Reinforcement Learning and Decision Making

Project 1: Temporal-Difference (Lambda)

git hash 384cdd046126ca764258141844cce66c125c212e

***Abstract*—This document duplicates Figures 3, 4 and 5 from Sutton [1] and analyzes Temporal-Difference reinforcement learning technique discussed in the paper.**

I. Introduction

Supervised learning algorithms associate the input with the output. However, they do not take advantage of the sequential behavior of the input data. Sequential behavior can be temporal such as Saturday weather forecasts from Monday, Tuesday, Wednesday etc. For these kind of problems, temporal difference (TD) methods can be used instead of sequential behavior. TD methods make use of Monday's data to estimate Saturday's weather from Tuesday. Further, supervised learning techniques can only be trained when the true output is available, which is Saturday's weather in this example. We cannot train the supervised learning model before Saturday. On the other hand, TD is good for online problems like this as it can be trained as the data becomes available day-by-day. Further, TD uses only a few concentrated data from the previous timesteps. Therefore, it uses less memory space and computational time. On the other hand, supervised learning needs all the data and creates large matrices with derivatives, and it uses all the data at the same time. This requires a lot of memory space and peak computational time. TD is highly recommended for sequential input data instead of supervised learning. Sutton [1] also shows that TD is more accurate than supervised learning. Therefore, the advantages of TD compared to supervised can be summed up as the following:

1. TD can be implemented online.
2. TD requires less memory space and less peak computation time.
3. TD is more accurate than supervised learning.

In the following sections, I first derive the TD method. Then, I introduce the example problem and the two experiments carried out at [1]. Then, I plot the Figures 3-5 from [1]. Finally, I give the conclusions.

II. Methods

Below, I derive TD from supervised learning. Assume an input data consists of a vector of $\boldsymbol{x}_t$ [1](temperature, humidity, winds etc.) for each timestep t (Monday, Tuesday, Wednesday etc.). Assume each $\boldsymbol{x}_t$ outputs a scalar P_t for Saturday's weather forecast. Monday outputs P_1, Tuesday outputs P_2 etc. by using a linear correlation $P_t = \boldsymbol{w}^T\boldsymbol{x}_t$. Here, the transpose of column vector w (weights) is multiplied with the column vector of x. Note that w does not have t subscript which means that the same weights are used for each sequential input. The following equation can be used to update the weights,

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \sum_{t=1}^{m} \Delta \boldsymbol{w}_t \tag{1}$$

where m is the size of the temporal sequence. From the supervised learning update procedure, change in weights can be calculated from

$$\Delta \boldsymbol{w}_t = \alpha(z - P_t)\nabla_{\boldsymbol{w}} P_t \tag{2}$$

where α is the learning rate, z is the true outcome (Saturday's weather as scalar), P_t is the estimate of Saturday's weather at day t. $\nabla_{\boldsymbol{w}} P_t$ is partial derivative of P_t with respect to $\boldsymbol{w}$. Assuming $P_t = \boldsymbol{w}^T\boldsymbol{x}_t$, Eq. 2 gives the Widrow-Hoff method [2]. Equation 2 also indicates that, for supervised learning, we need to wait until z is available to train the model. However, TD learning introduces a new model approach which can be implemented online. z-P_t term used in Eq. 2 can be written as the following:

$$z - P_t = \sum_{k=1}^{m}(P_{k+1} - P_k) = P_2 - P_1 + P_3 - P_2 + P_4 - P_3 + \cdots + P_{k+1} - P_k \tag{3}$$

where P_{k+1}=z and all the other P's are estimations coming from timestep t. This way, we can start to make TD work as soon as two sequential data are available. Combining equations 1, 2 and 3 gives

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \sum_{t=1}^{m}(\alpha \sum_{k=t}^{m}(P_{k+1} - P_k)\,\nabla_{\boldsymbol{w}} P_t) \tag{4}$$

