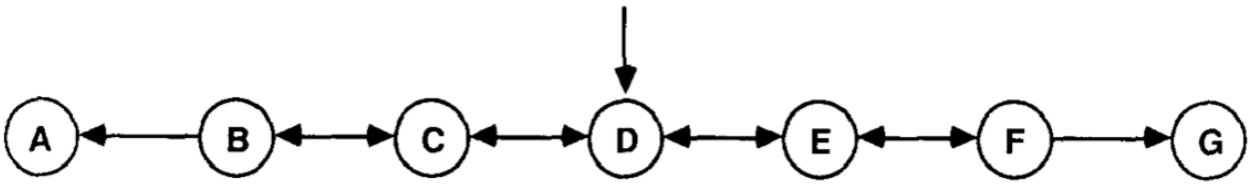
Replicating Sutton 88 TD(λ) Experiments

Huiwen Duan

I. Introduction

This paper replicated figure 3,4,5 of Sutton(1988) publication on TD(λ) learning algorithms. On a high level, I used this learning algorithm to solve for a random walk problem and compare the errors among a range of λ values and learning rates. In a random walk setting, the action taken in the next step is only dependent in the current state, and is not influenced by any prior states.

The specific random walk case we are simulating is one dimensional, as shown in the figure below. ABCDEF are reachable states in the random walk. The start state is always D. A and G are terminal states with rewards of 0 and 1 respectively. At any non terminal state, i.e. BCDEF, the direction of next step is stochastic. For example, if an agent is currently on C, it is equally likely that its next step will be B or D. However, if an agent is currently on A or G, the walk ends.



Temporal difference(TD) is a modelless reinforcement learning method. Its equation is shown in the figure below. TD learning uses experimental methods to obtain subsequent states, adjusting the weights accordingly and incorporate these weight adjustments into calculating the value of current state. For each state, it adjust weights using the current state and all previous states, giving more credit to more recent predictions (when $< 0\lambda <1$). By altering the value of λ, we change how much credit we give to each of the previous states. According to the random walk set-up, the true probabilities for each of the non-terminal states are $\left[\frac{1}{6} \quad \frac{1}{3} \quad \frac{1}{2} \quad \frac{2}{3} \quad \frac{5}{6}\right]$ for states BCDEF(1, p.14). I implemented an algorithm based on this equation to estimate the weights for intermediate states.

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

II. Methods

A. Generating random walk training set

To simulate one run of random walk in python, I first created an array to record the states and initialized it with 3 as the only element, as the walk always starts at D. The numerical representation of the letter states is as follow: A:0, B:1, C:2, D:3, E:4, F:5, G:6. Then a random float in the range of [0,1) is generated using numpy's random number generator. The next state will be the current state number minus one if this random number is smaller than 0.5, and plus one if the random number if bigger than or equal to 0.5. Repeatedly, we move forward to the next state until we reach 0 or 6, which represents state A and G. All states are recorded into the states array. The training data set is a list of numpy arrays generated from the this method. In addition to this, I also created a method for generating balanced data sets, which ensures that a data set has equal number of sequences that end with 0 and 6.

B. Implementation of TD learning

This section details the algorithm implementation for using TD learning method to update weights based on one random walk sequence. First of all, we convert the intermediate states into vector representations and combine them into one matrix

to facilitate the later matrix multiplications. For example, the vector representation of state D - x_D is $\begin{bmatrix}0\\0\\1\\0\\0\end{bmatrix}$. If a random walk sequence is DCDEFG, then the matrix representation for the intermediate states is $\begin{bmatrix}0&0&0&0\\1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$. The algorithm loops through the intermediate states. At each state, we calculate the the delta weights using Sutton's equation. Alpha is an arbitrary pre-determined learning rate. Pt is the weight of the state at time t. λ is an arbitrary number belonging to [0,1]. There are two special cases. When $\lambda = 1$, this implementation is the same as supervised-learning method, in other words, the Widrow-Hoff procedure. When $\lambda = 0$, the weight change is only determined by the most recent state. $\nabla_w P_k$ is the vector representation of the state at time k. Therefore the summation part of the equation can be calculated as the matrix product of the lambda vector and previous states matrix. $\begin{bmatrix}x_0\\x_1\\...\\x_t\end{bmatrix} \times \begin{bmatrix}\lambda^{t-1} & \lambda^{t-2} & ... & \lambda^0\end{bmatrix}$. This matrix form of summation drastically reduced execution time. After all the intermediate states have been updated and used to update delta weights, the algorithm returns delta weights.

C. Experiment one

The first experiment utilized the paradigm of repeated representation to train the weights. There are 100 training sets and each of them contains 10 random walk sequence. I initialized the weights vector to $\begin{bmatrix}0.5 & 0.5 & 0.5 & 0.5 & 0.5\end{bmatrix}$. The weights vector is only updated after a complete representation of a data set. The same data set is repeatedly used for training until the weights vector converge. I defined convergence as the change in weights is lower than 1^{-5}. Then the root mean squared error (RMSE) is calculated between the weight estimates and the true probabilities. $RMSE = \sqrt{\frac{\sum (w_{i,estimate} - w_{i,true})^2}{5}}$. Before training a new data set, the weights vector is reset to $\begin{bmatrix}0.5 & 0.5 & 0.5 & 0.5 & 0.5\end{bmatrix}$. The error is average RMSE over 100 training sets. The learning rate is fixed at 0.01. The relationship between the λ values and resulting RMSE is shown in Figure 1, with Sutton's experiment results plotted in Figure 2.

