CS7642 Project 1 Report – Desperately Seeking Sutton

Minghan Xu (minghan.xu@gatech.edu)
4835951e59f403e3273710bcc7b30363c5586d2f (Git hash)

***Abstract*— This project report discuss the fundamentals of Temporal Difference (TD) Learning and provides details of steps in reproducing experiment results in Sutton's paper "Learning to Predict by the Methods of Temporal Differences" using the bounded random-walk process**

I. INTRODUCTION

In the problem of making predictions using past experience without the complete knowledge of the system, conventional method involves supervised learning which pairs input features and target output variable. The learners are expected to associate such paired items first (aka. the training process), then recall the target variable when presented with only input features. Sutton proposed the concept of Temporal Difference (TD) learning: instead of pairing up observations with the target variable, TD takes the temporal relationships between states into the considerations and pairs up the consecutive predictions in the training sequence. This learning mechanism is driven by the errors between temporally successive predictions rather than predicted vs. actuals in the supervised learning approach. Hence TD approach has the advantages of lower requirements in computation & memory, faster convergence and more accurate predictions.

II. RANDOM-WALK

A. Setup of the random-walk

Sutton proposed a very simple stochastic system where the states can be observed over time and illustrated that TD methods have more efficient learning than supervised learning (Widrow-Hoff).

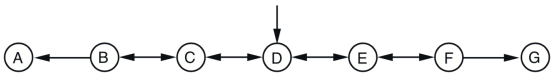

Fig. 1. The generator of bounded random-walks

Illustrated in figure 1, the random-walk process always starts with the initial center state D. For each state transition, the walk moves to its neighboring state (left or right) with equal probability(50 vs. 50). If terminal state A or G is entered, the random-walk ends. When random-walk ends at state A, the reward $z = 0$; When it ends at state G, the reward $z = 1$.

B. Implementation of random-walk

In the settings of the bounded random-walk, there are two types of states:

- Non-terminal states: B,C,D,E,F
- Terminal states: A,G

Vectorized representation (one-hot encoding) of non-terminal states is adopted. Each state is represented as a column vector with a dimension of (5×1). For example, state D's representation is, $X_D = (0, 0, 1, 0, 0)^T$. For terminal state A and G, they are not one-hot encoded but rather associated with a reward of $z = 0$ or $z = 1$.

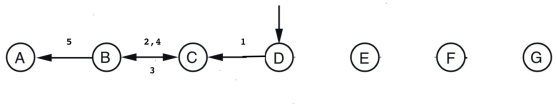

Fig. 2. A random-walk sequence $X_D, X_C, X_B, X_C, X_B, X_A$

For a complete single instance of random-walk sequence, it is represented by column stacking each individual non-terminal state in the sequence. For a sequence illustrated in figure 2 with states $X_D, X_C, X_B, X_C, X_B, X_A$, it is represented with reward $z = 0$ and non-terminal state sequence as

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The algorithm 1 illustrates the bounded random-walk sequence generation. The generator keeps appending non-terminal states until state A or G is reached. The function returns the vectorized non-terminal states sequence and the reward associated with the terminal state.

III. TEMPORAL DIFFERENCE (TD) LEARNING

Sutton's paper illustrates the difference of two types of learning-prediction problems: 1) single-step prediction 2) multi-step prediction. As the single-step prediction problem does not have temporal relationships in which TD can take advantage of, this report will sorely focus on the discussion of multi-step prediction problem using TD.

At the high-level, TD offers two benefits over the conventional supervised learning approach: 1) enable incremental

Algorithm 1: Random-Walk Sequence Generation

Input: None
Output: Numpy vectorized representation of a random-walk sequence seq_vec, $reward$

Init $seq\_vec, reward$
while *current_state not in [A,G]* **do**
  $next\_state = get\_neighbor()$
  if *next_state not in [A,G]* **then**
    seq_vec append $next\_state$
  else if $next\_state == G$ **then**
    $reward = 1$
  else
    $current\_state = next\_state$
  end
end
return $seq\_vec, reward$

computation which lowers system requirements on peak computation and memory consumption 2) speed up the learning and generate more accurate results.

A. Supervised Learning

For a multi-step prediction problem in the form of observation-outcome sequence: $x_1, x_2, x_3, ..., x_m, z$, where x_t is a vector state/observation at time t and z is a scalar value of the reward/outcome, The prediction is a sequence of estimation of z value associated with each state: $P_1, P_2, P_3, ...P_m$, where P_t is a function of all proceeding states including the current state: $x_1, x_2, ...x_{t-1}, x_t$.