When m = 3, this nested summation term can be expanded as

$$\alpha\nabla_{\boldsymbol{w}}P_1(P_2 - P_1 + P_3 - P_2 + P_4 - P_3) + \alpha\nabla_{\boldsymbol{w}}P_2(P_3 - P_2 + P_4 - P_3) + \alpha\nabla_{\boldsymbol{w}}P_3(P_4 - P_3) = \alpha(P_2 - P_1)\nabla_{\boldsymbol{w}}P_1 + \alpha(P_3 - P_2)(\nabla_{\boldsymbol{w}}P_1 + \nabla_{\boldsymbol{w}}P_2) + \alpha(P_4 - P_3)(\nabla_{\boldsymbol{w}}P_1 + \nabla_{\boldsymbol{w}}P_2 + \nabla_{\boldsymbol{w}}P_3)$$

which is equivalent to,

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \sum_{t=1}^{m}[\alpha(P_{t+1} - P_t)\sum_{k=1}^{t}\nabla_{\boldsymbol{w}}P_k] \tag{5}$$

I wrote this equation in detail as it was not clear whether the summation terms were independent or nested, and how the summations changed places. Writing out the equation simplifies understanding the derivations. Note that, for TD learning, we do not need to use any input data from previous timesteps. Therefore, that data doesn't need to be stored or computed. This is another advantage of TD learning on top of the ability of being implemented online or offline. Equation 4/5 is mathematically equivalent to supervised learning and gives the exact same result as supervised learning.

[1] It was confusing to differentiate scalars and vectors at [1]. In this paper, all vectors are written in bold to make the equations more readable. All the vectors are column vectors.

This implementation is also called TD(1) as explained below. One can notice that every timestep information is given the same importance. For example, if we are estimating Saturday's weather forecast, should we give the same importance to Monday's forecast and Friday's forecast? Of course, Friday's Saturday forecast is more accurate than Monday's Saturday forecast. Therefore, TD(λ) is used to geometrically increase the importance of the recent data with respect to previous day's estimations. TD(λ) modifies the weight update as

$$\Delta \boldsymbol{w}_t = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_{\boldsymbol{w}} P_k \qquad (6)$$

λ is a term between 0 and 1 that geometrically decreases the importance of the previous estimations. For example, as *t* is increasing, we are getting closer to end date (Saturday), and the exponent term increases, reducing the impact of a timestep with lower *k* value.

Equations 5 and 6 can be combined as

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \sum_{t=1}^{m} [\alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_{\boldsymbol{w}} P_k] \,. \qquad (7)$$

For *m* = 3, the summation term can be expanded as

$$\alpha(P_2 - P_1)\nabla_{\boldsymbol{w}} P_1 + \alpha(P_3 - P_2)(\lambda \nabla_{\boldsymbol{w}} P_1 + \nabla_{\boldsymbol{w}} P_2) + \alpha(P_4 - P_3)(\lambda^2 \nabla_{\boldsymbol{w}} P_1 + \lambda \nabla_{\boldsymbol{w}} P_2 + \nabla_{\boldsymbol{w}} P_3).$$

From the expansion, it can be seen that the first timestep's derivative information is used in all the proceeding timesteps. This could be giving too much emphasis on the first day, which is the opposite of what we want. Therefore, I evaluated the following expansion:

$$\alpha(P_2 - P_1)\lambda^2 \nabla_{\boldsymbol{w}} P_1 + \alpha(P_3 - P_2)(\lambda^2 \nabla_{\boldsymbol{w}} P_1 + \lambda \nabla_{\boldsymbol{w}} P_2) + \alpha(P_4 - P_3)(\lambda^2 \nabla_{\boldsymbol{w}} P_1 + \lambda \nabla_{\boldsymbol{w}} P_2 + \nabla_{\boldsymbol{w}} P_3).$$

