Project 1 - CS7642 | A Random Walk

Andres Miguel Menendez, amenendez6, 902969924
Email - amenendez6@gatech.edu

Commit Hash: 4cf78838bf3d895492e0b3de27fa7292aa62a938

INTRODUCTION

Machine learning involves a system that can adapt to its own experience in order to predict its own future. This means that such a system would be able to accurately evaluate its current state and thereby change its state to maximize its returns. Richard S. Sutton's paper[1] on "Learning to Predict by the methods of Temporal Differences" introduces a novel method to prediction learning that gains experience directly from temporal predictions without supervision. These intermediary temporal steps can be unique to each problem and can be weighted at different values depending on the selected approach. The problem described below ventures to explore the optimal weights for the various parameters that can be used to uniquely adapt the TD method to a problem. This problem is from Sutton's paper[1] and looks to replicate the same results in order to explore the implications and the design.

PROBLEM

To further demonstrate the use and potential of temporal difference learning, a simple bounded-walk example was used in Sutton's paper; Consider a scenario where an agent follows a random sequence of movements to the left ,or the right until it reaches the boundary of the environment. If it reaches the edge-state on the left the game ends with a reward of z=0, while if it reaches the right-most edge-state, it is rewarded with z=1. This could also be defined as a Markov Decision Process with deterministic actions where the policy's actions are stochastic in nature; ½ chance to go left, ½ chance to go right.

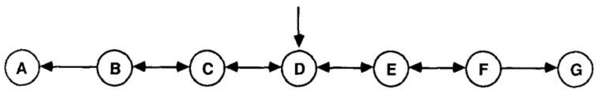


Figure 1: All agents begin at D and can stochastically transition to states to the left or right.

APPROACH AND IMPLEMENTATION

This system can be evaluated with a straightforward linear TD supervised learning method. With respect to the states [A B C D E F G] , an agent's state can be represented by $\{x_i\}$, a vector of 0's and a 1. For example, state C would be represented by [0 1 0 0 0] where a 0 is where the agent **is not,** and a 1 where the agent **is** currently at. The end states are omitted from these vectors because when the agent reaches states A or G, the game ends. The vector $\{x_i\}$ at time-step 0 is reinitialized at state D, [0 0 1 0 0]. The terminal states can be represented as z.

At each step of an episode, the method will make a prediction of what its eventual outcome could be. If its prediction is 50% confident that the outcome of that episode will result in a reward of 1, then that state's value will be 0.5. The agent will make another guess and compare it to its previous predictions to update its means of evaluation. These predictions will be based on a weight vector that acts as a memory for the learning system. Thus, these weights are what the system seeks to learn and optimize. This vector combines the value of all previous states to wind up with a prediction. Furthermore, because of the way $\{x_i\}$ is defined and given that the terminal states' rewards are z = {0, 1}, the resulting weight vector will be equal to the probability of success for the non-terminal states. The actual true value of each state can be mathematically resolved to ⅙, ⅓, ½, ⅔, ⅚ for states B, C, D, E and F, respectively. This allows for a very intuitive error assessment at any given time by the absolute difference between an episode's state-value estimation vector and the true state-values.

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

The weight vector will be updated throughout the training sets by adding a Δw_t value to the initialized weight vector. This temporal weight-update is recorded at every step but is not necessarily updated into the weight vector at each step. This is expressed in equation (1), where w is updated with all recorded updates from time t = 1 to t = m. Using the classic prototypical supervised-learning update procedure, we can begin to implement this approach where α is the learning rate, and $0 \leq \alpha \leq 1$.

$$(2) \qquad \Delta w_t = \alpha(z - P_t)\nabla_w P_t$$

In (2), we can note that due to the linear nature of the problem, $\nabla_w P_t$ simply resolves to x_t. Furthermore, the prediction, P_t, can be equated to $w^T \bullet x_t$ such that we can refactor (2) to represent the delta-rule (also known as the Widrow-Hoff rule); $\Delta w_t = \alpha(z - w^T \bullet x_t)\, x_t$. This is still a supervised learning update procedure because the update can only be evaluated when a terminal state has been reached (z). However, the error of terminal state z and a specific prediction can be evaluated as the sum of the changes in the predictions (3).

