# CS 7642: Reinforcement Learning: Project 1

Ahmed Bilal

## I. Introduction

In his paper 'Learning to Predict by the Methods of Temporal Differences'[1], Richard Sutton introduces a class of incremental learning procedures: Temporal Difference (TD) Learning Methods specializing in prediction in an incompletely known system. In contrast with conventional prediction-learning methods that aim to update their learning parameters based on the model prediction and actual label loss function, incremental learning methods base their parameter updates on the loss function of the difference between temporally successive predictions. Sutton aims to show that in any dynamic system with observable states evolving over time, TD methods learn more efficiently than conventional supervised learning methods. The aim of this report will be to replicate Sutton's 'Random Walk' example to verify his findings that TD methods are more efficient in producing more accurate predictions. It is important to note here that Sutton's experiment has far reaching consequence, since through his example he aims to show not just about state and reward systems, but any system with observable evolving states that TD($\lambda$) methods tend to perform better than supervised learning methods.

## II. TD($\lambda$)

TD methods in essence are learning methods that make a prediction by learning from another prediction. This is in direct opposition to conventional supervised learning algorithms (Windrow-Hoff), TD(1) methods, that learn from actual outcomes at the end of the episode. TD($\lambda$) take the exponential weighted sum of TD learning methods with a TD value all the way up till $\lambda$. Sutton makes the case that the reason TD ($\lambda$) is a more efficient learning model than TD(1) methods is because deriving our prediction from future predictions allows for more efficient backpropagation of learning parameter updates than waiting for the actual outcomes of an entire episode. In addition, it seems to me that the reason TD($\lambda$) is a more efficient algorithm is because by using predictions, it is able to significantly increase the training set in the same way that data augmentation is used in Deep Learning for instance. In contrast, TD(1) is limited to only the outcome training example. And since TD($\lambda$) is able to use more training data than TD(1), it is able to construct a Markov Decision Process system and use Maximum Likelihood Estimates to solve it. This allows it generalize better to future predictions and not overfit to the training data up until t as TD(1) does.

[1]Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. Machine Learning, 3(1), 9–44. https://doi.org/10.1007/bf00115009

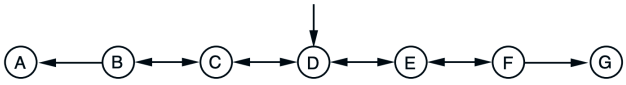



Fig. 1. Suttons' Random Walk

## III. Random Walk

Sutton's Random Walk example is meant to showcase one of the simplest instantiation of a dynamic system. The simplicity is meant to eliminate discussion based on the particular complexities of the system being analyzed. In the Random Walk, there are seven states, as showcased in Fig. 1. State A and G are the terminal states with rewards 0 and 1 respectively. The player starts at state D and moves either right or left with equal probability at each time step, t, until the player reaches a terminal state, receives the subsequent terminal state reward and a sequence/episode ends.

A single walk from start state D to terminal state A or G will be referred to as an episode. The learning methods aim to estimate the expected value for each non-terminal state. For each non-terminal state, the expected value equals the probability of a right side termination. In a single episode, for each time interval t, there is an observation vector, $x_i$ where $x_t = x_i$. As such, each $x_i$ corresponds to a vector with 0's corresponding to all but the state at which the player is at, at time t, and a 1 for the occupied state at t. The learning procedure for a single sequence is a matrix of all the transpose vectors for each $x_i$ in the episode. For example, for a sequence [D,C,B,0], the learning procedure would be:

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

Since the learning methods are predicting the expected value for each non-terminal state, the actual values of the non-terminal states are as follows: [B:1/6, C:2/6, D:3/6, E:4/6, F:5/6]. As such, our model performance will be judged by how close the predictions are to these actual values. We will use the Root Mean Square Error as our loss function.

