Revisiting Temporal Difference Learning: A Replication Study of Sutton's TD(λ) Methods

Abstract

This report revisits Richard S. Sutton's seminal TD(λ) methods by replicating the experiments from his 1988 paper on the random walk prediction problem. We evaluate the robustness and applicability of TD learning through a comparative analysis of supervised learning and TD(λ) strategies. Our findings confirm the efficacy of TD(λ) in learning from temporal differences and adapting to partial information. Adjustments in learning parameters like rate and convergence thresholds highlight their impact on learning outcomes, especially the influence of λ values on prediction accuracy and efficiency. This study supports the foundational principles of TD learning and corroborates its relevance through rigorous empirical validation.

1. Introduction

In his paper "Learning to Predict by the Methods of Temporal Differences", Sutton[1] proposed $TD(\lambda)$, a class of incremental model-free prediction-learning procedures, synthesizing Monte-Carlo learning and previous works on Temporal Difference methods[2]. He further demonstrates their convergence and optimality with respect to the **Widrow-Hoff** approach.

Central to Sutton's discourse are experiments on a random walk prediction problem, showcasing the efficacy of $TD(\lambda)$ methods, particularly in Figures 3, 4, and 5 . The replication of these experiments is crucial for reinforcing the foundational principles of TD learning and examining their applicability in contemporary contexts.

2. Methodology

2.1. Computation Theory of TD methods

Sutton began by distinguishing between single-step and multi-step prediction-learning problems. He concluded that TD is only distinguishable from supervised-learning methods under the latter context, where only partial information related to the correctness of a prediction is revealed at each time step, due to its ability to learn from incomplete information.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

2.1.1. Supervised Learning Approach

Define an observation-outcome sequence:x_1, x_2, x_3, $\ldots$, x_m, z, where x_t denotes the observation at time step t, and z is the terminal outcome.

For each x_t, we can produce prediction:

$$P\left(x_t, w\right) = w^T x_t = \sum_i w(i) x_t(i). \quad (1)$$

as an estimated value of z, where w is a vector of trainable weights, which can be updated after the presentation of a training sequence:

$$w \leftarrow w + \sum_{t=1}^{m} \Delta w_t. \quad (2)$$

For typical supervised learning, we update the weights via gradient descent:

$$\Delta w_t = \alpha \left(z - P_t\right) \nabla_w P_t. \quad (3)$$

where α is the learning rate. We choose the simplest form $P_t = w^T x_t$ and substitute (1) into (3), we obtain the **Widrow-Hoff** rule, which is the supervised-learning baseline we would be comparing the TD methods to:

$$\Delta w_t = \alpha \left(z - w^T x_t\right) x_t. \quad (4)$$

2.1.2. $TD(1)$ Approach

However, we can modify the $z - P_t$ so that it is represented as a sum of temporal differences:

$$z - P_t = \sum_{k=t}^{m} \left(P_{k+1} - P_k\right) \quad \text{where} \quad P_{m+1} \overset{\text{def}}{=} z. \quad (5)$$

Substitute Eq.5 into Eq.3, we obtain

$$\begin{aligned}\Delta w_t &= \sum_{t=1}^{m} \alpha \sum_{k=t}^{m} \left(P_{k+1} - P_k\right) \nabla_w P_t \\ &= \alpha \left(P_{t+1} - P_t\right) \sum_{k=1}^{t} \nabla_w P_k. \end{aligned} \tag{6}$$

Since Δw_t is only dependent on a pair of successive predictions $\{P_t, P_{t+1}\}$ and the sum of past gradients, this procedure can be performed incrementally without the final reveal of z. Therefore, its more computationally and memory-efficient.

We refer to the approach above as $TD(1)$ approach. Since it is simply an alternative calculation method of the same training procedure as Eq.2, we have therefore proven $TD(1)$ produces the same weight changes as the *Widrow-Hoff* approach.

2.1.3. $TD(\lambda)$ APPROACH

Sutton proceeds to further expand on the TD method by emphasizing more on the recent predictions via applying exponential decayed weighting with recency:

$$\Delta w_t = \alpha \left(P_{t+1} - P_t\right) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k, \quad 0 \leq \lambda \leq 1 \tag{7}$$

It can be noted that $TD(1)$ mentioned in 2.1.2 is exactly $TD(\lambda)|_{\lambda=1}$.

On the other hand, if we take $\lambda = 0$, the weight increment is only determined by its most prediction:

$$\Delta w_t = \alpha \left(P_{t+1} - P_t\right) \nabla_w P_t \tag{8}$$

Due to its overall simplicity and formal similarity to Eq.3, Sutton consider it to be an focal point for comparison.