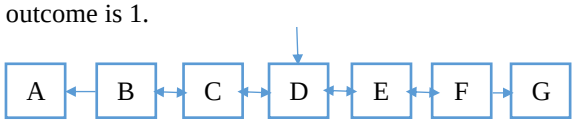
This would be equivalent to

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \sum_{t=1}^{m} [(P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{m-k} \nabla_{\boldsymbol{w}} P_k] \,. \qquad (8)$$

Eventually, I decided to go with Eq. 7 because P_2-P_1 is largely ignored by the latter version. Also, the latter version cannot be implemented online as *m* is not known beforehand. I plot the result of Eq. 8 later in the experiments section. Finally, TD(λ) indicates that a TD algorithm with the selected λ is used. In the next section, we will also study the effects of λ and α on accuracy.

III. Experiment

In this project, we implement the random-walk problem which is explained as follows. Consider a problem where every sequence starts from D. The next step can only be a neighbor on the left or right. Every sequence must end at A or G. There is no limitation on the number of steps. It is possible to go back and forth, and probability of left movement is 0.5 as well as right movement. If the sequence ends at A, the outcome z is 0. If the sequence ends at G, the outcome is 1.



P_t is calculated at the intermediate steps B, C, D, E and F as $P_t = w^t x$. x is a unit vector that is different for every step. For A, x = [1,0,0,0,0], for B, x = [0,1,0,0,0] etc. This makes $P_i = \sum_1^5 w_i x_i = w_i$ for i = 1,..,5, and i can be mapped to a location. For example, i = 1 and 2 indicate B and C, respectively. Gradient calculation is very simple: $\nabla_{\boldsymbol{w}} P_t = \boldsymbol{x}_t$ because *x* is non-zero only at *t*. Note that $\boldsymbol{x}_t$ is a vector while x_i is a scalar. For the random-walk problem, Eq. 7 becomes

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \sum_{t=1}^{m} [\alpha(w_{t+1} - w_t) \sum_{k=1}^{t} \lambda^{t-k} \boldsymbol{x}_k] \,. \qquad (9)$$

The above equation can be thought of as two nested loops. First, the unit vectors $\boldsymbol{x}$ are multiplied with λ and then summed from time = 1 to time = *t*. This summation gives a vector consisting of non-zero values for all the locations that have been visited. The locations were visited earlier have a smaller value depending on λ. Then, this summation vector is multiplied with a scalar that is $\alpha(w_{t+1} - w_t)$.

Now, I implement two experiments as described at [1]. The purpose of the first experiment is to show the effect of λ on accuracy. I had noted that TD is more accurate than supervised learning according to [1]. We already know that TD(1) is equivalent to supervised learning. Therefore, we expect to see that TD(λ) is more accurate than TD(1).

First experiment generates 1000 sequences of paths. First, the seed is fixed for reproducibility. I was confused on the length and structure of each sequence. Later, I learned that in a bounded Markov Decision problem, length of the sequence is finite and the sequence ends at a terminal state. Each is sequence is generated the following way.

1. Start at D.
2. Randomly go left or right with 0.5 probability.
3. Stop when A or G is reached.

Weights are initialized to 0.5. The weights are updated by using Eq. 7. One confusion I had at this moment was updating the weights by using all the training sets. Sutton notes that he divides the data into 100 training sets with 10 sequences each. My misunderstanding was that I thought we need to sum the weight changes for 10 sequences and then update the weights. The new weight would be used in the next training set and so on. However, the student community corrected me that each training set is trained separately and independently from each other. The correct way is to sum the weight changes for the whole training set (10 sequences), use the summation of weight changes to update the weight. Now, the new weights are used in the same training set again. This continues until the weights converge. My criterion here is that the maximum of the absolute of weight changes is less than 0.001. I have tried different epsilon values for convergence, and a stricter criterion slows down the training without increasing the accuracy significantly. Therefore, an epsilon value of 0.001 is a good balance between accuracy and speed.