One pitfall I run into was mistakenly resetting the weights vector after each representation of a data set, but the correct way was to reset it after repeated representations and the weights converged, in preparation of training on a new data set. This error caused a unrealistically long execution time, as the algorithm was only trained once on each data set and could not converge.

D. Experiment two

Different from experiment one, the second experiment updates the weight estimations after training on each sequence, and feed the updated weights into the algorithm again for training the next sequence. We only train on each data set once and then get the resulting RMSE on this data set. Similar to the first experiment, we average out the RMSE over 100 data set. There's no longer a fixed learning rate, and instead, a range of learning rates and λ values are used to compare their resulting errors. I used 61 α values between 0 and 0.6 with an incremental step of 0.01, and 11 lambda values between 0 and 1 with an incremental step of 0.1. Based on Sutton's figure, he used α alpha values between 0 and 0.6 with an incremental step of 0.1. I enriched it for a more granular understanding.

III. Results

A. Experiment one, Sutton 88 figure 3

The results from my implementation is shown in Figure 1, with Sutton's original graph on the right. I chose the same set of λ values as Sutton did and plotted the RMSEs against them. The Widow-Hoff method, or TD(1) learning, resulted in the highest error. TD(0) works in a similar way as a supervised learning does whose training pairs are states and next state predictions, and has the lowest error. When $\lambda < 0.7$, the error increases slowly as λ increases, whereas when $\lambda > 0.7$, the error quickly climbs up and reaches maximum at $\lambda = 1$

While the shape of the line is extremely similar in two graphs. The range of RMSE differs. My RMSE values are within [0.13,0,19] but Sutton's are in the [0.21,0.25] range. There are two major reasons behind this difference. First of all, Sutton did not specify the α value he used for training, whereas I experimented a few α values and chose the one generating the least magnitude of error. As proven in Fig 45, the learning rate can impact the error. Secondly, the paper did not specify the method for generating the training set. It is unclear whether their random number generation in the year of 1988 is "random" enough and whether their training data sets were representative. An example of unrepresentative, in other words, unbalanced

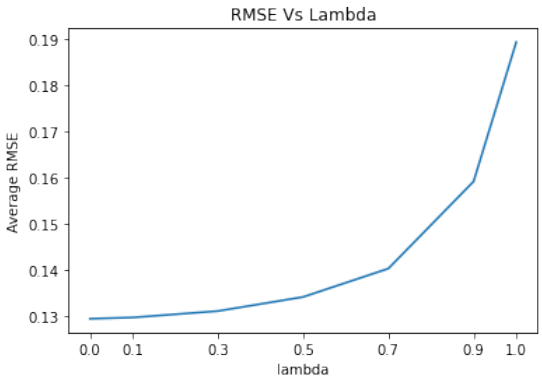


Fig. 1. My Figure 3

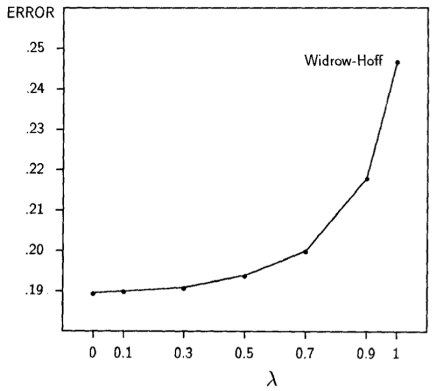


Fig. 2. Sutton's Figure 3

training set can be one with 9 sequences ending at state A and only one ending at state G. In this case, the algorithm does not have enough opportunity to learn the "G" ending sequences and therefore its predictions will derail from the true values.

To further investigate the relationship between training set and accuracy. I replicated this experiment with balanced data sets. In each data set, there are 5 sequences that end at A and the other 5 end at G. The resulting graph is shown below. The error range boundaries are reduced by half as compared to training on unconstrained data. This further shows that whether the data is representative has an important influence to the error range.

