CS7642 Project 1 report

Qinghui Ge

Abstract—**In this report, I demonstrate my efforts in reproducing some of the results regarding $TD(\lambda)$ algorithm in Sutton's paper: *Learning to Predict by the Methods of Temporal Differences* (1988) [1]. I will explain in detail about my implementation of two experiments presented in the original paper, and the discuss the similarity and discrepancy in my results and that of Sutton's paper.**

I. Problem

In Sutton's paper [1], he proposed a new learning procedure called $TD(\lambda)$ and apply it to a random walk problem. Sutton first derived $TD(1)$, which is equivalent to the Widrow-Hoff procedure in the supervised learning context, but is formulated to be implemented incrementally and is able to make prediction on each time-step of a sequence. He then generalized $TD(1)$ by introducing a parameter λ. The resulting algorithm $TD(\lambda)$ has different weight update than any supervised learning method, and Sutton showed that in the random walk problem, $TD(\lambda)$ has better performance than the Widrow-Hoff procedure.

The $TD(\lambda)$ update procedure for the weight parameter ω is:

$$\begin{aligned}\omega &\leftarrow \omega + \sum_{t=1}^{T} \Delta\omega_t \\ &= \omega + \alpha \sum_{t=1}^{T} (P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_\omega P_k \qquad (1)\end{aligned}$$

P_t is a prediction of final observation z at time step t, and $P_T = z$ if the sequence terminate at step T. The prediction is a function of state vector x that defined by the parameter ω. For example, if P_t is a linear function of x_t, then $P_t = \omega^T x_t$ and $\nabla_\omega P_t = x_t$.

α in Eq.1 is the learning rate. The λ parameter takes value from [0,1]. It effectively scales gradient of k steps in the past by a factor of λ^k. $\lambda = 1$ is the Widrow-Hoff limit. The sum in Eq.1 can be evaluated incrementally (using the sum of previous time step):

$$\begin{aligned}e_t &= \sum_{k=1}^{t} \lambda^{t-k} \nabla_\omega P_k \\ &= \nabla_\omega P_t + \lambda \sum_{k=1}^{t-1} \lambda^{t-1-k} \nabla_\omega P_k \\ &= \nabla_\omega P_t + \lambda e_{t-1}\end{aligned}$$

Putting this in the context of random walk, we have 7 states: A-G (see figure 1). A random walk sequence starts at center state D, and can take a left or right in each step, until ends in terminate states A or G. A sequence has outcome reward $z = 0$ if it ends at A, and $z = 1$ if it ends at G. At each time step, we can predict the expected outcome based on the current state. For example, if at time step t we are at the center state D, we would predict $P_t = P_D = 0.5$ since the sequence is equally likely to end in A and G. If at time t' we are at F, then we would expect $P_{t'} = P_F$ to be some number between 0.5 and 1, since the sequence is now more likely to end in G. In fact, we can prove that the expected out come for states B-F is ($\frac{1}{6}$, $\frac{1}{3}$, $\frac{1}{2}$, $\frac{2}{3}$, $\frac{5}{6}$).

In Sutton's paper, the state vector are chosen to be unit vectors, that is $x_B = [1, 0, 0, 0, 0]$, $x_C = [0, 1, 0, 0, 0]$ ... $x_F = [0, 0, 0, 0, 1]$. In this case, P_t is seq$[t]^{th}$ element of ω. Any other set of orthonormal basis vectors can be used to represent the states, but the exact forms of P_t and $\nabla_\omega P_t$ have to be changed accordingly, and the elements of ω vector can not be directly interpreted as the expectation value of z (one has to take the general form $\omega^T x$).

II. Implementation

In this section, I discuss my implementation of $TD(\lambda)$ method to solve the random walk problem, as well as problems I encountered during the process.

Same as in Sutton's paper, I prepare 100 training sets, each containing 10 sequences of random walk. The pseudo-code for generating sequence is presented in Algorithm 1. Although the sequences can be generated on-the-fly during training process, I prepare all training sets in the beginning, and use the same 100 training sets through all experiments. This is consistent with Sutton's setting.