I used the following pseudocode to update the weight for each sequence:

```
P_t = w_t  for ∀t ϵ [B,C,D,E,F].
P_t = 0 if t = A.
P_t = 1 if t = G.
Δw = 0 for i = 1,…,5
start t loop between [1,m]:
   vector = 0 for i = 1,…,5
   start k loop between [1,t]:
      state ← k
```

```
        vector[state] = vector[state] + λ^(t−k)
        Δw = Δw + vector × (P_(t+1) − P_t)
Δw = Δw × α
return Δw
```

This pseudocode is used to update the weight for a single sequence. I used a learning rate of 0.02 for the experiment 1. As long as the learning rate is small, the results should converge to the same value. These weight updates are summed for the whole training set. Then, the weights are updated and the whole training set is repeated with new weights until $\boldsymbol{w}$ converges. This procedure is repeated independently for all the training sets.

Once the weights are calculated for 100 training sets, they are compared to the correct results $\boldsymbol{p} = [\frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{5}{6}]$. While the correct results for the weight can be found analytically (see next paragraph), they also represent the probability of ending in G. For example, at position B, the probability of ending up at G is 1/6. The final accuracy is calculated by taking the mean of root mean square error between $\boldsymbol{w}$ and $\boldsymbol{p}$: $\sum_{k=1}^{100} \sqrt{\frac{\sum_{i=1}^{5}(w_{ik}-p_{ik})^2}{5}} / 100$.

Here, I show the calculations of the ideal weights as shown in [1]. Sutton [1] shows that [2]

$$\boldsymbol{p} = \left(\bar{\bar{\boldsymbol{I}}} - \bar{\bar{\boldsymbol{Q}}}\right)^{-1} \boldsymbol{h} \tag{10}$$

Where $\boldsymbol{p}$ is the ideal weights, $\bar{\bar{\boldsymbol{I}}}$ the identity matrix, $\bar{\bar{\boldsymbol{Q}}}$ is the transition matrix. Transition matrix contains the probabilities of moving from one state to another. However, the rows do not have to sum up to one because the terminal states are not included. Therefore,

$$\bar{\bar{\boldsymbol{Q}}} = \begin{bmatrix} 0.0 & 0.5 & 0.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.5 & 0.0 & 0.5 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.0 & 0.5 & 0.0 \end{bmatrix}$$

$h_i = \sum_{j \in T} Q_{ij} z_j$ where T is the list of terminal states e.g. [A,G]. z_j is the reward received for that state. Considering the only transition probability that leads to a non-zero reward is from F to G, h becomes a column vector of $[0,0,0,0,1]^T$.

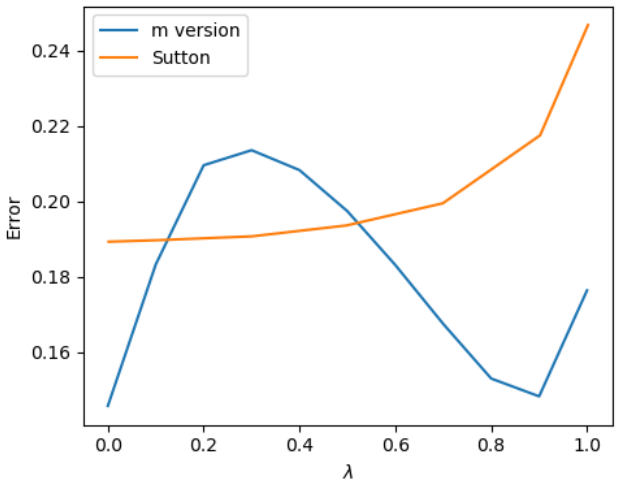


Fig. 1. Reproduction of Figure 3 from [1] by using Eq. 8. Orange line represents the digitized version of Fig. 3 from [1].

