Desperately seeking Sutton

Safin Salih

CS-7642 Reinforcement Learning

Hash: 914fa5b4574b6932bb882d383322d348d1fcf759

Overview

The objective is to replicate the problem and solution that Sutton presented in his paper: *Learning to Predict by the Methods of Temporal Differences* [1] by Richard S. Sutton. In doing so, we'll analyze the results we got after following his random walk problem, possible issues we encountered , and see if our conclusion align with his: TD methods has it's advantages over traditional supervised-learning. One being that TD methods requires less memory and less computation; and two, it makes better prediction when data is limited.

Problem

The problem that Sutton presented in his paper [1] is a bounded random walk. First create a dynamical system with ordered states; A , B ,C ,D ,E , F and G. At the beginning of each sequence, start at state D, and since this is a random walk, the transition probability from its current state to its neighboring states will be %50 to the left and %50 percent to the right. This will result in changing from state to state until it reaches either terminal states; far left for State A, or far right for state G. No reward(feedback) is given unless it's one of the two terminal states.If it reaches state A , the reward we'll be 0, and if it reaches state G, the reward will be 1. Our primary goal is to predicted the expected return of each state given that we started at state D. The true values for these states are as follows: 1/6, 1/3 ,1/2, 2/3, and 5/6 for B,C,D,E and F , respectively.

Algorithm

Since this is an iterative process, We're going to need an initial weight vector, this can be initialized with random values or fixed.This weight vector will evolve over time to converge to a final weight vector, denoted as W_{final}. At each iterative step, we update our weight vector, W_{new} = W_{old} + $\sum_{t=1}^{m} \Delta W_t$. For our problem , m represents the last state in the random walk sequence, and we are accumulating all of the ΔW_t. To compute ΔW_t at time t, first pick lambda, $0 \leq \lambda \leq 1$, and a small step size α. x_i represents the state x at time i, for simplicity each state is represented as a unit vector. For example if at timestep 2 we landed on state E, we return a vector {0,0,0,1,0} for x_2. Let's look at the equation that Sutton [1] gave us to solve for TD[λ].

$\Delta W_t = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k$ [2]

Where $P_t = W^T x_t$, and $P_{t+1} = W^T x_{t+1}$

Since $P_k = W^T x_k$ = $W_{(1)} * x_{(1)k} + W_{(2)} * x_{(2)k} + W_{(3)} * x_{(3)k} \cdots$

$\nabla_w P_k = \frac{\partial}{\partial W_1} W_{(1)} * x_{(1)k} + \frac{\partial}{\partial W_2} W_{(2)} * x_{(2)k} + \frac{\partial}{\partial W_3} W_{(3)} * x_{(3)k} + \ldots$

$\nabla_w P_k = x_{(1)k} + x_{(2)k} + x_{(3)k}$

Hence, $\nabla_w P_k = x_k$

To put everything into perspective, $W \leftarrow W + \sum_{t=1}^{m} \alpha(W^T x_{t+1} - W^T x_t) \sum_{k=1}^{t} \lambda^{t-k} x_k$

For Widrow-Hoff (supervised learning) we have, $W \leftarrow W + \sum_{t=1}^{m} \alpha(W^T x_{t+1} - W^T x_t) \sum_{k=1}^{t} x_k$

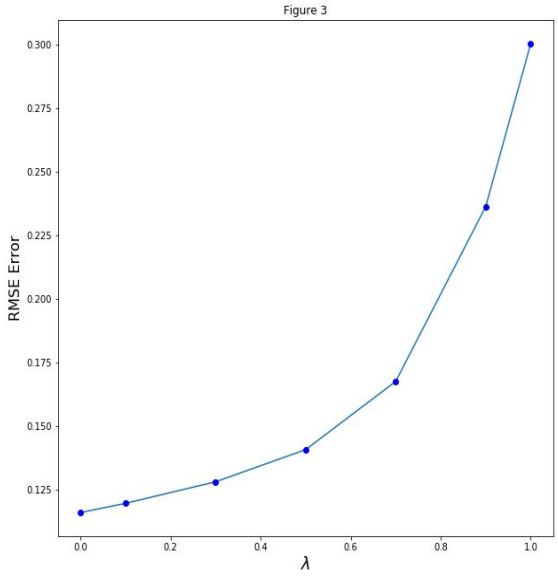
And TD[0] or one-step look ahead , $W \leftarrow W + \sum_{t=1}^{m} \alpha(W^T x_{t+1} - W^T x_t) x_t$

We can see that this part $\sum_{k=1}^{t} \lambda^{t-k} x_k$ of the equation causes the most computation depending on how large a sequence is. And TD[0] is missing that term, therefore it's faster in terms of computation.