I explored two slightly different implementations of Eq.1 (described in Algorithm. 2 and 3). In each time step, we compute $\Delta\omega_t = \alpha * (P_{t+1} - P_t) * e_t$, where α is the learning rate, $P_t = \omega^T x_t = \omega[\text{seq}[t]]$. e_t is updated by scaling the previous e_{t-1} by the factor λ, and adding $\Delta e_t = \nabla_\omega P_t = x_t$.

The two algorithms differ in whether the weights of the terminate states A and G are trainable. In Algorithm 2, the first and last elements of ω vector are initialized to z value of corresponding terminate states (0 and 1) and never get updated in the time-iteration. In Algorithm 3, the two elements are trainable, they get updated in the last time step when the terminate state is reached. These two elements can be initialized to 0.5 as other elements, and will approach 0 and 1 respectively given the training converges. The two algorithms gives similar results and I decided to stick with Algorithm 2, because it is slightly simpler and more efficient (as final signal needs to propagate through 1 less state). As Sutton didn't consider the error of first and last elements, and his state vectors only have 5 elements, I think Algorithm 2 is also what Sutton used.

Algorithm 1 Prepare Training Sets

```
function GENERATE_SEQUENCE(start, n_state)
    seq = [start]
    s = n_state
    while s != 0 and s != n_state-1 do
        $s$ += $(rand() < 0.5? 1 : -1)$
        seq.append(s)
    end while
    return seq
end function
                                    ▷Generate 100
training sets, each has 10 sequences. Each sequence starts
at state D(idx=3) and ends at A(idx=0) or G(idx=6)
for i in 1,2,...100 do
    for j in 1,2,...10 do
        all_training_set[i][j] = GENERATE_SEQUENCE(start=3, n_state=7)
    end for
end for
```

Algorithm 2 compute $\Delta\omega$ given a sequence

```
function UPDATE($\omega$, seq, $\alpha$, $\lambda$)
            ▷first and last element of $\omega$ should be initialized
to the true observation value z (0 and 1 in this case), they
will not be updated during the iteration.
    $n_s$ = len($\omega$)
    initialize $e_t = [0] * n_s$
    initialize $\Delta\omega = [0] * n_s$
    for t = 0,1,2...len(seq)-2 do
        $\Delta e = [0] * n_s$
        $\Delta e[\text{seq}[t]] = 1$
        $e_t = \Delta e + \lambda e_t$
        $P_t = \omega[\text{seq}[t]]$
        $P_{t+1} = \omega[\text{seq}[t+1]]$
        $\Delta\omega_t = \alpha * (P_{t+1} - P_t) * e_t$
        $\Delta\omega$ += $\Delta\omega_t$
    end for
end function
```

Sutton's two experiments follows different updating procedure given each training set (10 sequences). Algorithm 4 shows the pseudo-code for experiment 1. The ω values are updated after seeing the entire training set, during which $\Delta\omega$ is accumulated. Each training set is presented multiple times until ω converges. In my implementation, the convergence criterion is set to be ... Smaller thresholds are experimented and did not bring significant change in results. It is worth noting that as $\Delta\omega$ is accumulated for the entire training set, experiment 1 typically needs a smaller learning rate than experiment 2. Sutton states in his paper that a small enough learning rate can ensure the convergence (but will certainly requires more iteration). In my implementation, I choose $\alpha = 0.1$ and scale it by the size of training set. All training is able to converge under this effective learning rate (0.01), without

Algorithm 3 Alternative way of compute $\Delta\omega$ given a sequence

```
function UPDATE($\omega$, seq, $\alpha$, $\lambda$)
▷In this case, first and last element of $\omega$ can be initialized
like other elements and will be updated in the iteration.
    $n_s$ = len($\omega$)
    initialize $e_t = [0] * n_s$
    initialize $\Delta\omega = [0] * n_s$
    for t = 0,1,2...len(seq)-1 do
        $\Delta e = [0] * n_s$
        $\Delta e[\text{seq}[t]] = 1$
        $e_t = \Delta e + \lambda e_t$
        $P_t = \omega[\text{seq}[t]]$
        if t!=len(seq)-1 then
            $P_{t+1} = \omega[\text{seq}[t+1]]$
        else:
            $P_{t+1}$ = (seq[t]==0? 0: 1)
        end if
        $\Delta\omega_t = \alpha * (P_{t+1} - P_t) * e_t$
        $\Delta\omega$ += $\Delta\omega_t$
    end for
end function
```

reaching maximum iteration = 10000.

