Project 1 Report

Rushi Desai
rdesai65
GTID: 903295640
Commit Hash: 23bfd764f2e79860932d47244a0bc5e5020cd418

Replicating "Learning to Predict by Methods of Temporal Differences"

Problem Statement: In this paper we try to replicate the results of the Sutton 1998 paper. The original paper introduces incremental methods focused on prediction. The Temporal Difference (TD) methods assign credit by taking difference between temporarily successive Predictions rather than actual and predicted outcomes. The paper relates TD methods to supervised learning and authors make claim that they perform better than supervised learning. Paper relates TD methods to supervised learning and authors make claim that supervised learning methods are inadequate and TD are far preferable.

Supervised learning setting assumes a tuple where given an input, learner must re-produce output as seen during training. For TD learning, the input can be not just 1 observation but can have multiple observations (multi-step). Authors claim that TD methods are better for multi-step prediction problems. Authors also argue that this is fine since most single step problems can also be viewed as multi-step problems. This report will focus on multi-step problems.

We consider multi-step prediction problems in which experience comes in observation-outcome sequences of the form $x_1, x_2, x_3, .., x_m, z$ where each x_t is a vector of observations available at time t in the sequence and z is the outcome of the sequence. Both x and z are real valued. For each observation-outcome sequence, the learner produces the sequence of corresponding predications $P_1, P_2, P_3, \ldots, P_m$ each of which is an estimate of z. The predictions are also based on a vector of modifiable parameters or weights of w. P_t functional dependence on x_t and w will sometimes be denoted explicitly by writing it as $P(x_t, w)$.

The learning procedure will be expressed as rules for updating w. For each observation an increment to w denoted w_t is determined and after a complete sequence is processed, w is changed by sum of all the sequences increments.

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

And Δw be

$$\Delta w = \alpha (z - P_t) \nabla_w P_t$$

To derive an incremental update rule for the learning procedure to update w, the report will consider a special case where P_t is linear function of x_t and w.

$$P_t = w^T x_t = \sum_i w(i) x_t(i)$$

Now, for TD methods, instead of waiting for sequence to end and then update, we can update as we see each observation. To incorporate weight update based on observations we have seen, we will use exponential weighting with recency in which alterations to the predications of observation vectors occurring k steps in past are weighted according to λ^k for $0 \leq \lambda \leq 1$.

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

Above is what we will use as update rule. It should be noted that we we only need current and next step predictions to get weight update.

Computational Experiments:

To show efficiency to TD methods w.r.t supervised learning, bounded random walk problem setting is used. In this problem setting, a underlying Markov Process (MP) generates the sequences of observations with terminating conditions. When the terminating is reached, we treat that as an input sequence and the output label is noted.

Above Markov Process has stochasticity modeled along with the property that the learner is able to observe the state as it evolves over time. This setting is good for us to analyze in detail and we will be able to show that TD methods are more efficient than supervised methods at learning. To generate one sequence of data we just flip a coin to pick to go right or left randomly and keep following this until either of there terminating states i.e. A or G is reached and then that sequence ends. We record the sequence and its outcome i.e. If we terminated at either A or G. This sequence and outcome becomes our input/output pair that we will use for training. We want to estimate probabilities of ending in rightmost state G, given we are in any other state.

Specifically, below **fig1** is the Markov Process described.

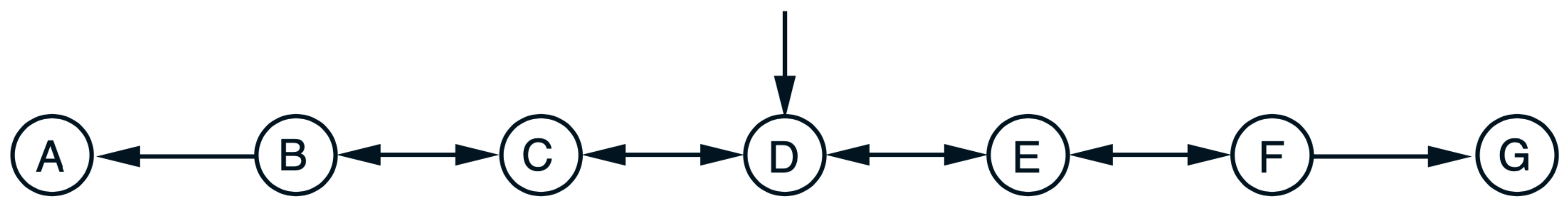

fig1: A generator of bounded random walks. This Markov process generated the data sequences in the example. All walks begin in state D. From states B, C, D, E, and F the walk has a 50-50 chance of moving either to the right or to the left. If either edge state, A or G, is entered, then the walk terminates.