First Experiment

To replicate the first experiment[1] , here is how I did them and the pitfalls I encountered. To recreate figure 3[1], It's important to choose a α that is smaller than 0.01, since we are dealing with convergence of an iterative algorithm, we don't want to make big step sizes in our error term . This will cause us to miss the minimum value and diverge to infinity. Similarly, if we let a step size too small for example 0.0001, it will converge, however we'll be locked in some local minimum, and consequently our final weight vector won't be that far off from where we started with .Hence, I let $\alpha = 0.06$ and $\lambda = 0$, then generate 100 training set of random walks, each containing 10 sequence. I presented 10 sequence to the learner at a time, then the learner will update it's weight vector. Do this until $|W_{old} - W_{new}| < \epsilon$ for some small ϵ(0.001) value ,this should converge . After convergence, compute root means square error between the real expected value(1/6, 1/3 ,1/2, 2/3, and 5/6 for B,C,D,E and F) and the W_{final} in the learning procedure. Then repeat this process for λ = 0.1,0 .3,0 .5, 0.7,0.9 , 1 . Given the nature of this problem's randomness, my graph didn't look exactly like the figure[]. To do so, my consensus is that you should iterate this process many times and then average each graph's RMSE datapoint, and it will look like the figure[1].

Although there is variation in the graph each time you run the code, but the results seems to be consists with what Sutton was trying to convey. Under repeated presentation of training set , TD[1] or Widrow-Hoff procedure minimizes the RMS error on the training set but doesn't take into account future results. Hence on average, λ =1 performed worse than any other λ values, and the optimal lambda value seems to exist in between $0.2 \leq \lambda \leq 0.3$.(The graph on the left is what I got after averaging over 5 different datasets):

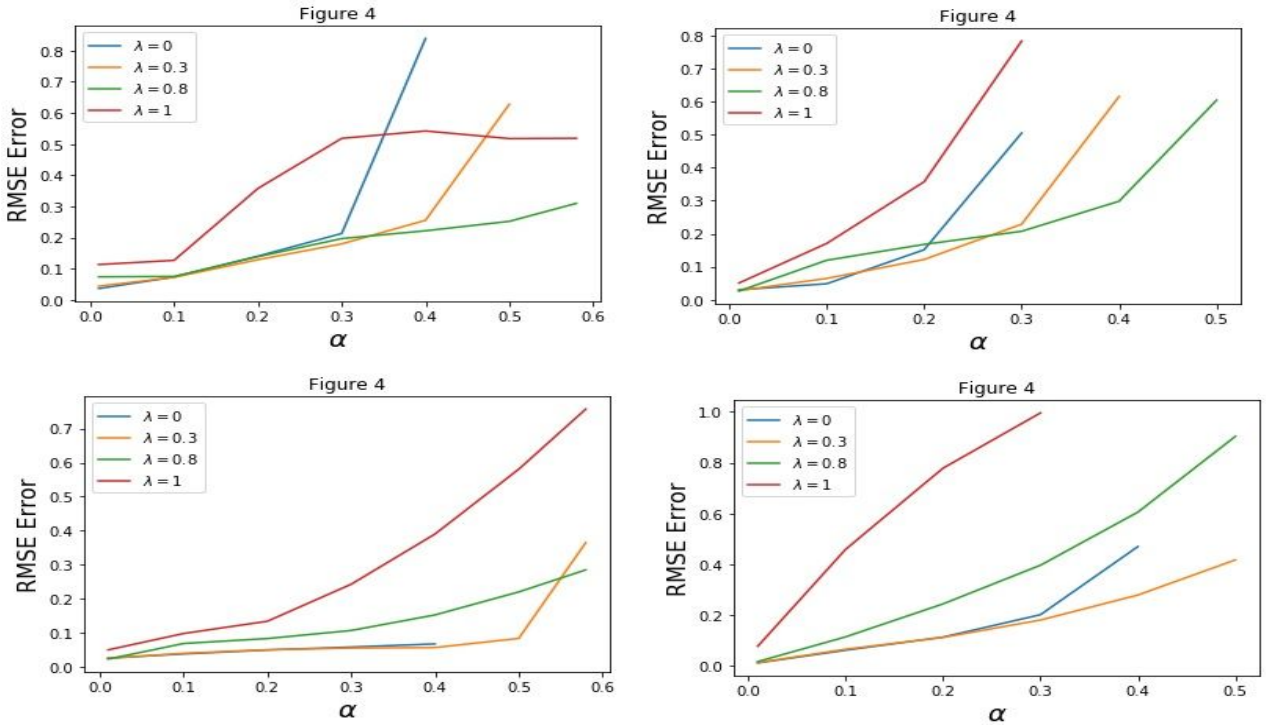


The complexity of this problem was lengthy in running time rather than taking memory space. I don't know the correct big O notation for this problem, but it's certainly not linear . If I tried this again, I would try to implement dynamic programming(computer programming method) into the code, in which I would give up more memory space in exchange of saving computational time .

