# Project 1 Report, CS 7642 Reinforcement Learning and Decision making

Farhad Batmanghelich (fbatmang3@gatech.edu)

Git hash: ee07beb9e296614ced7220e8eec412b0ad71b071

***Abstract*— In this report, results of two of the experiments presented in Sutton 88 [1] are replicated. Experiments were performed on a bounded random walk system. Through this process, fundamentals of Temporal Difference (TD) learning is reviewed, specifically impact of learning rates and lambda values are investigated.**

***Keywords—temporal difference learning, reinforcement learning***

## I. Introduction

Reinforcement learning is one mechanism for sequential decision making. In sequential decision making, usually there exist an agent that lives in an environment that is divided into different states. The agent interacts with the environment by taking actions that would take it to another state and present it with a reward. In such systems the goal is to come up with a policy, i.e. a function that returns which action to take at a given state, that maximizes the expected return received by the agent. Expected return of a state is called the value of the state and the policy is a function that at each state returns the action that generates the highest value. Now if the dynamics of the system, i.e. transition probabilities are known, the task is not really learning, instead it is planning. We can use dynamic programming methods by expressing values by Bellman equation and compute optimum policies through value iteration algorithm. But if the dynamics of the system is not known a priori, agent should rely on sample observations to learn values of states. Learning can be done by observing a whole episode of the sequence and updating the estimate of the value by comparing initial guess of the value with the observed return, and averaging overall observations to estimate the expected return, i.e. value. In this paper, Temporal Difference (TD) learning is presented as the hallmark of reinforcement learning as it allows the learning agent to use rather successive observations to update its value estimates. Sutton uses a simple state sequence system, bounded random walk, to perform two experiments revealing the difference between supervised and TD learning methods.

## II. Concepts

### A. Value function

Value of a state is defined as the expected return, if the agent starts from that state and follows the policy. Value estimate of a state can be written as a function of states and some parameters (weights). At a given time, $t$, state can be represented as a d-vector, i.e. $state_t \in R^d$, along with another d-vector for weights, $w \in R^d$, and the value estimate can be written as a linear function of state and weight vectors, i.e. $V(state) = w^T.state_t$

### B. Multistep prediction problem

In Sutton 88, a multistep prediction problem is a state sequence $x_1, x_2, \dots, x_m$ that culminates in a prediction value, $z$. Each state at time $t$ is a d-vector $x_t \in R^d$ and prediction value is a scalar, $z \in R$. Return of an episode of the sequence is the sum of discounted rewards of that episode. From my understanding of Sutton 88, *outcome value*, $z$, is the return of the state that an episode starts in and we try to predict (estimate) expected value of $z$ through observation. *Ideal predictions* are true expected return values of states which can be computed if transition probabilities are known. Sutton uses ideal predictions as a gold standard to measure performance of learning procedures.

As mentioned in the previous section, value estimate of a given state, can be written as a function, in this case linear, of state and a weight vector, $P_t = W^T x_t$ where $x_t, w \in R^d$ and $P_t$ is the

estimate of expected return. Now there are two approaches to learning. In one approach, the learning procedure entails observation of a full episode of the sequence, recording the return and using that return to update the initial guess we had for expected return. This approach where observation of a full episode is required, is called *supervised learning*. On the other hand, observation can be limited to a few steps of an episode prior to updating $w$. This method is the TD method which later is shown that the supervised learning is a special case of TD learning, namely TD(1).

### C. Bounded random walk outline

The bounded random walk, as presented in Sutton 88, is a state sequence that is generated by starting at a given state (D in Figure 1) and randomly stepping right or left with equal probability until a terminating state is reached. There are two terminating states, A with reward 0 and G with reward 1. All other states have 0 reward (Figure 1).

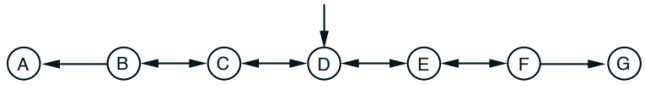


**Figure 1**. Bounded random walk Markov process [1]

Transition probabilities in this system are known, therefore the true value of states can be computed using Bellman equation from dynamic programming and used as gold standard to measure performance of learning procedures.