IV. Results

Fig. 1 plots the experiment 1 by varying the λ values. The blue line uses Eq 8. It is called the "m" version because the exponent in Eq 8 uses "*m*" instead of "*t*". Orange line shows the digitized version of Fig 3 from [1]. Please note that there is an erratum at the end of Sutton's paper [1]. Obviously, there is no match of trends between the orange and blue lines. Therefore, I decided to use Eq. 7 instead of Eq 8.

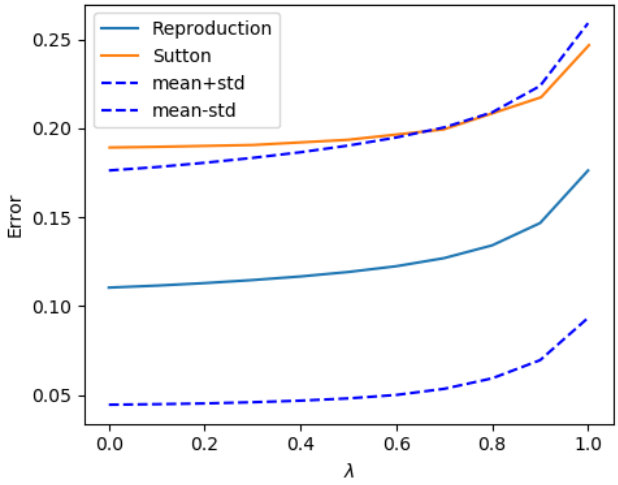


Fig. 2. Reproduction of Figure 3 from [1] by using Eq. 7. Orange line represents the digitized version of Fig. 3 from [1]. The blue dashed lines are the means +/- std of the reproduction.

Fig 2. shows the reproduction of Fig. 3 from [1] by using Eq. 7/9. This is the correct version as discussed before. We can see that my reproduction has the same shape as Sutton's. I also calculate the standard deviation for 100 RMSE's at every λ . I have noticed that the std is quite significant. Therefore, the results will be different based on the training set used. Fig 2. shows that my reproduction matches almost exactly with Sutton's plot when the std is added to it.

From Fig 2, we can also see that TD(1), which corresponds to supervised learning and Widrow-Hoff, is the least accurate both for Sutton's paper and my implementation. In fact, the most accurate is when $\lambda = 0$. Sutton, in his paper, proves that, for an absorbing Markov State, TD(0) goes to the optimal result while TD(1) goes to the suboptimal result under finite data. Under infinite data, they both converge to the ideal solutions. Here, Markov State means that the information of the whole path can be represented by a neighboring state. "Absorbing" means that there are no indefinite sequences i.e. the sequence definitely reaches a terminal state.

One perspective to look at why TD(0) performs better than TD(1) is, TD(1) minimizes the error in the training set without focusing on the future experience. For example, in our weather forecasting example, TD(1) would give equal importance to Monday, Tuesday, Wednesday, Thursday and Friday to estimate Saturday's weather. On the other hand, TD(0) only considers Friday's observation to estimate Saturday's weather. Therefore, it should be expected that TD(0) performs better than TD(1) under a finite number of data.

[2] Matrices are expressed as bold and with double bars on top.

Next, Sutton [1] analyzes the effect of learning rate on convergence. A similar experiment - experiment 2 - is carried out, with minor changes. This time, the weight is updated without iterating over all the training set, meaning that the next sequence is iterated using the updated weight coming from the previous sequence. Convergence of the weight parameters is not aimed for, only one iteration is carried out. Various values of learning rate are used for each λ. They also initialize the weights with 0.5 to give equal probability in the left and right direction. This is important if only one iteration is carried out. For experiment 1, initialization of weights didn't matter as long as the learning rate is small.