Second Experiment

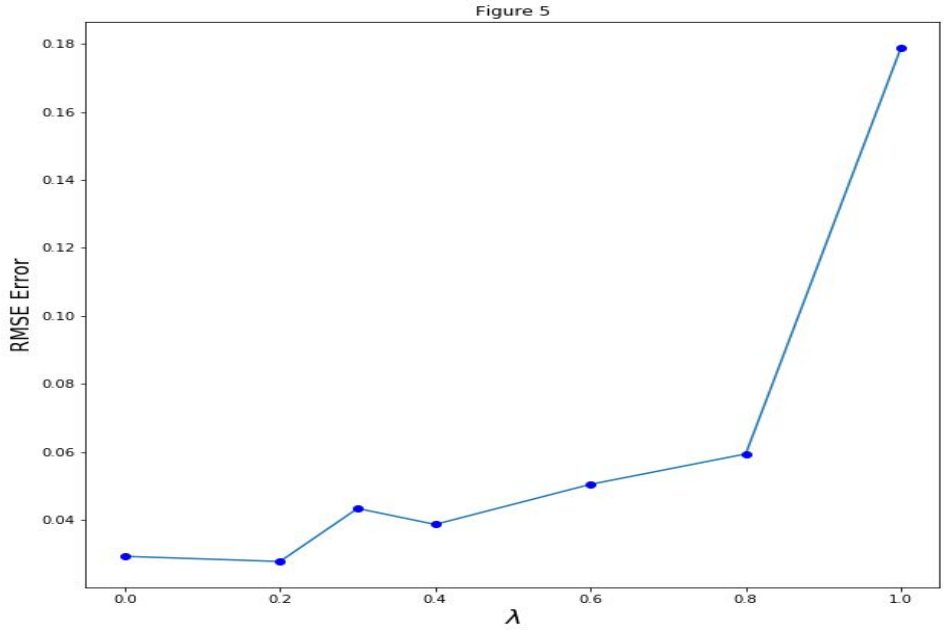
For the second experiment[1] , we raise the question of learning rates. The dataset is still the same as before, with 100 training set each set consisting of 10 random walks. The initial weight vector is {0.5,0.5,0.5,0.5,0.5}, this is to insure that either side have equal chances. The training set is presented only once rather than repeated representation like experiment one. Hence, convergences criteria isn't necessary, just one iteration over the entire 100 training set. Our focus is, given some λ, what is the optimal learning rate α for that λ.

To be precise, let L be a set of different λ's and A be a set of different α's, where L = {0,1,0.8,0.3}, and A = {0,0.1,0.2,0.3,0.4,0.5,0.6} . For a some $\lambda \in L$, we loop through every $\alpha \in A$. At each training set, we update the weight vector after each sequence, until the end of training set , then we compute RMSE of the true value and what we gathered in the learning procedure. Then we do this for the rest of the training 100 training set, and average all of the RMSE. Here is what we got for all the $\lambda \in L$



Here, you can see variation due to randomness in the dataset. However, the truth still holds, all of the other λ values (0 , 0.3, 0.8) did better than when $\lambda = 1$(Widrow-Hoff procedure). We go further and state on average, $\lambda = 0.3$ did better than the rest. One issue I encountered is when $\lambda = 0$, and $\alpha > 0.4$, the RMSE seem to explode. Which means that $\sum_{t=1}^{m} \alpha(W^T x_{t+1} - W^T x_t)x_t$ is diverging to infinity when $\alpha > 0.4$. Because TD[0] is only concerned with one step ahead to accumulate it's weight, a step size that is greater than 0.4 maybe too much, hence resulting a large RMSE error. Here is a pairing

of each λ's with corresponding best α that would yield the lowest RMSE .



Conclusion

All in all, our conclusion matched with Sutton's[1] with some variation on the graphs due to randomness. Hence, the reason why Widrow-Hoff or TD[1] did the worst on both experiments is because TD[1] only tries predict the final outcome and reduces the error on that prediction over the entire training set. Where as other TD λ values utilizes the intermediate prediction via bootstrapping onto the next value prediction making it more efficient empirically. When data is in temporal sequence, and is limited, TD methods is faster, and more efficient than supervised learning.

Reference

[1] Sutton, Richard S. 1988. "Learning to Predict by the Methods of Temporal Differences." ***Machine Learning*****3 (1): 1–25. doi:10.1007/bf00115009.**