An important advantage to adopting exponential decay is that it could be computed incrementally, $i.e.$

$$\begin{aligned} &\text{def:} \quad e_t = \textstyle\sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k \\ &\text{then:} \quad e_{t+1} = \lambda e_t + \nabla_w P_{t+1} \end{aligned} \tag{9}$$

which, as have been stated, improves computational efficiency.

2.2. Bounded Random Walk

Sutton constructed a simple dynamical system, the bounded random walk, to demonstrate the effectiveness of TD methods.

2.3. Environment Setting

As visualized in figure 1, a bounded random walk is essentially a sequence of unit walks subject to a limited action space, with the start and terminal conditions given. In our case, all walks begin in state D. From states B, C, D, E, and F, the walk has a $50 - 50$chance of moving either to the right or to the left. If either edge state, A or G, is entered, then the walk terminates. We wish to estimate the probabilities of a walk ending in the rightmost state, G, given that it is in each of the other states.

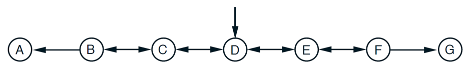


Figure 1. Visualized environment

2.4. Model Construction

We then apply linear supervised-learning and TD methods to this problem.

Firstly, we define the observation-outcome sequence: $x_1, x_2, x_3, \ldots, x_m, z$.

We define x_t based on the corresponding state the agent is in at times step t. For each non-terminal state i, we have a corresponding state vector $\mathbf{x}_i$; if the walk was in state i at time t then our observation $x_t = \mathbf{x}_i$.

In this case, the vectors $\{\mathbf{x}_i\}$ were the unit basis vectors of length 5, one-hot encoded. For example, for state C, $\mathbf{x}_i = [0, 1, 0, 0, 0]^T$.

Since the probabilities $P_{A \to G} = 0, P_{G \to G} = 1$, we set $z_A = 0, z_G = 1$. For the intermediate states, the probability of our agent going either directions is equal. Therefore theoretically, $P_{state} = \frac{P_{left} + P_{right}}{2}$, which indicates the sequence $z_A, P_B, P_C, \ldots P_F, z_G$ is linear, specifically $[0, \frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{5}{6}, 1]$. These are the ideal predictions which we would later use to calculate the RMS-error of our training results.

With the above parametrization, we began the replication of Sutton's results.

3. Experiment Replication

As the syllabus mentioned, we divided the experiment into three questions. We first generate the necessary data on which we would train out agent, the length distribution of all data is shown in figure 2.

To replicate figure $3 - 5$, we conducted 3 experiments correspondingly. For Experiment 1, the following two points need to be noted. w is to be updated only after the complete presentation of a training set, and each training set was presented repeatedly until convergence. The experimental results can be found in section 4.

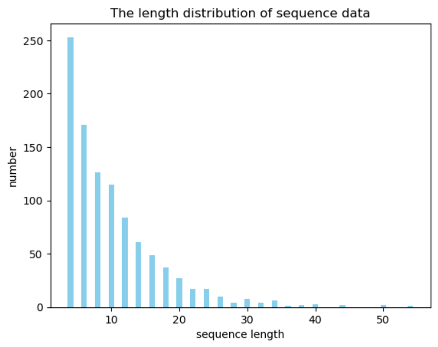


Figure 2. Distribution of generated sequence lengths

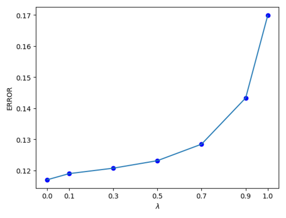
For experiment 2, the following three points need to be noted. The same data with experiment 1 will be used but each training set was presented once to each procedure. Weight updates now are performed after each sequence. And all components of the weight vector were initially set to 0.5.

In experiment 3, all the pre-related work has been completed. We calculated each lambda under the optimal alpha and find the best lambda using function train_2() . The experimental results and image display can be found in section 4.

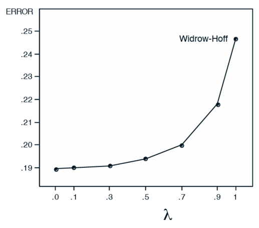
The corresponding pseudocode for our experiments are presented at section 8.1.

4. Results and Comparison

In this experiment, we obtained results similar to those in Sutton's paper. As shown in the figure below, Experiment 1 compares our results (figure 3(a)) with Sutton's (figure 3(b)).



(a)



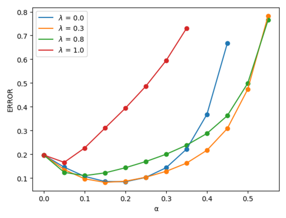
(b)

Figure 3. (a) Our results (b) Sutton's results

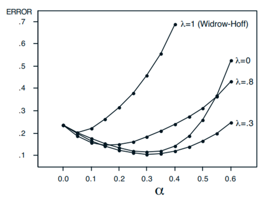
The trends of the two graphs are similar. As λ falls below 1, performance swiftly improves until it peaks when λ reaches 0. Contrary to Sutton, our experimental results show a smaller RMS error. With deliberation, we considered the following factors.

