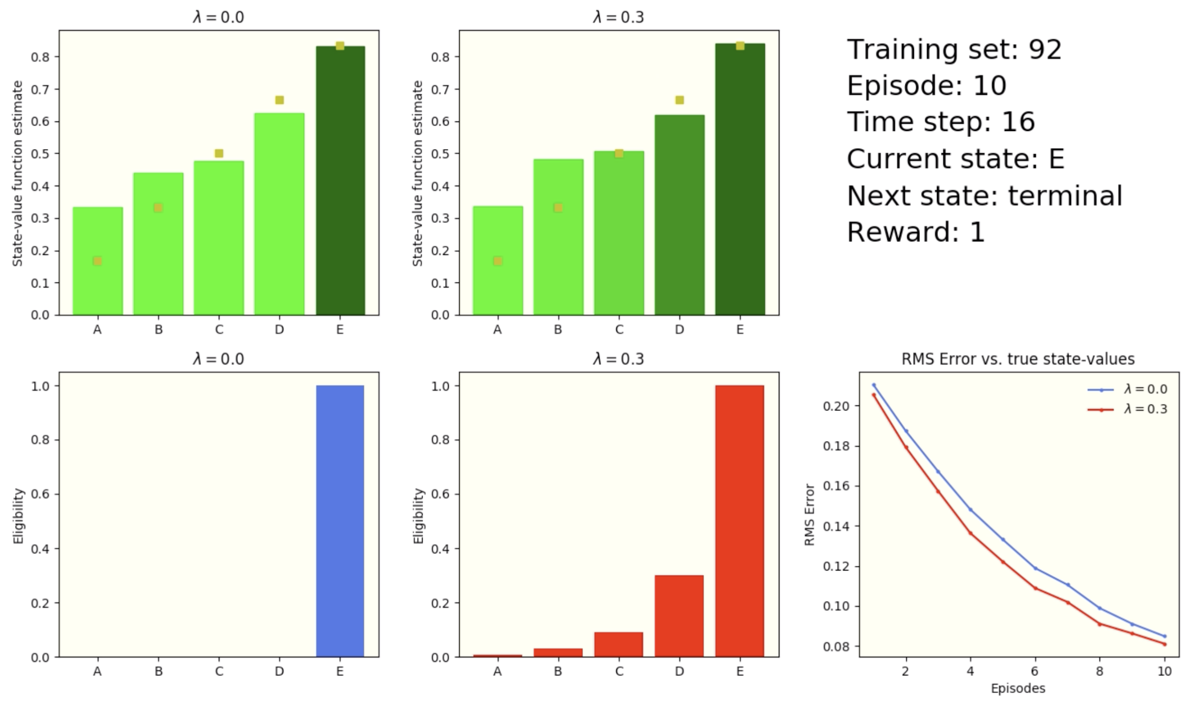
Figure 2 summarizes the results obtained by replicating Sutton's experiments. In the left, the chart shows the results of the first experiment, where RMS errors were calculated between the state-value function true values and the ones estimated after repeated presentations until convergence, for different values of λ. We averaged the errors over 100 training sets, exactly as done by Sutton. This chart replicates Figure 3 in the original article. Exactly as demonstrated by Sutton, we see, after running the first experiment, that lower values of λ produced smaller RMS errors. In fact, the best λ was zero, and the worst was 1. Despite the fact Widrow-Hoff procedure (TD(1)) is an optimal method to minimize error between predictions and actual outcomes in a training set, TD(0) converged to a better state-value function than TD(1). That's because, when presented repeatedly to a finite training set, TD(0) converges to the *optimal estimates*, while Widrow-Hoff (TD(1)) converges *to the estimates that minimize the error on the training set*, which are different from the optimal estimates.

In the center, the chart shows average errors obtained in a single run of each training set, averaged over 100 training sets, for different values of α and λ. This chart replicates Figure 4 in the original article. Here, again we confirmed Sutton's empirical results. We showed how TD methods produced smaller errors for all $\lambda < 1$, when compared to Widrow-Hoff (TD(1)). We also showed how different learning rates produced significantly different state-value function estimates, being the best learning rates around 0.2, as also shown by Sutton. As expected, we showed how too small learning rates made all learning procedures generate similar errors (i.e. all the different procedures did not learn successfully with smaller values of α). And we could observe as well how too large learning rates caused all learning procedures' errors to explode, confirming our expectation that learning rates cannot be neither too small nor too large for learning algorithms to work.

Finally, in the right, we can see the chart that replicates Figure 5 in Sutton's article. The results were produced by experiment 2, after a single run of each training set, also averaged over 100 training sets, for different values of α and λ. For each value of λ, we selected the best value of α. Again, our results matched Sutton's: all estimates of state-value function for $\lambda < 1$ produced lower errors than in the $\lambda = 1$ case. However, as shown by Sutton, the best case was not $\lambda = 0$, but a value for λ around 0.3. That's because TD(0) is slow at propagating prediction levels back throughout the episode, given eligibility factor. TD algorithm uses the eligibility factor, which in case of $\lambda = 0$, updates only the previously visited state in a given episode. For $\lambda > 0$, eligibility updates not only the previously visited state, but all states proportionally, which propagates the prediction levels faster. This is not a problem when the training set is repeatedly presented, but that was not the case in experiment 2, when the training set was seen only once by the learning algorithm.

To try to visualize how fast TD(λ) propagates prediction levels to other states in comparison to TD(0), for any $\lambda > 0$, we created a video animation of experiment 2 protocol, which can be seen in the following Youtube link:

`https://youtu.be/mBqyQpL8_Vc`



Training set: 92
Episode: 10
Time step: 16
Current state: E
Next state: terminal
Reward: 1

Figure 3: Screenshot of the video animation showing experiment 2 protocol, performed for 2 TD learning procedures in parallel. The first, TD(0), is shown in first column of charts. The second, TD(λ) for $\lambda = 0.3$, is shown in the second column of charts. In the first row of charts, we show the current estimate of the state-value function for each of the TD learning procedures. In the second row, we show the eligibility of each state at that specific time step, for each of the TD learning procedures (TD(0) in blue, TD($\lambda = 0.3$) in red). The upper right corner of the screenshot shows specific information about the current state of the simulation. The lower right corner of the screenshot shows the RMS error for each of the 10 episodes, averaged over all training sets seen up until that specific moment.

The animation shows experiment 2 protocol, a single run of each training set, for the 100 training sets, for 2 TD learning procedures in parallel. The first, showed in the first column of charts, is a simulation of TD(0). The second, in the second column, simulates TD($\lambda = 0.3$). Both simulations run in parallel, using exactly the same data and same learning rate (0.2). We used colors to see how each state-value estimate is changed at each time step. The darker the color, the more change it had versus the previous time step. This animation, played back in different speeds, helped us visually confirm Sutton's last point about how TD(0) is slow at propagating predictions.

4 Conclusion

In this work, we implemented TD(λ) reinforcement learning algorithm, replicating Figures 3, 4 and 5 from Sutton's 1998 paper *Learning to Predict by the Methods of Temporal Differences*. Our results matched the results presented in the paper, and we were able to confirm all main points made in the article about the random walk experiments.

References

[1] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, Aug 1988.

[2] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[3] Leslie Pack Kaelbling, Michael L. Littman, and Andrew P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[4] Csaba Szepesvari. *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers, 2010.