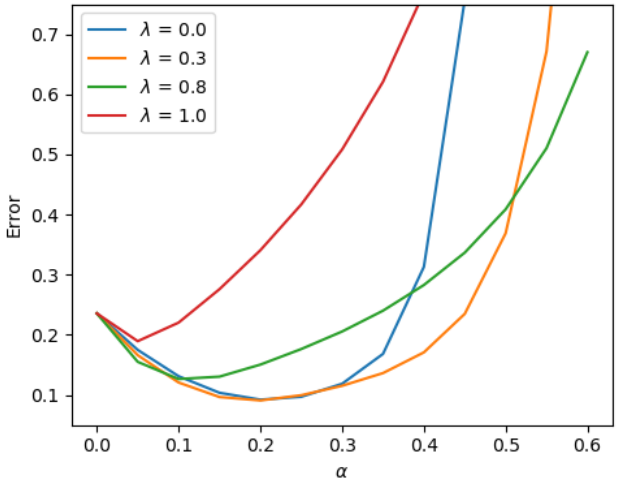


Fig. 3. Reproduction of Figure 4 from [1].

Figure 3 replicates the Fig 4 from [1]. We can see that $\lambda = 1$ gives the largest error in consistent with Sutton. Similar to Sutton, $\lambda = 0$ does not seem to be giving the best results. Intermediate values of 0.3 and 0.8 seem to be performing better for most of the learning rate range. This is also in consistent with Sutton's findings.

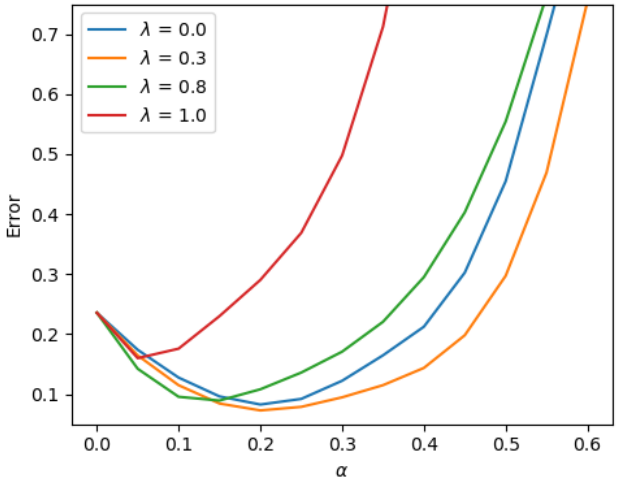


Fig. 4. Reproduction of Figure 4 from [1] by using 10 training sets each containing 10 sequences.

I also noticed that TD(0) error goes above 1 at large learning rate where this does not occur for Sutton. Furthermore, TD(0.8) gives larger error than TD(0) for most of the learning rate ranges ($\alpha <= 0.55$) for Sutton, whereas in my case TD(0) gives less error only when $\alpha < 0.4$. In Figure 4, I show that these differences occur because of the difference between the training sets. A training set with 10 sequences is too small to represent the maximum-likelihood portfolio of all the probabilities. Therefore, we are going to see significant differences of weight values for each training set which will affect rmse. In Fig 4, I only use 10 training sets with 10 sequences. You can see that Fig 4 is more similar to the findings of Sutton [1]. Every time I run the code, it gives a slightly different plot due to the probabilistic nature of the algorithm.

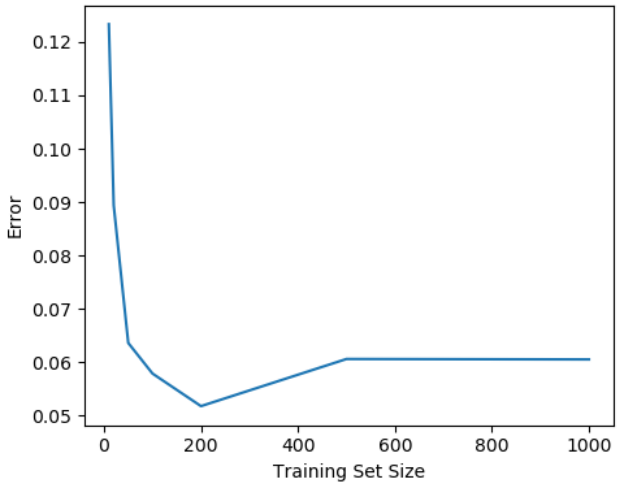


Fig. 5. Error declines as the training set size increases.