$$(3) \qquad z - P_t = \sum_{k=t}^{m}(P_{k+1} - P_k)$$

By plugging equation (3) into (2), into (1) we can deduce the principal equation, (4). Unlike (2), this method does not depend on the resolution of a sequence and can work off of temporal predictions. The only concern with this specific procedure is that as it updates the weight vector, it considers every action taken equally, no matter how recent or far from state t (this is also known as the Widrow-Hoff procedure):

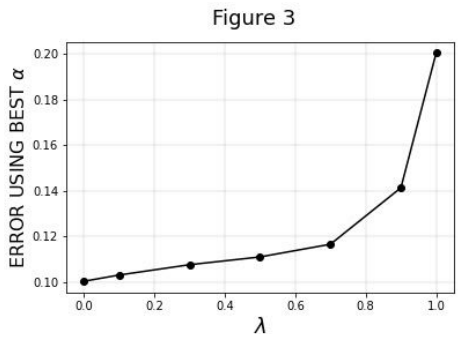
$$(4) \qquad \Delta w_t = \alpha(P_{t+1} - P_t)\sum_{k=1}^{t} \nabla_w P_k$$

To implement a more thorough TD system, we will introduce the parameter λ that will act as the horizon that the learning agent can see, where $0 \leq \lambda \leq 1$. This allows the predicting agent to weigh recent predictions more than distant predictions. Yet, when $\lambda = 1$, all steps are equally important, and inversely, when $\lambda = 0$, the agent only cares about the most recent prediction. Ultimately, λ, is how fast the system decays the assignment of credit throughout all visited states.

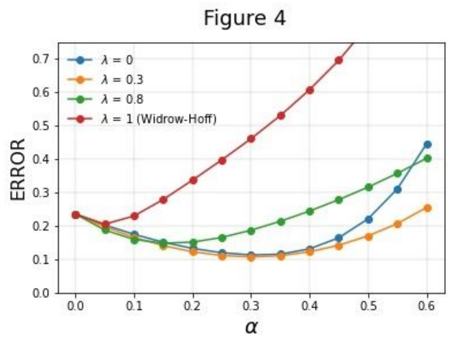
$$(5) \qquad \Delta w_t = \alpha(P_{t+1} - P_t)\sum_{k=1}^{t} \lambda^{t-k}\, \nabla_w P_k = \alpha(P_{t+1} - P_t)\sum_{k=1}^{t} \lambda^{t-k}\, x_t$$

A temporal difference lambda method (TDλ) was implemented to act as a learner of state values for the simple bounded walk. Two experiments were run with two different setups to gauge the sensitivity of the two learning parameters, α and λ, with respect to the problem at hand. Error was calculated by averaging the RMS error across several training sets. The specific methods are explored in the next section.

METHODS



In this first experiment, we observe the impact of varying lambda values over the training set of 10 episodes. A Δw value is aggregated over a training set of 10 episodes, and then used to update the weight vector after the 10 episodes. Each set of 10 episode updates were repeated until convergence was reached. The resulting errors were then averaged across 100 separate training sets for each evaluated lambda value. Each procedure was iterated across several α values such to find the optimal learning rate (approx. 0.01) for that specific lambda value (where the error was the least). The results are plotted in *Figure 3*.



The second experiment ran a training set of 10 episodes while updating the weight vector after the completion of each episode. This was run over several different learning-rates and lambda values. Each procedure averaged over 100 random training sets. Before each training set, the weight vector was initialized to [½ ½ ½ ½ ½]. Both *Figure 4* and *Figure 5*, involve the use of this setup.

RESULTS

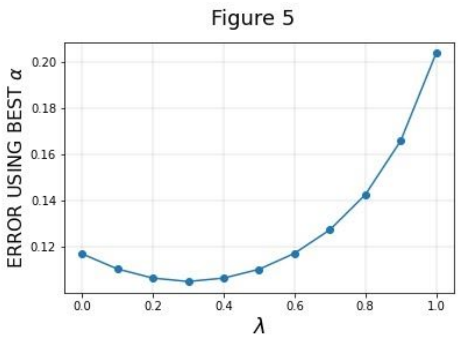


Figure 3 plots all the errors derived from methodology 1 and demonstrates the relevance of lambda for that specific setup. It is observed that the prediction error for the TD λ method increases with increasing lambda values. The learning-rates used for each point were very small and approximated to 0.01. Furthermore, the initialized weight vector did not matter as it always converged on the same values. Although the first method demonstrates a monotonically increasing relationship between error and λ, the second method instead shows that the prediction error is minimized at a λ value near 0.3. What is unanimous across both methods is that a high λ values inevitably result in high error. With respect to method 2, all errors at at $a = 0$ were equal due to the fact that resulting weight vectors did not change from the initialized [½ ½ ½ ½ ½] since the subsequent update, Δw_t, will inevitably be multiplied by 0.