Algorithm 4 Experiment 1 ω update on single training set

```
function TRAINING_1($\alpha$, $\lambda$, training_set)
    $\omega$ = [0, 0,5, 0.5, 0.5, 0.5, 0.5, 1]
    for n = 1,2,3...max_iteration do
        for seq in training_set do
            $\Delta\omega$ += $\text{UPDATE}(\omega, \text{seq}, \alpha/\text{traing\_set\_size}, \lambda)$
        end for
        $\omega$ += $\Delta\omega$
        if sum(abs($\Delta\omega$)) $<$ thresh then
            break
        end if
    end for
    return RMSE($\omega$[1:-1], $\omega_{true}$)
end function
```

In experiment 2 (see Algorithm 4), each training set is only presented once, and ω vector is updated for each sequence of a training set.

Algorithm 5 Experiment 2 ω update on single training set

```
function TRAINING_2($\alpha$, $\lambda$, training_set)
    $\omega$ = [0, 0,5, 0.5, 0.5, 0.5, 0.5, 1]
    for seq in training_set do
        $\omega$ += $\text{UPDATE}(\omega, \text{seq}, \alpha, \lambda)$
    end for
    return RMSE($\omega$[1:-1], $\omega_{true}$)
end function
```

Both Algorithm 4 and Algorithm 5 return the root-mean-squared error (RMSE) between ω and the theoretical values ($[\frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{5}{6}]$) after training on a given training set. Finally, given a parameter set $\{\alpha \text{ and } \lambda\}$, both experiment 1 and 2 use Algorithm 6 to compute the average and standard deviation of RMSE in 100 training set.

Algorithm 6 Compute the average and standard deviation of RMSE error in all training set

```
function COMPUTE_ERROR($\alpha$, $\lambda$, all_training_set)
    errors = []
    for training_set in all_training_set do
        rmse = TRAINING_1/2($\alpha$, $\lambda$, training_set)
        errors.append(rmse)
    end for
    return mean(errors), std(errors)
end function
```

III. Results

Before trying to reproduce Sutton's results, I did a quick sanity check on my UPDATE algorithm: I averaged the ω of 1000 training sets with size 100. The average ω vector is [0.162, 0.331, 0.495, 0.666, 0.835, 1], which is very close to theoretical value, with RMSE=0.00337.

I reproduced figure 3,4,5 in Sutton's paper. In addition to average RMSE, I also plot the standard error (estimated by $SE = \frac{std(RMSE's)}{\sqrt{n\_training\_set}} = \frac{std(RMSE's)}{10}$) in order to have a sense of statistical significance.

Figure 3 plots the average RMSE vs different choices of λ. Both my and Sutton's figure show the optimal λ is $\lambda = 0$, and the error increases as λ increases toward 1 (Widrow-Hoff limit). Despite agreeing on general trend, my RMSE ranges from around 0.11-0.18, while Sutton's ranges from 0.19-0.25. Initially I suspect this is due to randomness of training sets preparation, however, this is unlikely given the standard errors $SE < 0.01$ (this is consistent with Sutton, who also reports $SE < 0.01$). In addition, a few different run changing random seed do not alter my results significantly.

Figure 4 shows the average RMSE while altering learning rate α, at four different λ values (0,0.3,0.8,1). When $\alpha = 0$, there is no update on ω, so the error for $\lambda's$ is the RMSE of the initial guess ([0.5,0.5,0.5,0.5,0.5]), which is 0.2357. The general shape of the lines are similar to Sutton's. Errors of $\lambda = 1$ increase quickly as λ increases. Errors of $\lambda = 0.3$ and 0.8 are relatively insensitive to α. The α's that give minimum error for the above 4 λ values are (0.2, 0.2, 0.1, 0.05), while in Sutton's paper there were (0.3, 0.3, 0.15, 0.05). The standard error for the larger α values are quite large, so the lines in this region can change significantly between different runs. As I understand the last sequence in each training set would have stronger influence on ω, especially when α is large, causing more variance between training sets.