In Sutton 88, each non-terminal state at time t, is represented as a Boolean 5-vector with all zeros except agent's location, $\boldsymbol{x_t} = \boldsymbol{x_i} \in R^5$ . For example, if agent at time t is at state B, then $x_t = [1,0,0,0,0]^T$ or if it is at state E, then $x_t = [0,0,0,1,0]^T$. Given that value estimate is a linear function of $\boldsymbol{x_t}, \boldsymbol{w}$, the reason for this representation is that the value estimate of a state in the i[th] location of state vector can be given by only the i[th] component of the weight vector. For example, value estimate of state C, is:

$$P_t = [W_B \quad W_C \quad W_D \quad W_E \quad W_F] \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = W_C$$

### D. Supervised learning approach

As previously mentioned, the supervised learning approach entails observation of a full episode before the value, which in the random walk is equal to weight, is updated:

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

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

$$\xrightarrow{\nabla_w P_t = x_t} \Delta w_t = \alpha(z - P_t)\mathbf{x}_t$$

In above equation, since value function is linear, gradient of $P_t$ w.r.t $\boldsymbol{w}$ equals $\boldsymbol{x_t}$ and $\alpha$ is the learning rate. The above update rule is also called Widrow-Hoff. One downside of the Widrow-Hoff rule is the strict dependence on $z$, which makes it impossible to incrementally update values. To overcome this shortcoming, we can write the error term, $z - P_t$ as the sum of consecutive changes in predictions:

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

Which changes weight increments to:

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

$$= \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \boldsymbol{x}_t$$

Above can be computed incrementally which is much more space-complexity efficient as there is no need to save all past values of $\boldsymbol{x_t}$ before value estimate is updated. In the next section, $TD(\lambda)$ will be described and we will see that the above equation is $TD(1)$.

### E. $TD(\lambda)$ procedures

In the equation for weight updates in the previous section, there is no notion of latency of a state on its impact on $\Delta w$. To this equation, we can add a weight, $0 \leq \lambda \leq 1$, that gives a lower impact to $w$ update for later observed states. Depending on the value of $\lambda$, this will generate a family of learning procedures called $TD(\lambda)$ which can be computed

incrementally. For $TD(\lambda)$, $w$ alterations are given by:

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

## III. Experiments and results

In Sutton 88[1], there are two experiments where performance of various $TD(\lambda)$ procedures are compared by measuring their value estimates against that of true values as gold standard. In both experiments, 100 training sets, each containing 10 random-walk sequences are generated. Since bounded random walk is a simple system with known transition probability, we can compute true values of states (which are equal to weights) by writing Bellman equation for each state and then solving the linear system of equation. Bellman equation along with system of linear equations to compute true values is as follows:

$$V_\pi(s) = \sum_{s',r} p(s', r|s, a)[r + \gamma V_\pi(s')]$$

Here return is un-discounted, i.e. $\gamma = 1$, therefore:

$$\boldsymbol{V(B) = 0.5V(C)} \qquad \boldsymbol{V(E) = 0.5[V(D) + V(F)]}$$
$$\boldsymbol{V(C) = 0.5[V(B) + V(D)]} \qquad V(F) = 0.5[V(E) + 1]$$
$$\boldsymbol{V(D) = 0.5[V(C) + V(E)]}$$

After solving the above system of linear equations, true values of states B, C, D, E, F will be $[\frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}]^T$ which is equal to probability of right hand side termination for each state.

In experiment 1, for each training set, $\boldsymbol{w}$ vector is initialized and $\Delta\boldsymbol{w}$s are added together after each sequence in that training set. After all 10 sequences were passed, accumulated $\Delta\boldsymbol{w}$s were added to the previous $\boldsymbol{w}$**.** If the difference between the new $\boldsymbol{w}$ and initial guess for $\boldsymbol{w}$ is more than a threshold, $\Delta\boldsymbol{w}s$ (not $\boldsymbol{w}$**)** are set to zero and all 10 sequences in the training set are presented again until the difference between old and new $\boldsymbol{w}$ is less than the previous iteration. This is done for various $\lambda$ values. Experiment 1 is also called the *repeated presentation* scheme. The pseudocode I used to replicate experiment 1 results is presented below. In experiment 2, which is also called *single presentation* scheme, unlike experiment 1, each training set is only presented once, and the goal of this experiment is to explore the effect of both $\lambda$ and learning rate $\alpha$. Another difference is that in experiment 2, $\boldsymbol{w}$ update is performed after each sequence rather than after all the sequences in a training set is done. The pseudocode I used to replicate experiment 2 results is as follows:

---

**Experiment 1 algorithm:** repeated presentation scheme

**Input:** list of $\lambda$ values, matrices capturing vector of states for each episode, list of rewards for each episode