## IV. Experiment 1 - Repeated Presentations

For the first experiment, we will simulate training data with 100 sets, where each set corresponds to 10 episodes. In addition, we keep the learning rate $\alpha$ constant at 0.01. We will then use supervised learning method - TD(1) and incremental learning procedures with different lambda values - TD($\lambda$) to make predictions of the expected state values. After

which, we will use Root Mean Square Error loss function to compare each method's performance. The RMSE scores will be averaged for the 100 sets to create statistically reliable statements about each method's performance.
We start by initializing all the weights for each of the non-terminal states to 0.5 to create a neutral starting point. For each episode, at each step, t, we use our weights at t, $w^T$ to calculate the prediction of the probability of the state occupied at t, for a right side termination, $P_t$ using Sutton's equation A[2]:

$$P_t = w^T x_t = \sum_i w(i) x_t(i) \tag{1}$$

From the above equation it also follows:

$$\begin{aligned} P_t &= w^T x_t \\ \nabla_w P_t &= x_t \end{aligned} \tag{2}$$

Depending on the $\lambda$ value of our current TD($\lambda$) model, we calculate $\Delta w_t$ using Sutton's equation 4[3]:

$$\Delta w_t = \alpha \left( P_{t+1} - P_t \right) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_t \tag{3}$$

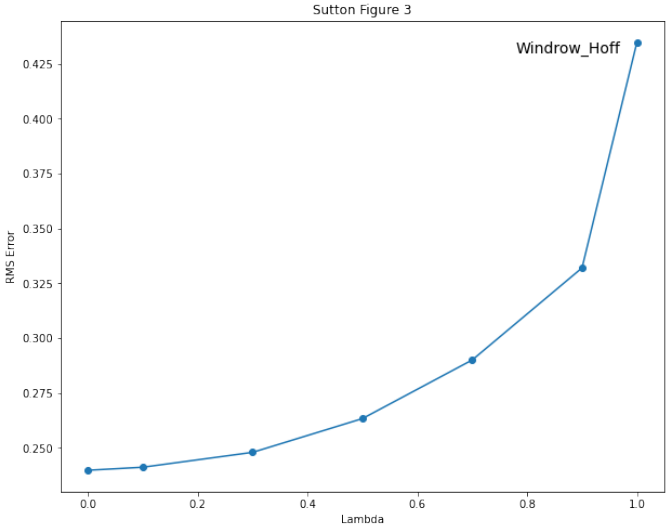
For experiment 1, we will continue to sum over the weight update for an episode till the end of all the 10 episodes in the training set. We continue to sum together the $\Delta w$ values for the entire training set, and only once we have iterated over all 10 episodes in the set, do we update w with the summed $\Delta w$ values using Sutton's equation 1:

$$w \leftarrow w + \sum_{t=1}^{m} \Delta w_t \tag{4}$$

Each of our $\lambda$ methods will repeat this iteration and update over the training set until we no longer see any significant changes in the weight vector before moving on to the next training set. Once, all of the 100 training sets have been similarly iterated over, we will find the RMSE of our final predicted weights and actual weight.

### A. Analysis

In testing, we found that after around 5 iterations on a training set, there were very small changes in the weight vector. As such, we assumed convergence at that point for the training set and moved on to the next set. Sutton does not mention any explicit convergence criteria and as such, our decision to assume convergence after 5 iterations is arbitrary. As such, choosing different convergence criteria would have yielded slightly different RMSE values for each $\lambda$ value. This arbitrary convergence criteria, randomly generated training data and choice of $\alpha$ value explains why the RMSE values for experiment 3 are slightly different from Sutton's implementation of experiment 1.
Despite this difference in the RMSE values due to convergence criteria, it seems that the relative difference in RMSE for the different $\lambda$ values is maintained in a manner similar to Sutton's implementation. Since the aim of experiment 1 is not to showcase any specific RMSE values for the learning models, but to illustrate the fact that TD($\lambda$) models should learn more efficiently and thus have lower training loss values, we observe good replication of Sutton's overall claim for experiment 1 results.



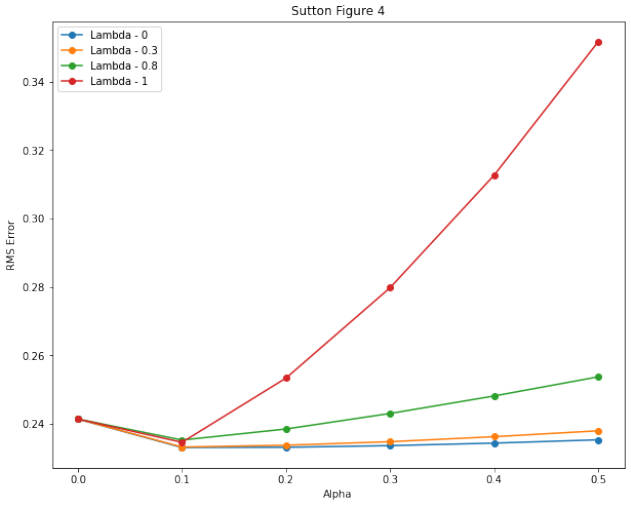

We also found that a small $\alpha$ value is necessary for this experiment to avoid overflow issues in testing during convergence. As it seems a higher $\alpha$ value makes it harder to converge at each training set.
Our usage of a training set that is repeated until convergence and weight updates after a whole set allows us to mitigate the effects of the step size $\alpha$ in our results. As we move to experiment 2 and relax our repeated presentation idea, we will see that the step size again becomes very relevant in our results and needs to be explicitly accounted for.

