Recreating Sutton’s Temporal-Difference Learning

# Random walk

The random walk problem is a Markov process. Given a list of linear states, A through G in this paper’s example, the program has a 50% chance of going either to the left or right side. The walk only ends at the absorbing states at either the far right or left, in this case A or G. The program will keep going until it has hit one of these absorbing states. In our problem the program always starts at D, the middlemost state. A possible example would be DEDCBA or DEFEG.

The goal of the TD algorithm used in Sutton’s paper is to find the expected value of terminating to the right at each state of the random walk, which is at the state G. Using statistics, the actual expected value is known to be 1/6, 1/3, 1/2, 2/3, and 5/6 respectively for B, C, D, E, and F [1]. These values will be used later to find the root mean squared error between the actual values and the expected values.

# The TD(λ) Algorithm

Since Sutton only deals with linear functions in his paper, we can say that . The P stands for predictions which come from the sequences given from the random walk problem. The predictions are an estimate of z, which is the outcome of the sequence or the reward of the sequence. Ending in G would be a 1.0 reward and ending in A would be a 0.0 reward. w is the weight vector which will be updated during the algorithm as such:

where m is the number of states (not including absorbing states). The supervised-learning update procedure is given as

where α is the learning rate and are the partial derivatives of with respect to each component of w. On pages 14-15, Sutton breaks this down into an equation that can be computed incrementally rather than waiting for the end of a sequence, as supervised learning would require. Making this equation to be incremental allows for the algorithm to compute and update weights before the result or reward is known. Combining that equation with a λ to give diminishing returns for steps in the past we have the TD(λ) algorithm

λ is between 0 and 1 inclusively with λ = 0 giving the TD(0) method where the algorithm looks only one step ahead. λ = 1 would give the TD(1) method which when given Sutton’s Theorem 1 produces the same per-sequence weight changes as the Widrow-Hoff procedure.

Since the problem given is a linear one, which reduces the equation to

There is an observation vector which is a unit vector where all states that the learner is not in is 0 and the one the learner is in is 1. For example . Given them, the equation for the change in weights is

The reason why TD(λ) is preferable over traditional supervised learning algorithms is that it propagates rewards back through the states and the current state is dependent upon the state in which it goes to next. Whereas traditional supervised learning only cares about the reward. The game-playing example shows this concept, where if a learner hits a novel “bad” position, supervised learning will fully equivocate winning with this position. Whereas a TD approach would realize this is a bad position and would be able to buffer that reward correctly. He does go on to say that both methods should converge to the same evaluation with infinite experience, but the TD method learns much faster. This is the point Sutton looks to prove in his experiments.

# The experiments

## Updating After the Training Set

In Sutton’s first experiment, he only updated the weight vector after the entire training set had been seen by the learner. This is very much in line with a traditional supervised learning algorithm. Meanwhile he accumulated the change in weights and only added it to the weight vector at the end of the set. In this experiment the training set was presented repeatedly until the weight vector converged, which is to say that the change in the weight vector was smaller than some given epsilon. He ran the algorithm for a small α and λ’s of 0.0, 0.1, 0.3, 0.5, 0.7, 0.9, and 1.0.

Sutton used the root mean squared error (RMSE) between the expected and actual value. He averaged the RMSE for a given algorithm over 100 training sets each having 10 sequences. He found that the performance was best at λ = 0 and declined in performance until λ = 1. He argued that the Widrow-Hoff procedure only minimizes the error on the training set and does not minimize error for future experience which is why TD(0) was so successful.

## Updating Every Sequence

The second experiment he updated the weights with every sequence instead of after the entire training set. He also does not wait for convergence, rather he shows the training set once and computes the RMSE. He computed with different values of α from 0.0 to 0.6 in 0.05 increments for multiple λ. α had a significant effect on every single algorithm’s performance but in every case, a smaller λ outperformed the Widrow-Hoff procedure.

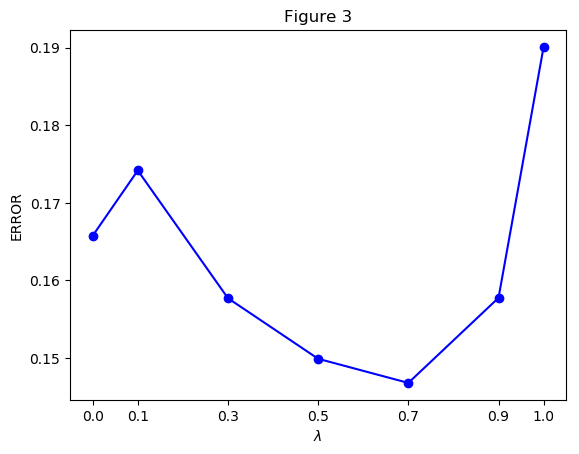
In figure 5 in Sutton’s paper, he plots the best error level achieved for each λ value. In this plot the best λ value is not at 0 but around 0.3. He argues that this is intuitive because TD(0) would be slow at propagating prediction levels back through the sequence, so some λ between 0 and 1 would outperform both TD(0) and TD(1).