In order to show that training set size is important to reach the ideal weights, I use 10000 sequences. I vary the size of training set between 10 and 100. The number of training sets is 10000 divided by training set size. RMSE is calculated for each training set and the mean of RMSE is calculated along the training sets. Each training set is trained separately. Each sequence takes advantage of the weights improved by the preceding sequence. The iterations continue until weight converges. Figure 5 shows that as the training set size increases, the error decreases. We should not expect a monotonously decreasing line because there are still probabilities of bad training sets. However, the error declines overall with the training set size because the training set starts to become more representative of the problem.

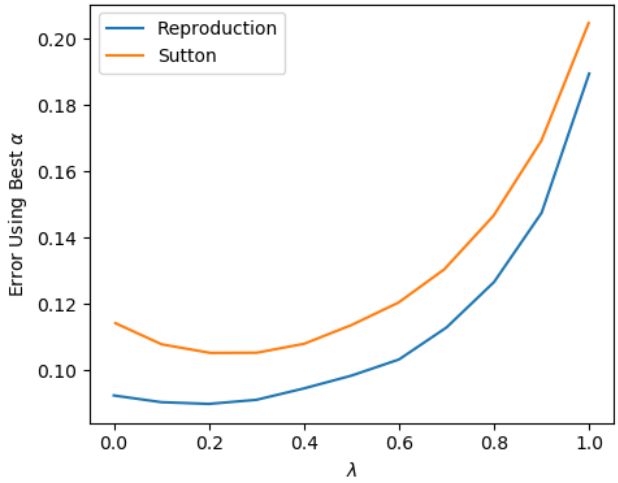


Fig. 6. Reproduction of Figure 5 from Sutton [1].

Finally, Fig 6 shows my reproduction of the Fig 5 from Sutton [1]. Basically, only the error using the best learning rate from experiment 2 is plotted along the λ axis. The results are almost exact. Orange line is that of Sutton and the blue line is my reproduction. As explained before, the slight differences are due to the probabilistic nature of the problem and finite training set size.

Figure 6 also shows that TD(1) gives the largest error. TD(0) gives very small error but the error is smallest for learning rate of 0.2-0.3. Sutton explains that TD(0) is not able to propagate all the information forward at one step whereas intermediate steps combine the information from all states. Sutton recommends propagating timeseries backwards rather than forward in order to reduce the error of TD(0) further.

V. Conclusion

In this report, I have replicated the Figures 3-5 from [1] correctly and explained the differences. I also explained the shapes of the plots. For example, TD(0) gives less error than TD(1). The differences between my plots and Sutton's plots are due to the probabilistic nature of the sequences and insufficient size of training sets.

I have derived the equations for TD starting from supervised learning. I have explained that TD is better than supervised learning for timeseries data where the sequential properties of the data may help to better estimate the output. Additionally, I have shown why TD can be implemented online while supervised learning can only be implemented offline. Further, TD methods is more efficient in terms of memory space and computational time because it only makes certain calculations at a time. The results of those calculations can be condensed into data that can be used in the next timesteps.

Further, I have shown the equations to derive the ideal weights from analytical calculations. I have further shown that TD(0) converges faster and gives the optimal results when there are only limited data. For infinite data, both TD(0) and TD(1) converge to ideal weights.

References

[1] R. Sutton, "Learning to Predict by the Method of Temporal Differences" Machine Learning, vol. 3, pp. 9-44, 1988.

[2] B. Widrow and M.E. Hoff, "Adaptive Switching Circuits" WESCON Convention Record, Part 4, pp 96-104, 1960.