ANALYSIS

Figure 3 demonstrates a unique aspect of the λ value. Weights update only after every 10 episodes which means that any early random biases arising from these the initial episodes could skew the data for that specific training set. Those early weights are iterated upon and self-driven deeper into their own bias. If we imagine an early training set that heavily favors going left, the resulting Δw after 10 episodes will be an aggregate of all the experiences. Thus, such a biased sample would return a weight vector with disproportionately high values on the left side, which would impact future learning if it isn't quickly corrected in the next few training sets. When we update the weight vector, we heavily impact the next training set because it is then multiplied into the next Δw_t through the predictions where $P_t = w^T \bullet x_t$.

Therefore, even if the next training sets have a statistically balanced training set, the left values will multiply to higher orders. This is a dangerous bias as it impacts learning by skewing future rewards. Every training set thereafter would internalize the bias of the first training set, increasing almost exponentially. What is key to note is that this is specific to *Figure 3* as the weight vector update is an aggregate of 10 episodes. This allows any biased samples to really skew the next weight vector to one side. *Figure 3* only achieves such small errors because of the incredibly small learning rate (approx .01) which slows down any biased learning. Additionally, these updated weight vectors use averages instead of entire summations of all learning.

In contrast, *Figure 5* instead updates weight vectors immediately and therefore never ends up in such a predicament. Another observation with respect to these two plots is that the error for $\lambda = 0$ in *Figure 3* is lower. This is because a TD(0) method does not learn from any prior steps except for its immediate state and prediction. Thus, the agent learned the same amount in every step in both experiments, but in *Figure 3*, the agent ran 100 training sets until convergence. In *Figure 5*, the agent instead only ran 10 x 100 episodes. This enables experiment 1's TD(0) to have lower error due to more sampling data (*Figure 3*'s).

Experiment 2 demonstrates the relationship between learning rates and λ values. What can be deduced is that there is an ideal degree of learning where the agent learns enough to make a better prediction but not too much where it gets distracted by stochastic anomalies. Generally, low learning-rates are necessary to enable an agent to converge onto a weight vector. Otherwise, the weight vector would be updated with large values that would then affect the next training set; thereby enabling the weight updates to spiral out and away from convergence (as discussed above with respect to experiment 1). As lambda increases, the summation of states in *equation (5)* can increase significantly and have a similar effect to that of a high learning-rate. Thus, it is critical to balance both parameters to achieve quick convergence and reliable error.

DISCUSSION

These results demonstrate that with the correct learning parameters, a TDλ method can quickly and effectively guide an agent to make low-error predictions. The λ element introduces the agent to a sensibility to its memory by enabling previous rewards to flow into future predictions. Choosing the right λ is key to reducing error as it can bring an agent to either be blind to important information or become overly influenced by anomalies of the past. What is clear is that this method achieves convergence quickly, consistently and effectively when paired up with the correct strategies.

Experiment 1 shows that under a high variance training method, the agent can still converge with reasonable errors. In this case, the $\lambda=0$ resulted in the smallest error. A low learning rate combined with the lower λ values stop the propagation of the biases and therefore result in the lower error. This TD(0) can also be characterized as a one-step estimator where it will estimate its outcome based on its previous move alone. With better learning strategies, a TDλ method can reach even lower errors.

The learning rate is equally important to create an effective TDλ method, and is essential to guaranteeing convergence. But as shown by experiment 2, *Figure 5*, the learning rate must be optimized to minimize error. *Figure 1* consistently has the lowest learning rates due to the troublesome sampling method discussed above, while the other experiments use higher learning rates and yet still achieve better results. This indicates that the sampling and training strategies can have an equally big impact on the agent's effectiveness and errors when paired up with appropriate λ and α values.