# Implementation

## Training Set Generation

The first part of implementing Sutton’s algorithm was to create the training sets. Sutton did not include his training sets in his paper. There can be a large amount of variation in the training sets. For example one training set can have a sequence of length 4, DEFG, whereas another can have a length of 13 or more. There were not any rules given if certain sequences should be thrown out or if sequences should have a maximum length. My training set generation was simple. It ran through a random walk procedure starting at state D and with a 50% chance went either left or right until it ended in an absorbing state. It generated 100 training sets with 10 sequences each, all outputted to files so that I could use the same training sets for each experiment.

## Probability Estimator



### Updating After the Training Set

For both experiments I store the x vectors in an array. For example, x[1] would correspond to which is [0,1,0,0,0]. States A and G are not stored in the x vectors as they are treated as rewards.

Sutton did not specify for his first experiment what he initialized his first weight vectors to. Because he initialized them all with 0.5 in the second experiment, I did this as well for the first experiment. He also did not specify an epsilon. I used an epsilon of 0.001. I used 100 training sets with 10 sequences each, as Sutton did. I used the same training sets on both experiment one and two.

For reference, my code for experiment one can be found in ProbabilityEstimator with most of the code starting at method noUpdatingPerSequence. The algorithm’s steps proceed as follows: while the change in weight was more than the epsilon, I showed the algorithm the same training set. I computed the TD(λ) algorithm for determining the change in weight, which I call the function, deltaW, in the code. deltaW is the same for both experiment 1 and 2 and returns the vector of the changes in weight which should be added to the weight vector when appropriate.

The deltaW function has an array called delta\_w which will hold the change in weight throughout the calculation. The algorithm proceeds into a loop for 0 to the number of steps in the sequence, which is the length of the sequence minus one. I calculate with the indexes in code altered to accommodate for the fact that while parsing the sequence A = 0 and G = 6, but the weight vector starts at B since A and G are not included in the weight vector. For this reason, I also must have separate logic if the step I am calculating is the last step, as the last state will be G or A and therefore I must return the reward instead of the weight. After that, I calculate with the caveat that if λ is 0 to just return else the program would not execute correctly. I multiply the two parts together and add to the delta\_w array. After the calculation is complete, I return the delta\_w.

For experiment one, the change in weight is not added until the entire training set has been shown to the learner.

For both experiments I calculate the RMSE the same way using a Python library and using the same expected values that Sutton gave in his paper. That is the main output of the program for both experiment one and two.

### Updating Every Sequence

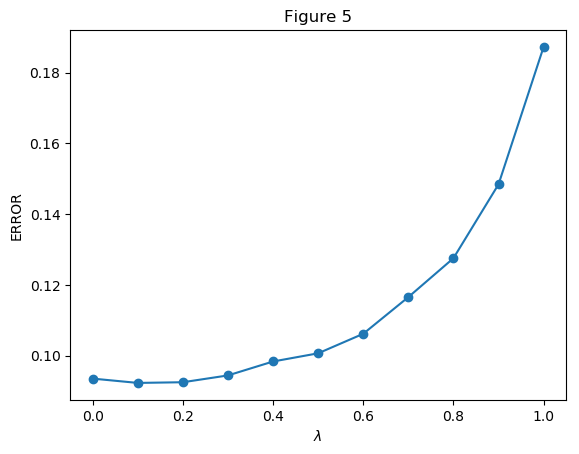
There are two differences between experiment one and two. The first being that the change in weight, delta\_w, is added per sequence instead of every training set. The second difference is that each training set is shown once instead of shown until convergence. The method that starts the algorithm is called updatingSequences in my code.

# Results

The result from the first experiment is given by Figure 3. I kept the naming conventions for the figures the same as they are in Sutton’s paper for less confusion.

A big difference is that TD(0) is not the best estimator in my results, but rather λ = 0.7. There are a couple reasons that may explain this discrepancy. Firstly, my training set is very likely different from Sutton’s. Also if he threw out longer sequences or just happened to have very short sequences it would explain why TD(0) would do much better, because it would require less error propagation for a shorter sequence. Secondly, he did not specify his α value. I toyed with several α’s between 0.03 to 0.15 (since he did mention it converged for small α’s) for different λ values and just decided on an α equaling 0.04 since it did the best for TD(0). A third explanation is I did not know what he set his initial weights to, which could have been all zeros or a random decimal between 0 and 1, which are common initializers. However, the explanation for the difference is probably the combination of data sets and α values. I would also argue that it makes more sense for the optimal value be some λ between 0 and 1 to accommodate for diminishing rewards propagated through the sequence.