For simplicity, Sutton suggest a special case where the prediction at time t is a function of current state x_t and some learnable weights w: $P(x_t, w)$, where $w(i)$ and $x_t(i)$ are the ith dimention of w and x_t:

$$P_t = w^T x_t = \sum_i w(i) x_t(i)$$

The weights update/learning procedure is derived as below where α is the learning rate:

$$\Delta w_t = \alpha(z - P_t)\nabla_w P_t = \alpha(z - w^T x_t)x_t$$

For supervised learning schema, vector weights w is only updated after a complete observation-outcome sequence:

$$w_{new} = w_{old} + \sum_{t=1}^{m} \Delta w_t$$

In short, the supervised learning starts with some initial weights w. For each prediction at time t, it compares P_t with the actual reward z and calculates the correction Δw needed. The needed correction is accumulated across the states till sequence termination. Then an update is performed to generate the new weights, such process is usually repeated until convergence. The learned weights hence estimate the intrinsic values of non-terminal state respect to the final reward.

B. TD & Incremental Learning

For the same multi-step prediction problem earlier, Temporal Difference (TD) Learning breaks down the difference between P_t and z into cumulative differences between consecutive predictions where $P_{m+1} \equiv z$:

$$z - P_t = \sum_{k=t}^{m} (P_{k+1} - P_k)$$

For TD, The weights update/learning procedure can then be expressed as:

$$w_{new} = w_{old} + \sum_{t=1}^{m} \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \nabla_w P_k$$

$$\Delta w_t = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \nabla_w P_k$$

Such breakdown brings in few benefits compared to the supervised learning schema. In the supervised learning, the learner has to remember all the non-terminal states and perform updates after a complete sequence. For TD, as the weight correction Δw_t at non-terminal states does not depend on the z value, the learning can take place in a incremental fashion. Hence the learner only needs to remember most recent states and perform the updates on weights. This could significantly reduce the memory requirements and speed up the learning. Sutton paper has also proved that supervised learning and TD(1) produces the same learning outcome.

C. TD(λ) Learning

In the previous section, TD(1) is discussed and it has been proved in Sutton's paper of generating the same learning outcomes as Widrow-Hoff approach. However, TD(1) is a special case of the temporal difference learning paradigm, where each prediction P_t carries the same weights in the learning process as t changes. A more general case of TD(λ) proposes that more recent predictions shall carry heavier weights in the updating process of w. The coefficients of these prediction are exponentially weighted with recency as follows where λ is a coefficient between 0 and 1:

$$\Delta w_t = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k$$

The error term e is defined and can be incrementally computed as:

$$e_{t+1} = \sum_{k=1}^{t+1} \lambda^{t+1-k} \nabla_w P_k = \nabla_w P_{t+1} + \lambda e_t$$

The consideration on the recency of prediction could improve the performance of regular TD(1) in terms of accuracy and convergence speed.

IV. Experiments & Results

In this section, the reproduction of figure 3-5 in Sutton's paper will be illustrated in technical details. First, the theoretical values/ideal predictions of non-terminal states in the random walk problem are calculated using the method suggested in the paper. Two experiments are then conducted with the first one focusing on the repeated presentation of training sequences using different values of λ and the second focusing on the single presentation of training sequences with unbiased initialization and different combinations of lambda & learning rate pairs.

A. Theoretical Values of Weights in Random-Walk

Again to understand what the weight $w(i)$ represents for non-terminal state sequences, it is the expected value of the outcome z given that the state sequence is starting in i.

For the bounded random-walk example, Q is the transition probability matrix where Q_{ij} is the transition probability between non-terminal states i and j, and h is the transition probability vector between non-terminal states and terminal state (G). The ideal weight of $w(i)$ can be calculated as follows:

$$E\{z|i\} = [\sum_{k=0}^{\inf} Q^k h]_i = [(I-Q)^{-1}h]_i$$

The vector form of ideal weights (true values) for non-terminal states B,C,D,E,F is derived as:

$$E(z) = [(I-Q)^{-1}h]$$

$$= (I - \begin{bmatrix} 0 & 0.5 & 0 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0 & 0.5 & 0 \\ 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 0.5 & 0 \end{bmatrix})^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 1/6 \\ 1/3 \\ 1/2 \\ 2/3 \\ 5/6 \end{bmatrix}$$

B. Repeated Presentations Paradigm

100 randomly generated training sets with each comprising 10 bounded random-walk sequences are prepared using the techniques covered in part II of this report.

