Project 1 Report

Yihuan Su

Git hash: 440b2013ab97d0e49ea70394bce403433778e1f3

***Abstract*—This report talks about the fundamental concepts of temporal difference learning based on Richard Sutton's paper (Sutton 1988). Replication of the random walk experiment in section 3.2 of the paper is attempted. The results are discussed in this report.**

I Introduction

Sutton describes the problem of learning to predict as "using past experience with an incompletely known system to predict its future behavior" (Sutton 1988). Examples of this type of problem are: whether a customer will churn, how long will it take to drive from home to work, whether a team will win a DOTA game for a given state of the game. Supervised learning has been the main conventional learning paradigm for this type of problem. It is driven by the error between prediciton and actual final outcomes. It's most suitable for signle-step problems, of which the outcomes are revealed at once, because it assumes obversations and outcomes come in pairs. For a multi-step problem, of which the outcomes aren't revealed until more than one step after prediciton is made, supervised learning methods disregard sequential structures of the problem due to it's pairwise approach.

In his paper, Sutton (1988) formally introduced temporal difference (TD) learning which is a class of incremental learning method that's driven by the difference between temporally successive predictions. With this methods, learning doesn't have to wait for final outcome, it can happen over time whenever there is change in predction. For example, we are trying to predict every month, whether a mortgage borrower will face forclosure this year. The supervised learning method will have to wait for the forclosure to happen or till the end of year in case the event never happen in order to learn. TD method compares predciton of each month with prediction made the next month. If a borrower missed one payment in March, we know his chanced of getting foreclosed increased from 0.1% to 1%, instead of waiting for the final outcome, we can increase Janurary and Febraury's prediciton right away.

Sutton (1988) aruges TD aproach possesses two adavntages compared to supervised learning methods when dealing with multi-step problems. 1. it's easier to compute due to it's incremental nature. 2. it tends to be more efficient at using data. i.e. it makes more accurate predictions and converges faster.

II Temporal-difference learning

Referencing Sutton's paper, in this seciton, I discuss the fundamental concepts of TD learning and draw comparison between TD learning and conventional supervised-learning.

II.A Single-step and multi-step prediction

To understand temporal-difference learning, it's important to distinguish two types of prediction-learning problems: single-step learning and multi-step learning. The main difference between the two is when is the outcome revealed. In single-step problem, the outcome is revealed at once. In multi-step problem, it takes more than one step for the outcome to be revealed, but at each step, additional information related to the correctness of prediction is revealed. Consider the example of a mortgage borrower, in March we still don't know if the borrower will face foclosure this year. But we know he/she just missed a payment and that information tells us there is higher chance for forclosure to happen. On the other hand, if the goal is to prediction if a borrower will face foreclosure each month given all the information we know the month before, then this constitutes a single-step problem, assuming the data is of monthly increments.

Because of its pairwise approach, supervised-learning is well suited for single-step problems as the data comes in observation-learning pairs naturally. For multi-step problems, by pairing observation of one step and final outcome, supervised-learning approach is still applicable, but it ignores all the information from steps between the observation and final outcome during learning. For sigle-step problems, there is no distinguishable difference between temporal-difference methods and supervised learning approach. However, temporal-difference approach shines in multi-step problems because it can learn from information from each step. As Sutton (1988) states, we will be focused on multi-step scenarios here.

II.B Supervised-learning and TD(λ)

Consider a multi-step prediction problem in which data comes in every step in the form of $x_1, x_2, x_3 ... x_m, z$, where x_t is a vector of observed state at time t, and z is the outcome. A sequence of predicitons/values $P_1, P_2, P_3, ..., P_m$ is produced by the learner for each sequence of obversations, where P_t is an

estimate of z at time t. For simplicity, we assume P_t is a function of only x_t and weights w as Sutton (1988) suggests. It can be written as $P(x_t, w)$. Weights are modifiable, and all learning procedures are performed to updated w. As for now, it's assumed that w is updated only once for each complete sequence. The updating rule for w can be written as:

$$w \leftarrow w + \sum_{t=1}^{m} \Delta w_t$$

where Δw_t is an incremental change to w that are determined based on the observation at step t.