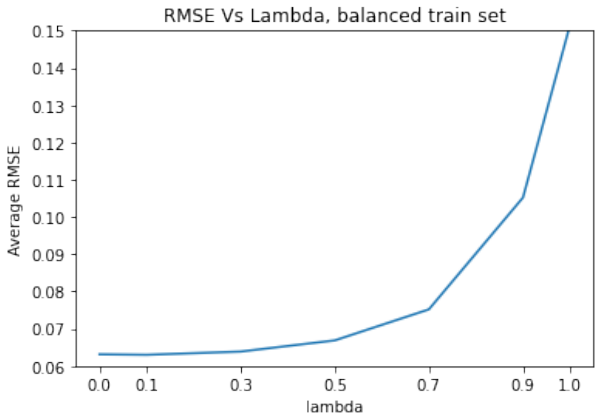


Fig. 3. My figure 3, Balanced Train Set

B. Experiment two, Sutton 88 figure 4

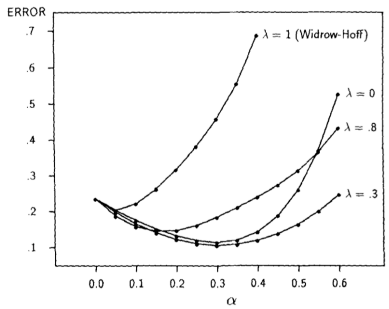


Fig. 4. Sutton's figure 4

Sutton's figure 4 plotted out the errors for different values of λ and α, with the x-axis differentiating the α values and each line representing a λ value. My implementation of this figure is shown in fig 5. For all the λ values, intermediate α values

had better errors. When the learning rate reaches around 0.4, both TD(1) and TD(0) start to blow up, whereas TD(0.3) and TD(0.8) had fairly reasonable errors when learning rate is beyond 0.4.

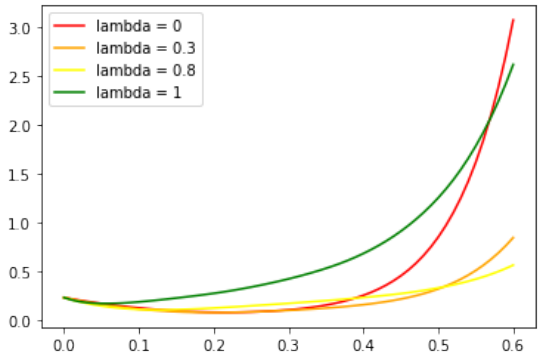


Fig. 5. My implementation, α range [0.01,0.6]

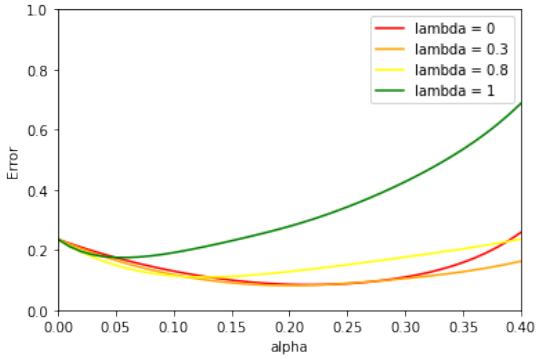


Fig. 6. My implementation, α range [0.01,0.4]

For each λ value, the shape of the line in my figure is similar to Sutton's. However, the layout and relative position of the lines are different how they are on Sutton's figure. Specifically, when $\alpha > 0.55$, the TD(0) line overtakes the TD(1) line. When $\alpha > 0.55$, TD(0.3) overtakes TD(0.8). These are not found in Sutton's figure. Confused at first, but a closer look at Sutton's figure revealed that the TD(1) errors for learning rate bigger than 0.4 are not plotted out. Based on the observed trend in his plot, it is possible that his the TD(0) line overtakes the TD(1) line at some bigger α value. Therefore, I re-plotted this figure with 0.4 as the maximum α, as shown in Fig. 6. The line shapesas well as their relevant positions in this plot and those in Sutton's figure are quite alike.

C. Experiment two, Sutton 88 figure 5

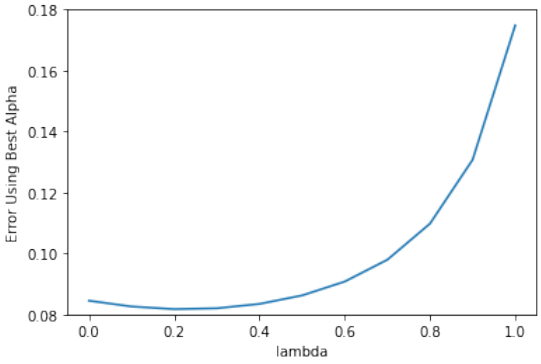


Fig. 7. My Figure

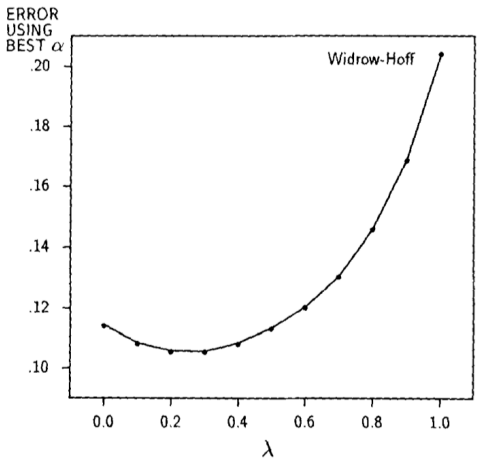


Fig. 8. Sutton's Figure 5

Sutton's figure 5 plotted out the errors for different values of λ using the best $alpha$. My figure has the same shape as his, but in a lower range of error values. The error decreases from TD(0) to TD(0.2), and increases from TD(0.2) to TD(1).

REFERENCES

[1] Richard Sutton, *Learning to Predict by the Method of Temporal Differences*, Machine Learning 3, Aug. 1988, pp. 9–44.