**Output**: a dictionary keyed on $\lambda$ and valued on root mean squared (RMSE) error between true values and value estimates

**for** $\lambda$ in list of $\lambda$ values **do:**
  **for** training set in list of training sets **do**
    initialize $w$ vector to all 0.5 and keep a copy of it in another variable $\boldsymbol{w_{prev}}$
    **while** difference between $\boldsymbol{w}$ and $\boldsymbol{w_{prev}}$ less than a threshold
      set $\Delta\boldsymbol{w} = 0$
      **for** sequence in training set **do**
        **for** each step in the sequence **do**
          $$\Delta\boldsymbol{w} \mathrel{+}= \boldsymbol{\alpha}(\boldsymbol{w^T}.\boldsymbol{x_{step+1}} - \boldsymbol{w^T}.\boldsymbol{x_{step}})\sum_{\boldsymbol{k=1}}^{\boldsymbol{step}}\boldsymbol{\lambda^{step-k}}\boldsymbol{x_{step}}$$
      $\boldsymbol{w} \mathrel{+}= \Delta\boldsymbol{w}$
      check if $|\boldsymbol{w} - \boldsymbol{w_{prev}}| \leq threshold$
    $RMSE[\lambda] \mathrel{+}=$ root mean squared error between true values (weights) and $\boldsymbol{w}$ /100. Division by 100 because RMSE is averaged over all 100 training sets

---

**Experiment 2 algorithm:** single presentation scheme

**Input:** list of $\lambda$ values, list of $\alpha$ values, matrices capturing vector of states for each episode, list of rewards for each episode

**Output**: a dictionary keyed on $\lambda$ and valued on lists that capture root mean squared error (RMSE) between true values and value estimates for each $\alpha$

**for** $\lambda$ in list of $\lambda$ values **do:**
  **for** training set in list of training sets **do**
    initialize $W$ matrix to all 0.5, number of columns of $W$ is the same as number of $\alpha$ values, i.e. each column of $W$ is a $\boldsymbol{w}$ vector for the corresponding $\alpha$ value.
    **for** sequence in training set **do**
      **for** $\alpha$ in list of $\alpha'$s
        set $\Delta\boldsymbol{w} = 0$
        **for** each step in the sequence **do**
          $$\Delta\boldsymbol{w} \mathrel{+}= \boldsymbol{\alpha}(\boldsymbol{w^T}.\boldsymbol{x_{step+1}} - \boldsymbol{w^T}.\boldsymbol{x_{step}})\sum_{\boldsymbol{k=1}}^{\boldsymbol{step}}\boldsymbol{\lambda^{step-k}}\boldsymbol{x_{step}}$$
        $\boldsymbol{w}[:, this\ \alpha's\ column] = \Delta\boldsymbol{w}$
    $RMSE[\lambda] \mathrel{+}=$ a list that for each $\alpha$ captures the root mean squared error between true values (weights) and $\boldsymbol{w}$/100. Division by 100 because RMSE is averaged over all 100 training sets.

---

Figure 2 illustrates a replication of Figure 3 of Sutton 88. As can be seen, $TD(0)$ has the best performance an $TD(1)$ has the highest error which is due to the fact that this learning procedure only minimizes error on the training set and neglects future experiences. Comparing this with Figure 3 of the paper, seems my replication has created a smaller error which could be due to the difference in training sets and sequences within them. In fact my implementation uses random.choice Python function to randomly select which side of a given state to jump. This would generate different sequences at each run and therefore results might slightly vary from run to run. Another culprit for the discrepancy could be the $\alpha$ value that Sutton has used. I'm using 0.01 , however there is no direct report in the paper about the $\alpha$ value used.

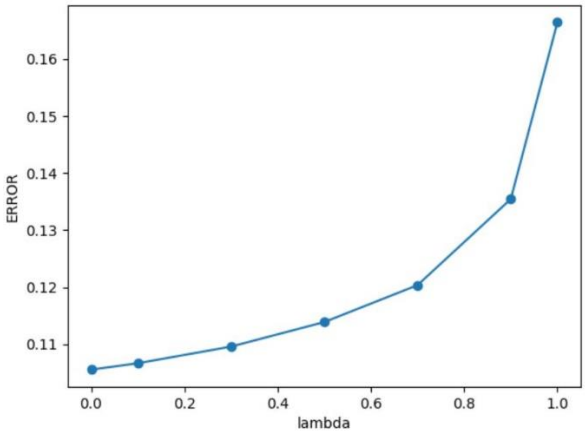


**Figure2**. replicate of Figure 3 of Sutton 88[1]

Figure 3, depicts a replication of Figure 4 of Sutton 88[1]. There are similar features among the two figures. For example, for $\alpha < 0.5$ Widrow-Hoff ($\lambda = 1$) has the highest error among all learning procedures in both figures. Furthermore, at learning rate $\alpha = 0$ error becomes independent of $\lambda$.