REFLECTIONS

Considering Sutton's original paper[1], my findings effectively replicate his conclusions. In fact, all of my figures ended up with lower error. One big difference I found during the development of these experiments is the nature of stochasticity and biased samples. As discussed in the analysis above, some of the methods were prone to being heavily distorted when the agent would find early anomalies. Proper agents should be able to appropriately deal with such artifacts, but these agents instead relied on the stochasticity of later training sets to correct it by being overly biased in the opposite direction. Although very small learning rates could partially solve these issues, it was not necessarily optimal to decrease learning rates. Instead, I selected a specific seed to predetermine the random numbers in order to avoid anomalies. Only with a controlled random number generator could I produce the same figure.

It is worth considering that in 1988, there weren't very powerful computers. It is likely that Sutton and his team were compelled to use *safe* data. It would have taken them a lot of time and money to recreate their experiment for many different seeds. Sutton's *Figure 3* only reached an error of approximately 0.19 while my figure achieved a lower error of 0.10. This may be because his convergence threshold was a lot wider than mine as computers in 1988 were slower and more costly to use.

Sutton's paper was also not very clear in several sections. One assumption I had to make was to average the weight vectors. Without averaging, some of these weight vectors would disproportionately spiral out of control and disrupt the agent's learning entirely when it would encounter even the smallest of anomalies. Sutton never explicitly mentions this. His only clue is "the Δw_s were accumulated over sequences and only used to update the weight vector" (Page , Sutton-1988). So, although I am averaging, my strategy still effectively accumulates the same data Sutton discusses.

CONCLUSION

As demonstrated by Sutton, the method TD λ can be incredibly powerful but has to be accurately optimized to guarantee reliable and quick results. This method implicitly flows all historical data into the current state to enable the agent to make more accurate predictions. The λ variable introduces a sensitivity to this working memory, and allows for the optimization of it to increase the fidelity of temporal predictions. An agent can therefore learn quicker than a supervised learning agent as it collects more data during its explorations and doesn't solely wait until the resolution of the entire training set. As λ approaches 1, it begins to value all previous states equally. As it approximates to 0, the least recent evaluations are scaled down to approximately 0. Furthermore, as demonstrated by the experiments' results, the best λ values will likely lie between 0.2 and 0.6. Therefore, considering the infinite horizon of an optimal TD λ, the agent will require a huge memory to remember all past states, even when the least recent states may be scaled down to approximately 0. An n-step TD method is similar to TD λ as it uses a working memory to leverage previous n-states to improve its predictions while limiting the amount of working memory (up to n states). Henceforth, an n-step method combined with TD λ could result in an even more effective solution to guarantee reliable learning through the TD λ method while leaning into the pragmatism behind the n-step strategy.

REFERENCES

[1] Sutton, Richard S. "Learning to Predict by the Methods of Temporal Differences." *Machine Learning*, vol. 3, no. 1, 4 Feb. 1988, pp. 9–44., doi:10.1007/bf00115009.