But the really important takeaway from this figure is that all values of λ less than one outperform TD(1) which is the supervised learning method. This was what Sutton really wanted to prove and the graphs do agree on this point.



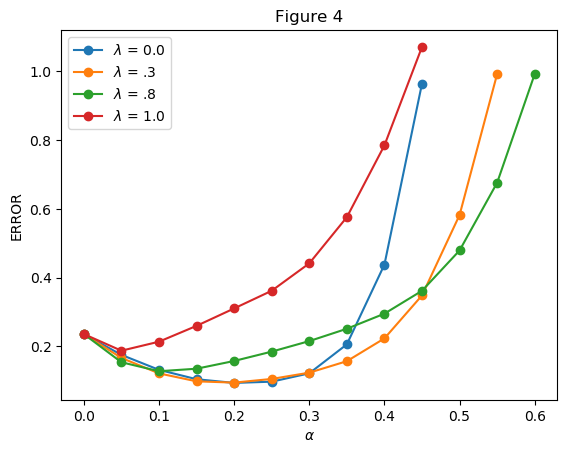


Figure 4 shows the results of the second experiment which look pretty similar to Suttons. I removed any data points that had error larger than 1 otherwise the differences between the different λ’s would be hard to tell. Sutton also does this in his paper for λ equaling 1.

The first big thing to note is that TD(1) is outperformed by every TD(λ) (with some discrepancies at larger α values for TD(0) which are not shown here). The α’s make a big difference in the performance of the algorithms, much like Sutton’s graphs show. The lines for the different λ’s have the same shape as Sutton’s showing that a small α is preferable and the algorithm will do better until a certain point where the α is too large and the error grows quickly from there.

A difference is that my TD(0) did not do as well as Sutton’s. The reason for this probably lies in the datasets. For example my 2nd training set has a sequence of DCDCBCDEDEFEDCDEDCDEFEDCBCDEDCDCDEFEFG, which is a highly unlikely sequence and since there are only 10 sequences per training set, this would alter the weight values quite a bit. A higher λ value would handle this case better than TD(0) would. You can imagine if the weights were [0.0, 0.1, 0.5, 0.8, 0.9] that TD(0) going from D to E would increase by but that going from D to C would decrease by , so if you have a sequence like the one given, and say going from D to E happens the same amount of times as going from D to C then is going to decrease in weight a lot more even though the reward will be positive at the end, since that reward will not propagate back to with TD(0). This probably could be mitigated with a higher number of sequences per training set, but 10 sequences does not seem to be enough.

Another reason for the difference could be that Sutton is working backwards through the sequences instead of forward like my algorithm. He mentions this on page 23 saying that TD(0) is slow at propagating, but that is not a problem for sequences if they are repeatedly presented. However, if the sequence is only presented once, the algorithm will learn slower unless one starts from the end state and works backwards to the initial state. The way Sutton writes the paper leads me to believe he did not work backwards, giving it as an example why TD(0) was not optimal in experiment two, but it is still a possibility that should be pointed out.

Figure 5 shows the average error at the best α value for different values of λ. The graph has the same shape as Sutton’s which shows that λ less than 1 is far desirable in the best-case scenario. The error starts to decrease at the beginning after λ equals 0 (less visible in my graph vs Sutton’s but it still exists). Then the error starts to increase at a certain point where TD(0) is more favorable. My point where that happens is λ equals 0.3 and Sutton’s appears to be λ equaling 0.6. Again, this difference is probably due to a difference in data sets. Overall the graphs look very similar.

# Problems and Pitfalls

One problem I faced was putting equations together. It took me a while to figure out what and were supposed to be and how to simplify equation 4. He mentions definitions and simplifications much earlier so it was a lot of piecing different parts together in order to come to a conclusion.

Figure 3 threw me off for a very long time when I first read the paper. Also “100 training sets, each consisting of 10 sequences” made me think that each training set was one sequence consisting of 10 states. Most of those problems were solved by just rereading the paper (and Erratum) multiple times.

Other problems just included missing information like missing α values or epsilon. A sample training set to confirm the algorithm would have been nice. My graphs not being exactly like Sutton’s worried me until I read Piazza messages from other students encountering the same problems.

Some assumptions I made were that the initial weight values were the same across exercise one and two, in exercise one I took epsilon to be .001 since that seemed to be a reasonably small value and α to be .04 since that was the α TD(0) performed the best with.

# References

1. Sutton, R. (1988). *Learning to Predict by the Methods of Temporal Differences*. Boston: Kluwer Academic Publishers, pp.10-44, 377.