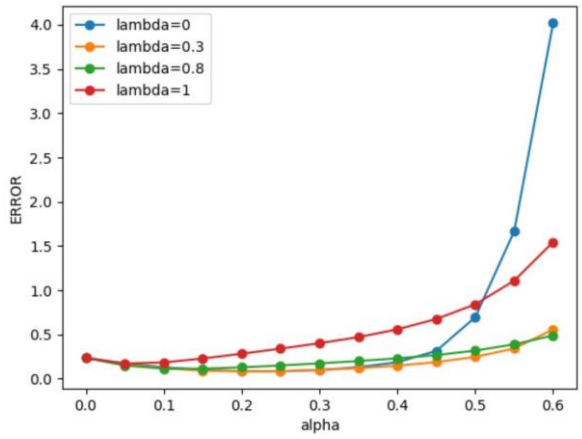


**Figure3**. replicate of Figure 4 of Sutton 88[1]

Figure 4, is a replication of Figure 5 of Sutton 88[1]. In order to generate this figure, for each $\lambda$, the error associated with the learning rate ($\alpha$) that has the lowest error is used. There is good agreement between Figure 4 and Figure 5 of the original paper with Widrow-Hoff ($TD(1)$) having the highest error. In this figure, $\lambda = 0.3$ corresponding to $\alpha = 2$ seemingly has the lowest error, even better than $\lambda = 0$. For $TD(0)$, the extend of reach for propagating prediction levels back is small. In experiment 1, where sequences were repeatedly presented until convergence, this should not be a problem (Figure 2 of this report and Figure 3 of Sutton 88[1]). However in experiment 2 (single presentation scheme) this has caused $TD(0)$ to perform worse than a procedure with a slightly higher $\lambda$, i.e. $TD(0.3)$.

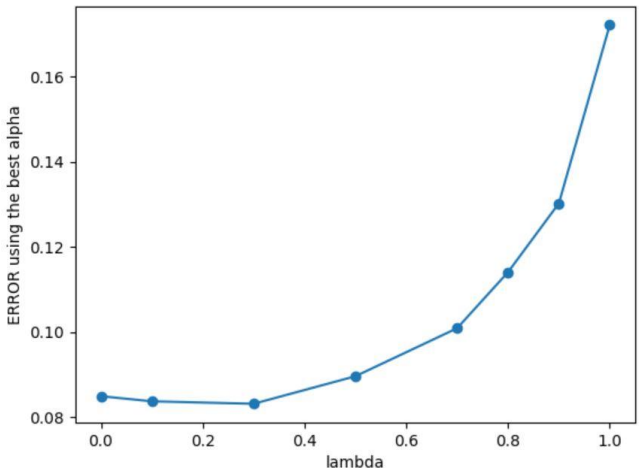


**Figure4**. replicate of Figure 5 of Sutton 88[1]

## IV. CONCLUSIONS

In this report, learning procedures that resulted in Figures 3, 4 and 5 of Sutton 88[1] were replicated. Replicated results had reasonable similarity with that of the original paper. There were slight differences in the figures which were identified to be due to the discrepancies in the randomly generated training sets. Another potential reason for slight deviations is that actual learning rate, $\alpha$ is not reported in the original paper. In both schemes, repeated and single presentations, learning procedure that looks less in the future and relies heavily only on the training set, i.e. Widrow-Hoff ( $TD(1)$ ) performed worse than the TD procedures with $\lambda < 1$. Another observation that was successfully replicated was the impact of repeated presentation of sequences on the

performance of $TD(0)$. In single presentation scheme, $TD(0)$ has a limited reach for propagating back predictions along a sequence and therefore, it performs worse than slightly higher $\lambda$ values.

## V. REFERENCES

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

[2] R. S. Sutton, A. G. Barto, Reinforcement Learning, An Introduction, 2nd edition, MIT Press, (2020)

[3] Lectures of Georgia Tech's CS 7642 (2021)