APPENDIX/FIGURES

Figure 3

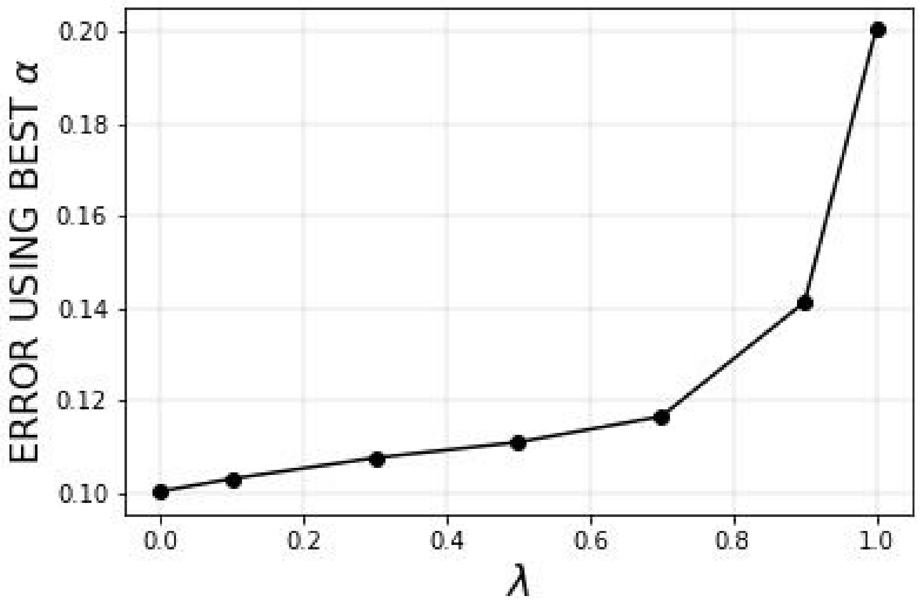


Figure 4

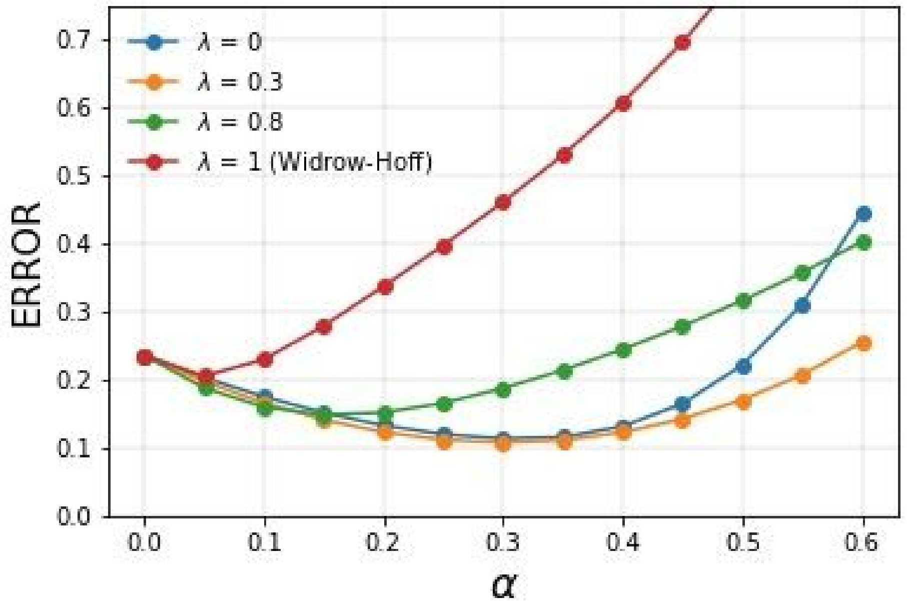
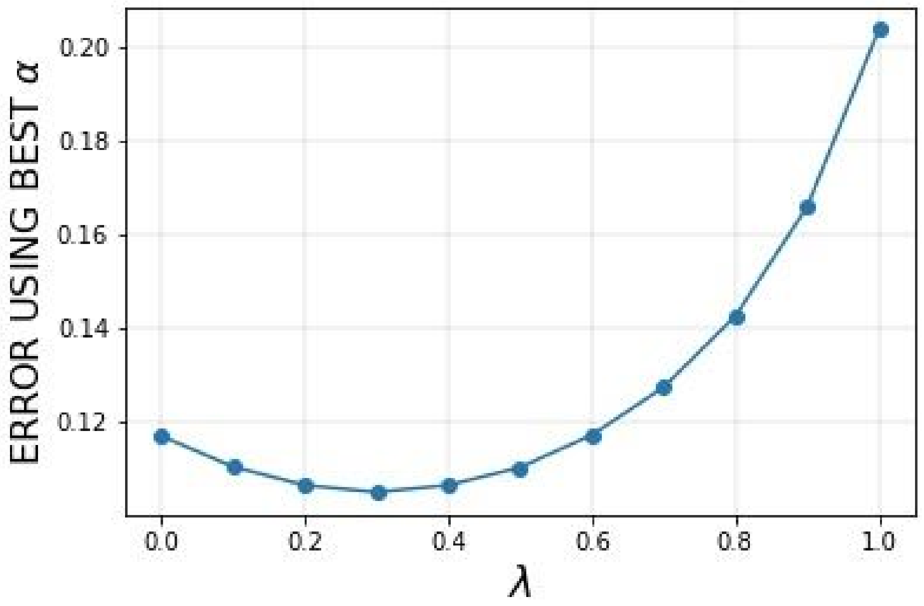


Figure 5


ERROR USING BEST α
0.20
0.18
0.16
0.14
0.12
0.0
0.2
0.4
0.6
0.8
1.0
λ