With supervised-learning approach, each observation in the sequence is paired with the final outcome. The error used to determine the incremental change is between P_t and z. Typically written as:

$$\Delta w_t = \alpha(z - P_t) \bigtriangledown_w P_t$$

where $0 < \alpha <= 1$ is the learning rate, and $\bigtriangledown_w P_t$ is the partial derivative of P_t with respect of each element of weight vector w.

Assuming P_t is a linear function of x_t and w, that is $P_t = w^T x_t$. Then $\bigtriangledown_w P_t = x_t$. So we have

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

which is known as the Widrow-Hoff rule (Sutton 1988). One can see that Δw_t is dependent on the final outcome z. Without the knowledge of z, the supervised-learning approach simply can not learn.

The key of TD methods is to represent the error $z - P_t$ as a sum of changes in predictions, which can be written as

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

where P_{m+1} is z. So we have

$$\begin{aligned} w \leftarrow w + \sum_{t=1}^{m} \alpha(z - P_t)\nabla_w P_t &= w + \sum_{t=1}^{m} \alpha \sum_{k=t}^{m} (P_{k+1} - P_k)\nabla_w P_t \\ &= w + \sum_{k=1}^{m} \alpha \sum_{t=1}^{k} (P_{k+1} - P_k)\nabla_w P_t \\ &= w + \sum_{t=1}^{m} \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \nabla_w P_k \end{aligned}$$

That is

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

This procedure is referred by Sutton (1988) as TD(1). Based on Sutton's explaination, we've shown that the TD procedure can produces exactly the same result as supervised-learning procedure. As a matter of fact when P_t is a linear function of x_t and w, TD(1) produces the same weight changes as Widrow-Hoff rule (Sutton 1988). However, TD procedure can be computed incrementally since each Δw_t is only dependent on the a pair of successive predictions P_t and P_{t+1}, and on the sum of $\bigtriangledown_w P_k \ \forall \ 1 <= k <= t$. According to Sutton (1988), compared to the supervised-learning procedure, this procedure require less memory and computation resource which are very limited 32 year ago.

What really makes temporal-difference approach unique is it's sensitivity to changes in consecutive predictions. An increment Δw_t is determined based on the difference betweenP_t and P_{t+1}, and predictions for some or all of the predceding obversations $x_1 \ldots x_t$ are updated accordingly. TD(1) is special case in which all of those predictions are updated with the same magnitude. TD(λ), as Sutton (1988) refers, induces greater adjustment to more recent predictions. In particular, it adopts an exponential weighting with recency. The adjustment to the predictions of obvervation occurred k steps before are weighted according to λ^k for $0 <= \lambda <= 1$. The increment change is now written as:

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

Note when λ=1, this becomes TD(1) precedure, which produces the weight changes as supervised-learning procedure. When λ<1, TD(λ) produces different weight changes. When λ=0, the difference is the greatest as the incremental change is determined by the prediciton of the most recent observation alone. That is:

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

Sutton (1988) also shows that, although past predictions can be weighted in other ways, the exponential form has an important advantage: it's possible to be computed incrementally.

III RANDOM WALK EXAMPLE

To demonstratet that TD methods are more efficient at using past experience, which implies faster convergence and better accuracy, than supervised-learning approach as claimed, Sutton (1988) conducted experiments and discussed the results. I attempted to replicate the random walk experiment in the paper and discuss my findings in this section.

III.A Bounded random walk

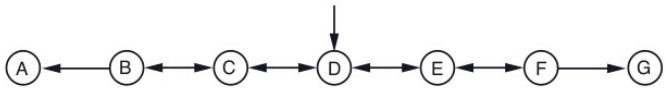


Figure 1. A generator of bounded random walks.