In the repeated presentations paradigm, for a specific λ value, the learner is repeatedly presented with 10 randomly generated random-walk sequences in a training set until convergence. The learner accumulates Δw across these 10 sequences and only perform the weight update after a full epoch of training. The above process is repeated for 100 training sets and computes the average root mean squared error (rmse) between the learned weights and ideal weights of non-terminal states: X_B, X_C, X_D, X_E, X_F.

The reproduced figure 3 demonstrates the average error between learned weights w and ideal weights of $[\frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{5}{6}]^T$ with different values of λ. It shows that TD(1) (Widrow-Hoff) performs the worst compared to other lambda values. The explanations to this observation could be that TD(1) process minimizes the error between predictions and actuals in the training sets. However it does not guarantee the better results for future predictions. TD(0) has the lowest mean error as it concerns only the most recent state and such result is consistent with the maximum-likelihood estimation (MLE) for the Markov process.

Algorithm 2: TD(λ) with repeated presentations

Input: Python lists of λ values
Output: Python dictionary stores the mean rmse for each λ: $rmse\_dict$

```
Init list of λ, learning rate α, stopping criteria ϵ
for each λ do
    for each training set do
        Init weights w
        while changes in w < ϵ do
            for each training sequence do
                Init error err
                for each state do
                    calculate err and Δw
                end
                accumulate Δw across sequences
            end
            update w using accumulated Δw
        end
        calculate rmse for the current training set
    end
    calculate mean rmse across all training sets
end
```

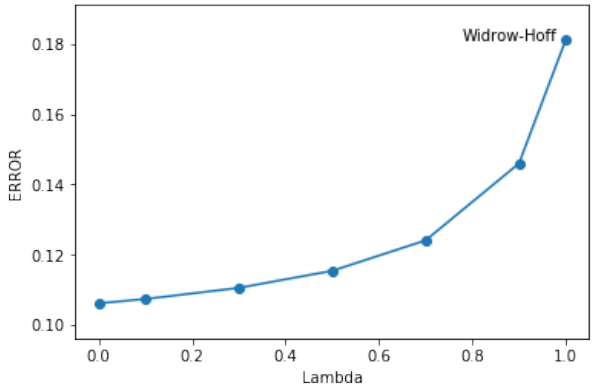

Fig. 3. Average errors on the random-walk with repeated presentations

The reproduced result well aligns with the figure 3 in Sutton's report, though exact values of errors are lower than those in the original report. This could be due to the different randomly generated training sets. Weight w initialization and learning rate α can also be the source of such difference, as Sutton does not specify them when producing figure 3. In this reproduction, $\alpha = 0.01$ and unbiased weight w initialization are used to ensure the convergence of the TD learning.

C. Single Presentation Paradigm

The same 100 randomly generated training sets are used in the second experiment. The weights are unbiased initialized as 0.5 for all non-terminal states

In the single presentations paradigm, for a specific com-

bination of (λ, α) values, the learner can only learn the 10 random-walk sequences for a single time in a training set. Instead of performing weight update after one full epoch of training, the learner updates the weights w at the end of each random-walk sequence. The above process is repeated for 100 training sets and computes the average root mean squared error (rmse) between the learned weights and ideal weights. Combinations of λ and α are tested to analyze the impacts of learning rate on learning outcomes.

Algorithm 3: TD(λ) with single presentation

Input: Python lists of λ and α values
Output: Python dictionary stores the mean rmse for each (λ, α) pair: $rmse\_fig\_4$

```
Init list of λ, list of α, learning rate α
for each λ do
    for each α do
        for each training set do
            Init weights w
            for each training sequence do
                Init error err
                for each state do
                    calculate err and Δw
                end
                update w using Δw
            end
            calculate rmse for the current training set
        end
        calculate mean rmse across all training sets
    end
end
```