## V. Experiment 2 - Single Presentation

Most of the experiment 2 setup is similar to experiment 1. Only this time, each training set will be presented only once to the learning models, and weights are updated after each episode instead of accumulating them for the whole set and updating after. For experiment 2, we will begin by using different $\alpha$ values for each of our $\lambda$ methods.

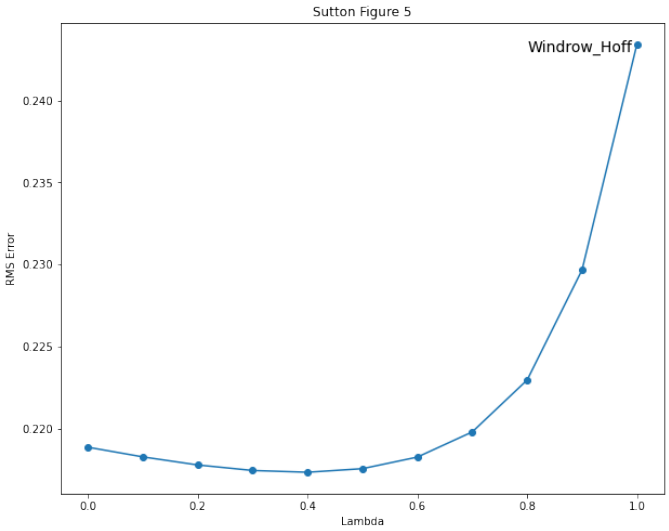
In experiment 2, we will update our weights after each episode. After finishing learning from a training set, we get the RMSE loss value for the training set. We then take the average loss value for the 100 runs. For each of our lambda methods, we chose different alpha values for our learning models and plot the error value averaged over all our training set for each (lambda, alpha) pair. Once we plot these error values, we end up with the following pattern.

[2]Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. Machine Learning, 3(1), 9–44. https://doi.org/10.1007/bf00115009
[3]Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. Machine Learning, 3(1), 9–44. https://doi.org/10.1007/bf00115009




### A. Analysis

Similar to Sutton's Fig 4, we observe a kink in our test results with most of our TD($\lambda$) methods performing the best for $\alpha$ values in the 0.1 to 0.3 range. The most notable similarity is the replication of the fact that TD(1) seems to perform worse than all other TD methods. The difference in performance becomes all the more significant as the $\alpha$ value increases.
However, unlike Sutton's Fig. 4, we see a linear pattern for all TD methods post the kink as we increase the $\alpha$ value. This difference can be attributed to floating point, rounding issues and random walk training data instantiation. This is evidenced by the fact that there was variance in the results for different random seed values when producing the training data.
Despite these differences, the main result of Sutton's Fig 4 can be clearly seen here by the fact that TD() methods are more efficient than TD(1), irrespective of our $\alpha$ values and the best model performance is seen for a learning rate that is not too large or too small in order to make weight updates that are not too insignificant or too large - subsequently overshooting good weight parameters.
We can now take the best alpha value for each of our lambda values and plot the average loss function for our TD $\lambda$ functions for experiment 2. This results in a pattern similar to Sutton's figure 5.




### B. Analysis

As discussed previously, the difference in particular RMSE values between our implementation and Sutton's Fig 5 can be chalked up to specific implementation differences as discussed above. Overall, our results match Sutton's Figure 5 results pretty well. This illustrates that with the best $\lambda$ value for each TD method respectively, we see that TD(1) seems to perform worse than all TD($\lambda$) methods. In addition, we observe that it seems that on average, TD($\lambda$) methods with $\lambda$ values greater than 0 and less than 1 perform the best.

## VI. Conclusion

In conclusion, despite the difference in our replication results due to the exact implementation conditions of Sutton's experiments, we were able to find results overall pretty similar to Sutton's. As such, we have good replicable empirical evidence for Sutton's theoretical claims of TD algorithms being more efficient algorithms than supervised algorithms for dynamic systems.