Sutton (1988) define a bounded random walk as 'a state sequence generated by taking random steps to the

right or to the left until a boundary is reached.' Figures 1 shows the system that generate bounded random walk sequences for this experiment. Every sequence starts from state D. At each step, the walk proceeds either to the right or to the left with the same 50% probability. The walk will end If a bundery state (A or G) is reached. The outcome z is defined to be $z = 1$ if the sequence reach G to the right and $z = 0$ if the sequence reach A to the right. Given the definition of outcome, the goal of this experiment is to estimate the expected probaility of reaching state G at each non-terminal state. Linear supervised-learning and TD methods are applied to this problem. The problem is constructed in a way that each non-terminal state i is represented by an obveration vector x_i. If at time t, the sequence state is i, then $x_t = \mathrm{x}_i$. For example, if the walk is DCDEFG, then sequence would be $\mathrm{x}_D, \mathrm{x}_C, \mathrm{x}_D, \mathrm{x}_E, \mathrm{x}_F, 1$. More concretely, the sequence can be presented as ((0,0,1,0,0)^T, (0,1,0,0,0)^T, (0,0,1,0,0)^T, (0,0,0,1,0)^T, (0,0,0,0,1)^T, 1), where each non-terminal state vector is of length 5.

Generating bounded random walks in Python is straight forward. The form I impletemented is slightly different from what Sutton has described. But it serves the same purpose. The following is the pseudo code of my implementation of the generator

Pseudo Code - Bounded Random Walk Generator
Input: none
Return: array of state vectors
Init state=D
Init sequence=[[0,0,1,0,0]]
while state not in [A,G]
    state=random neighbor state
    if state is A
        sequence append 0
    else if state is G
        sequence append 1
    else
        sequence append state vector
return sequence

III.B Experiments and Results

In this section, I discuss my attempt to replication the two conputational experiments that are described by Sutton in the paper. More specifically, I tried to reproduce figure 3, figure 4, and figure 5 in the paper. Following Sutton's description, 100 training sets are constructed with the random walk generator described above. Each one of these training are consisted of 10 sequences. Sutton states with the size of training sets, statistically reliable results should be obtained. However, with today's standard, this appears to be a very small data set. This could also be the reason why I'm not able to create the same graphs shown in the paper. TD(λ) procedures with λ=0, 0.1, 0.3, 0.5, 0.7, 0.9, 1 are applied to incrementally updating weights. As we've discussed, TD(1) results in the Widrow-Hoff supervised-learning procedure.

In the first experiment, instead of updating the weight vector after each sequence, the $\Delta w's$ are accumulated over sequences and the weight wector are not updated until a training set is presented in its entirety. Repeated presentations training paradign were applied. This means each training set was presented repeatedly to the learning procedure until weight vector no longer experience any signficant changes. In this way, Sutton (1988) states that when α is small, the weight vector always converges to the same value. Therefore, in my replication, initial weight vector is randomly generated.

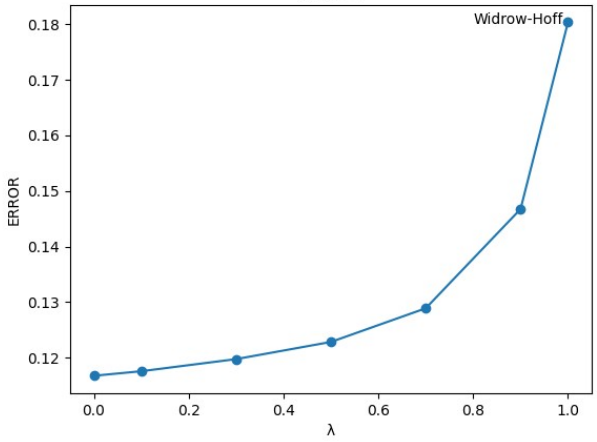
For each of the nonterminal states, the true probabilities of reaching state G are 1/6 for state B, 1/3 for C, 1/2 for state D, 2/3 for state E, and 5/6 for state F. Explaining how to compute the true probabilities is out the scope of this report. Sutton (1988) describe the calculation in detail in section 4.1 of the paper. To measure the performance of a learning procedure, Sutton (1988) selected the average root mean squared (RMS) error between the predictions of a procedure using a training set and the true probabilities over all the training sets. Below is the pseudo code of my impletementation of this experiment:

Pseudo Code – Experiment 1
Input: λ, α, trainning sets, initial w, ε
Return: average RMS
Init total_RMS=0
for each set in trainning sets
    init w=initial w
    Init weight change=1
    while weight change>ε
        Init $\Delta w = [0, 0, 0, 0, 0]$
        for each sequence in set
            for each state in sequence
                calculalte Δw_{state}
                update Δw by adding Δw_{state}
        update w using Δw
        calculate weight change
    calculate RMS
    total_RMS=total_RMS+RMS
average RMS= total_RMS / number of trainning sets
return average RMS

In the paper, Sutton never define what does 'significant changes in the weight vector' mean exactly. In my replication, with ε=0.001, if the maximum absolute value of Δw is less than ε, I'd consider that as no significant changes. Figure 3 is my replication of Figure 3 in Sutton's paper using the same learning rate α=0.01. The pattern matches the figure shown in the paper. The error is at the lowest when λ=0. As the value of λ increases the average RMS error increases gradually at first, but as λ approaches 1, the error increases more rapidly, with TD(1), the supervised learning precedure, having the highest error. Notice in my reproduction of

this experiment, the error is generally lower than what Sutton has. A plausible explaination for that might be today's 64 bit computers have much higher precision than the computers in 1988.

Figure 3



The Widrow-Hoff procedure is known to minimize the RMS error between its predictions and the actual outcomes in the training set. Yet the experiment shows that it performs much worse than TD methods with λ<1. Sutton (1988) explains that's because 'Widrow-Hoff procedure only minimizes error on the training set; it does not necssarily minimize error for the future experience.' In another word, with 10 sequences in each training set, each training set may not represent the true distribution. Widrow-Hoff procedure minimizes error only on the possibly biased training set, so predictions are likely to deviate from the true probabilities.

The second experiment aims to explore how does learning rate impact the performance of learning procedures. This experiment differs from the first experiment in four aspects: 1. each training set is only presented once instead of repeated presentation; 2. instead of updating weight vector after each complete training set, weight vector is updated after each sequence; 3. A range of different values for α was applied to each procedures; 4. the initual weight vector is set to [0.5, 0.5, 0.5, 0.5, 0.5,] to avoid bias towards either rightside or leftside terminations. Here is the pseudo code of my impletementation of this experiment:

<u>Pseudo Code – Experiment 2</u>
Input: λ, α, trainning sets
Return: average RMS
Init total_RMS=0
for each set in trainning sets
    init w=initial w
    for each sequence in set
        Init $\Delta w = [0, 0, 0, 0, 0]$
        for each state in sequence
            calculalte Δw_{state}
            update Δw by adding Δw_{state}
        update w using Δw
    calculate RMS
    total_RMS=total_RMS+RMS
average RMS= total_RMS / number of trainning sets
return average RMS