Figure 5 shows the smallest error for each λ with optimized learning rate α. $\lambda = 0.2$ has smallest error. In Sutton's paper, the minimum falls between 0.2-0.3. The minimal error is around 0.09, while Sutton's minimum is above 0.1. In both my and Sutton's results, $\lambda = 1$ gives largest error, but maximum value is different. Overall, the results shares similar pattern as

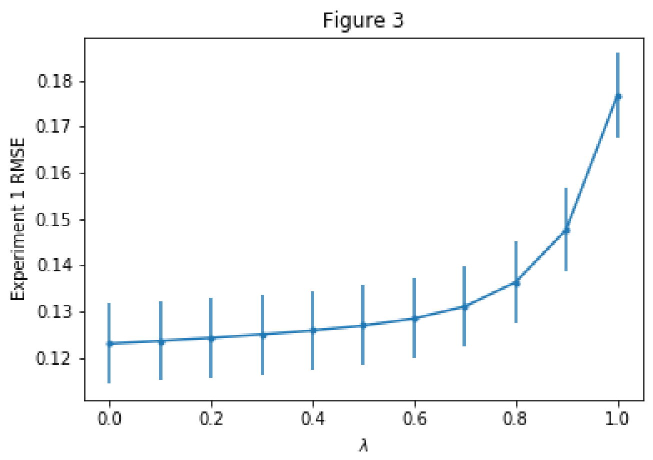

Fig. 1. Replication of Figure 3 in Sutton's paper. Experiment 1 average RMSE over 100 training set for different λ values. The error bar is $\pm 1 \times SE$ computed from 100 training set.

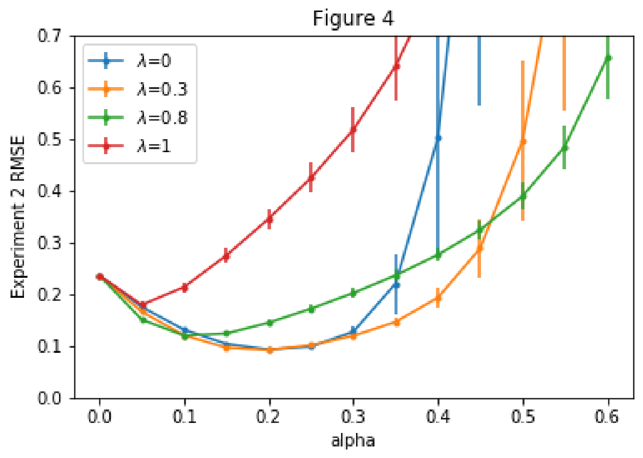

Fig. 2. Replication of Figure 4 in Sutton's paper. Experiment 2 average RMSE over 100 training set, for different learning rate α given $\lambda = 0, 0.3, 0.8, 1$.

in Sutton's paper, with noticeable difference in the range of RMSE values. Again, the small standard error (especially for smaller λ's) makes me think that the discrepancy is unlikely due to randomness in training sets.

References

[1] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

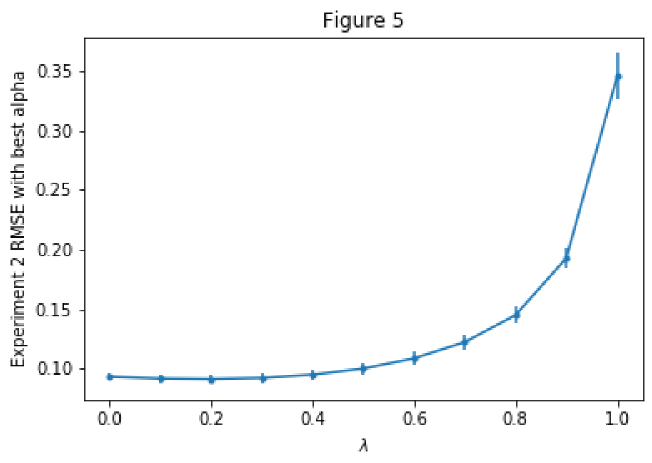

Fig. 3. Replication of Figure 5 in Sutton's paper. Experiment 2 average RMSE over 100 training set, for different λ values at optimal α.