Since the experimental results are greatly influenced by the choice of learning rate and convergence threshold. Different choices of convergence thresholds may affect the experimental results. Additonally, our weight initialization method varies from Sutton's, as Sutton didn't specify the weight initialization method for Experiment 1 in the paper. Without loss of generality, we employ the np.random.uniform() method to initialize weights through uniformly distributed random sampling, which may also cause a different result.

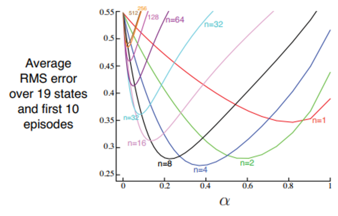
The n-step TD methods on random walk are introduced in chapter7.1 of Sutton's book, and the random walk problem is the same as that in this experiment. Different from n-step TD methods, $TD(\lambda)$is equivalent to the sum of different n results in n-step TD methods weighted by the weight of $\lambda^{n-1}(1-\lambda)$. When n is 1, the method is equivalent to $TD(0)$. When n goes to positive infinity, the method is equivalent to $TD(1)$. For $TD(\lambda)$, with smaller λ, the experimental result should closely align with the case where n has a smaller value. This translates to the elbow of the curve shifting from left to right. Figure 4 showcases Experiment 2. It provides a comparison between our method, Sutton's paper, and Chapter 7.1.



(a)



(b)



(c)

Figure 4.
(a) Our results (b) Sutton's results (c) Chapter 7.1's results

Our experimental results show that different values of λ correspond well with the trend of different n values, as seen in the right figure. As λ increases, the elbow of the rising curve shifts to the left, aligning well with Sutton's results. And it also demonstrates that the performance of the results improves with the decrease of λ, reaching an optimal value at 0.3. Beyond this point, as λ continues to decrease, the results gradually deviate from the optimal value.

The results of Experiment 3 are illustrated in the figure below. Figure 5(a) represents our results, while Sutton's results are depicted on figure 5(b). Both figures show similar trends. Both evident that the optimal value for λ is approximately 0.3 under the optimal α, aligning with Sutton's findings, which also remains consistent with the results from Experiment 2.

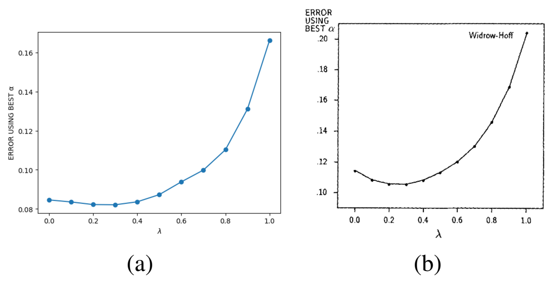


Figure 5. (a) Our results (b) Sutton's results

5. Discussion

In this section, we mainly discuss the effect of parameter λ and why $\lambda = 1$ has better performance than $\lambda = 0$. We first recap the formula for λ-return:

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t \tag{10}$$

It's obvious that when $\lambda = 1$, the first term will be eliminated, transforming the formula into a standard MC return with only the latter term preserved. In contrast, when $\lambda = 0$, only the term $G_{t:t+1}$ remains, which can be seen as a simplified $TD(\lambda)$ with a single step information.

There's a common flaw for MC methods in actual implementation, which is the high variance problem given the sampled trajectories from the environment. In contrast, single-step TD learning doesn't have such disadvantage, by avoiding the computation for the whole trajectory. Therefore, it's not surprising that $\lambda = 0$ outperforms $\lambda = 1$.

Additionally, we can give the proof regarding the optimality for $\lambda = 0$. We can express $G_{t:t+1}$ as:

$$G_{t:t+1} = R_t + \gamma V(s_{t+1}) \tag{11}$$

The $TD(\lambda)$ is reduced to a temporal difference learning with the Bellman error:

$$\mathscr{L} = \mathbb{E}_{s,r,s'}(R_t + \gamma V(s_{t+1}) - V(s_t))^2 \tag{12}$$

Satisfying this error, the algorithm's optimality can be proved by the optimality of the Bellman equation [3].

6. Conclusion

6.1. Reflection on the Replication Process and Discussions

In our replication of Sutton's experiments, we accomplished a thorough understanding of TD(λ) methods and their application in the classic experimental environment, "bounded random walk". Our results, while slightly different from Sutton's, potentially due to the randomness and different initial conditions, were consistent with the primary conclusions of his paper. The TD(λ) methods indeed outperformed the *Widrow-Hoff* approach, especially for smaller λ.