Figure 4

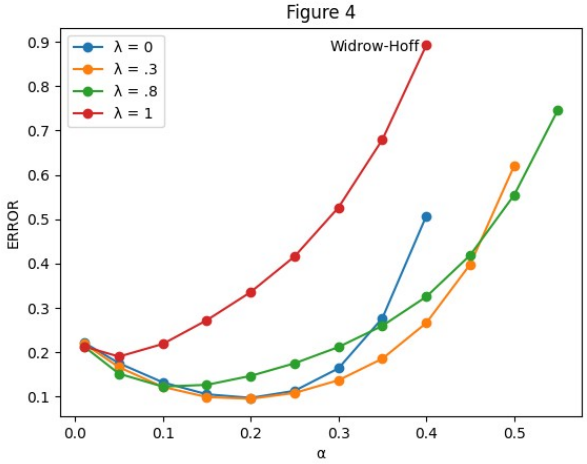
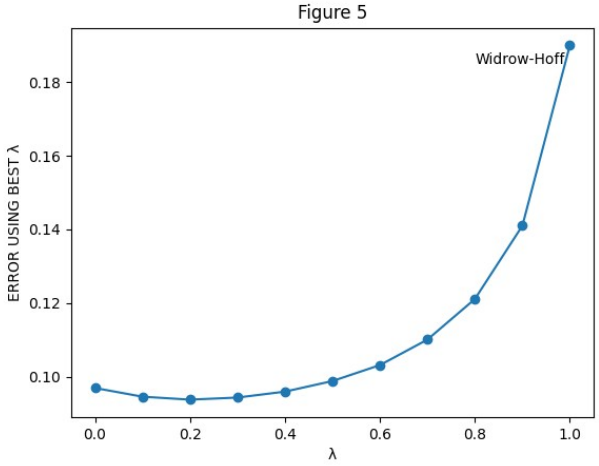


Figure 4 is my attempt to replicate Figure 4 in Sutton's paper. In the paper, Sutton clearly shows the λ values applied are 0, 0.3, 0.8, and 1. However, he doesn't specify what values for α is used. Looking at the Figure 4 in the paper, it's a reasonable guess that α could be 0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, and 0.6. As one can see, my Figure 4 doesn't match Sutton's figure exactly due to the randomness in sequences generated and the size of the training data. Nevertheless, the general pattern is the same. Different values of α certainly have big impact on the performance of the procedures. For all α values, TD(1) performs the worst. For both λ=0 and 0.3, the error is lowest when α=0.2. While it's hard tell from the graph, but TD(0.3) performances just slightly better.

Figure 5



Figures 5 is a replication of the same graph in Sutton's paper. It shows the error of TD methods with different values for λ using the best learning rate α.

Similar to Figure 4, Sutton does specifically call out the values used for λ, one can make reasonable guess based on the graph. I used λ=0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, and 1. My graph again looks slightly different from Sutton's plot in the paper. It's may due to randomness in the data, higher precision, and it may also because I possibly experimented α values at a higher granularity (50 difference α values ranging from 0.01 to 0.5.) which may not be feasible in 1988. The plot again shows every procedure with λ<1 performance better han TD(1). It also suggests that the best λ seems to be around 0.2 and 0.3.

Note the first experiment shows the TD procedure performs the best when λ=0, yet Figure 5 tell a different story. Sutton (1988) explains that's because TD(0) propagates prediction levels back a sequence at slower rate. In another word, TD(0) only updates the prediction of the state of one prior step, whereas other procedures will change predictions of states of all prior steps. With repeated presentaton, TD(0) will eventually converge, but with just one presentation, it learns slower than other procedures.

IV Conclusion

Based on Sutton's 1988 paper, this report starts by introducing the problem of learning to predict and TD learning approach. As well as how TD learning methods might be superior compared to conventional supervised learning approach at dealing with multistep problems. Then I covered the fundamental concepts of TD learning procedures and draw comparison to supervised learning methods. In the later section of the report, we discussed the random walk example in the original paper (1988), and presented the results of the reproduction of the experiments. The results are consistent with the results of Sutton's findings. We conclude that the supervised learning procedure performs worse than TD methods with λ<1 in all scenarios. We also find that learning rate has a significant impact on the performance of a precedure.

References

1 *Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. Machine Learning, 3(1), 9-44. doi:10.1007/bf00115009*