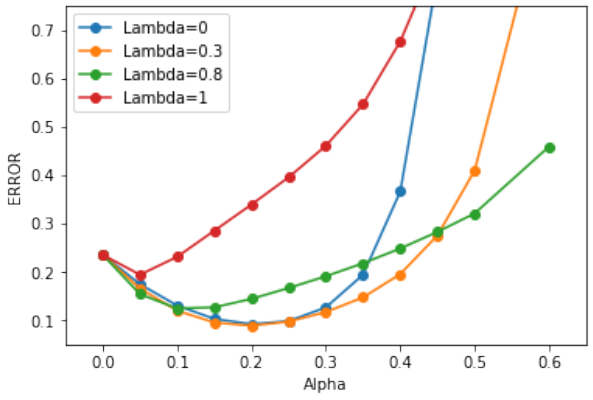

Fig. 4. Average errors on the random-walk with single presentation for $\lambda = 0, 0.3, 0.8, 1$

The reproduced figure 4 demonstrates the average error between the learned weights w and ideal weights of $[\frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{5}{6}]^T$ using different combinations of λ and α. Each line series in the graph represents the mean error given a specific λ across different learning rate α. The learning rate has a significant impact on the TD learning outcome. Lower α values (around 0.2 to 0.3) yield the predictions with the lowest errors. This could be explained that a well-balanced learning rate like 0.2 can achieve good rate of convergence and the quality of the weight update at the same time. Again, the red line (TD(1)) produces the worst results regardless of α values used. This re-illustrates the limitation of conventional TD(1) (Widrow-Hoff) approach. In the bounded random-walk problem, TD($\lambda < 1$) generates better predictions.

The reproduced result aligns with the general patterns in figure 4 of Sutton's paper and the errors are also distributed in the same range. However, the lines of $\lambda = 0$ and $\lambda = 0.8$ intersect at a lower value of α when comparing with the original graph. This along with the intersection between $\lambda = 0.3$ and $\lambda = 0.8$ might still be due to the different randomly generated training sets used in the learning process.

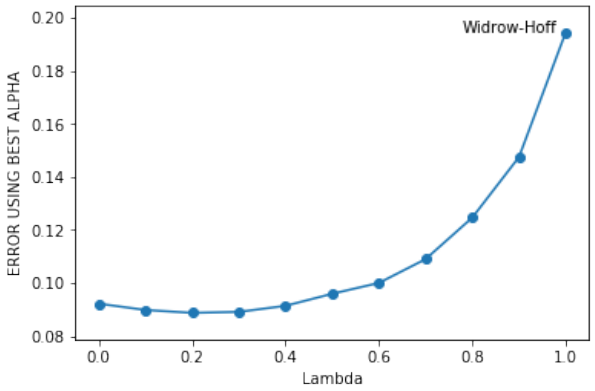

Fig. 5. Average errors at best α on the random-walk problem with single presentation for λ

The reproduced figure 5 demonstrates the best mean errors achieved in each λ value, which is a more condensed version of figure 4. From the graph, it shows again TD(1) gives the poorest predictions. The best λ in the experiment of single presentation is around 0.3. Similarly, the reproduced result closely resemble the original figure 5.

V. CONCLUSIONS

The author of this report has walked through in details on the setup of the bounded random-walk process. The implementations of generating training sets are illustrated in Section II. In Section III, the author synthesized the mathematical derivations in Sutton's paper and explained three types of learning schema: Supervised Learning (Widrow-Hoff), Incremental Learning (TD(1)) and the more generic TD(λ). Section IV discusses the algorithms implemented to reproduce figure 3-5 in Sutton's paper. Two learning paradigms: repeated presentations & single presentation have been elaborated along with results and their interpretations. The research process of paper reading, code design & implementation and result analysis has helped the author to acquire more concrete understanding of Temporal Difference (TD) Learning and comprehend the impacts from hyperparameters α and λ on the prediction outcomes. To achieve

the best results in future implementations of TD learning, the prediction problems shall be carefully evaluated (single-step vs. multi-step), a balanced learning rate to be selected to ensure convergence and speed. Last but not least, a good λ value shall be adopted based on the problem settings and leverages the strengths of TD which include incremental updates and better accuracy of the predictions.

REFERENCES

[1] R. S. Sutton, Learning to predict by the methods of temporal differences, Machine Learning, vol. 3, no. 1, pp. 944, 1988.

[2] G. Tesauro, Practical Issues in Temporal Difference Learning, Reinforcement Learning, pp. 3353, 1992.

[3] G. Tesauro, TD-Gammon: A Self-Teaching Backgammon Program, Applications of Neural Networks, pp. 267285, 1995.

[4] J. Tsitsiklis and B. V. Roy, An analysis of temporal-difference learning with function approximation, IEEE Transactions on Automatic Control, vol. 42, no. 5, pp. 674690, 1997.