Through our research we have encountered numerous pitfalls. For instance, during the replication of figure 3, we had trouble understanding when to update weights w and stop repeatedly present a given training set. In Sutton's paper the terminal condition is described as "when the procedure no longer produced any significant changes in the weight vector", therefore we introduced a threshold on the Euclidean norm $||\Delta w||$ to determine its convergence. This method proves to be effective and enables a decent replication of the original figure.

In addition, we were not able to make the model converge within appropriate time initially due to poor selection of the hyperparameters "learning rate α" and "convergence threshold ϵ". To elaborate, large α and small ϵ would cause $||\Delta w||$ to vibrate around the threshold, resulting in the algorithm failing to converge. We eventually settled on $\alpha = 0.01, \epsilon = 0.001$, such that we minimize the error within $(1.8, 4)$s without preventing the model's convergence.

During the process, we also figured out it is necessary to re-initialize w for every batch of training data, otherwise the RMS metric wouldn't have been justified, in the sense that it is supposed to be "averaged over training sets".

Lastly, although trivial, we tweaked the plotting function of figure 4 so it cuts out the errors greater than 0.80.8. It is justified by the fact that without doing so the figure would lose proportion due to a few outlier errors being exponentially larger than the rest and dominating the frame. If we were to reflect on the original figure, it is apparent that Sutton adopted a similar approach due to the same reason.

Proceeding the experiments, we discussed the impact of parameter λ and provided a compelling argument on why $\lambda = 0$ has better performance than $\lambda = 1$. The high variance issue in Monte Carlo methods and the optimality of Bellman equation supported our arguments.

This replication process and ensuing discussions have not only validated the original findings of Sutton's paper but also deepened our understanding of the fundamental principles and intricacies involved in reinforcement learning.

6.2. Final Remarks

To sum up, our replication of Sutton's $TD(\lambda)$ methods serves as a testament to the efficacy and relevance of these methods in temporal prediction problems. Despite minor discrepancies, our results aligned with Sutton's primary findings and reinforced the supremacy of $TD(\lambda)$ methods over traditional supervised learning approaches in reinforcement learning tasks.

7. Reference

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

[2] Samuel, Arthur L. "Some studies in machine learning using the game of checkers." *IBM Journal of research and development* 44.1.2 (2000): 206-226.

[3] Alekh A. and Nan J. and Sham M. K. and Wen S. "Reinforcement Learning: Theory and Algorithms" 2021.rl

[4] Richard S. Sutton and Andrew G. Barto. "Reinforcement Learning: An Introduction", MIT Press, Cambridge, MA,2018.

8. Appendix

8.1. Pseudocode

Below are a list of pseudocode for the algorithms mentioned in the paper.

Algorithm 1 Data Generation

```
output : data
data ← []
for $i \leftarrow 0$ to 99 do
    batch ← []
    for $j \leftarrow 0$ to 9 do
        pos ← 2
        sequence← []
        while pos $\neq$ -1 or pos $\neq$ 5 do
            action ← random choice 1 or -1
            pos ← pos + action
            sequence.append(pos)
        end
        batch.append(sequence)
    end
    data.append(batch)
end
```

Algorithm 2 update_weights function

```
input  : $w$, current_pos, next_pos, alpha, lamb, e
output : delta_$w$, e
Initialize delta_e as onehot posstion encoding by current_pos
 e ← lamb × e + delta_e
 delta_$w$ ← alpha × (state_value($w$, next_pos) − state_value($w$, current_pos)) × e
```

Algorithm 3 train

```
input  : data, alpha, lamb,convergence_threshold
output : err
Initialize: err ← 0
for batch $\in$ data do
    $w$ ← randomly and uniformly choose by $1 \times 5$ while True do
        delta_$w$ ← $[0] \times 5$
        for sequence $\in$ batch do
            e ← 0 for idx of batch do
                delta_$w$, e ←update_weight($w$, sequence[idx], sequence[idx+1], alpha, lamb, e)
            end
        end
        $w \leftarrow w$ + delta_$w$
        if norm(delta_$w$) <convergence_threshold then
            err ← err + rms(batch,$w$,P_ideal) break
        else
    end
end
err ← err / len(data)
```

Algorithm 4 train2

```
input  : data, alpha, lamb
output : err
Initialize: err ← 0
for batch $\in$ data do
    $w \leftarrow [0.5] \times 5$
    for sequence in batch do
        delta_$w$ ← $[0] \times 5$
        e ← 0
        for idx of batch do
            delta_$w$, e ←update_weight($w$, sequence[idx], sequence[idx+1], alpha, lamb, e)
        end
        $w \leftarrow w$ + delta_$w$
    end
    err ← err+rms(batch, $w$, P_ideal)
end
err ← err / len(data)
```