We will apply linear supervised learning and TD method to compare performance. A walks outcome was defined to be $z = 0$ for walk ending in A and $z = 1$ for ending in G. For each nonterminal state i there was a corresponding observation vector x_i if the walk was in state i at time t then $x_t = \mathbf{x}_i$. Thus, if the walk $DCDEFG$ occurred, then the learning procedure would be given the sequence $x_D, x_C, x_D, x_E, x_F, 1$. The vectors $\{x_i\}$ were the unit basis vectors of length 5, that is, four of their components were 0 and the fifth was 1 (e.g $x_D = (0,0,1,0,0)^T$) with the one appearing at a different component for each state. Thus, if the state the walk was in at time t has its 1 at the i^{th} component.

Above experiments can be replicated using git commit hash: **23bfd764f2e79860932d47244a0bc5e5020cd418**

Experiment 1: Repeated Learning Paradigm, Update at end of sequence

Here to simulate supervised learning we don't update weight vector until we see all observations from 1 sequence. We generate 100 training sets each containing 10 sequences. Each sequence can be of arbitrary length since it depends on when we would have reached end/absorbing/terminating state. True probabilities of right side termination are $1/6$, $1/3$, $1/2$, $2/3$ and $5/6$. These are not needed by learning algorithms (since that is what we are estimating) but we will only use them to check how well our learners are doing. As a measure or performance of learning procedure, Root mean squared error (RMSE) between the procedure's asymptotic predictions using that training set and the ideal predictions.

The result is shown below.

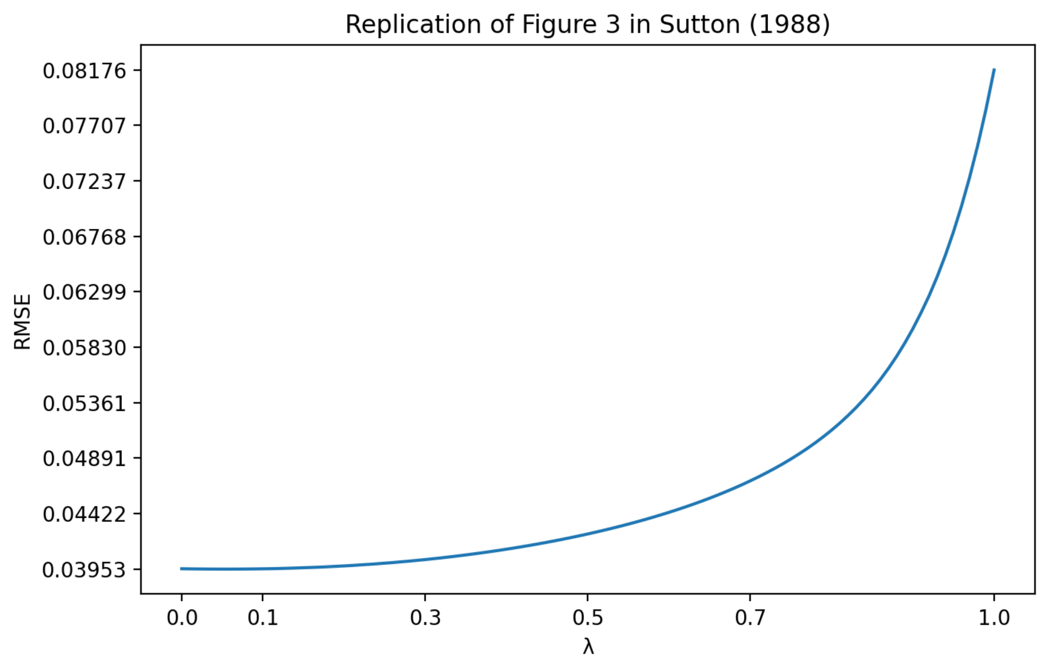

fig2: Replication of fig3 in Sutton (1998). RMSE with different λ values when $\alpha = 0.001$, Dataset1(seed=30)

This was the most ambiguous experiment for me. In the Sutton paper, they don't disclose what was the alpha that was used. For above fig, I used binary search to find out alpha for which we converge and for which convergence is faster and we get similar figure the one in the paper. Specifically, I used alpha of 0.001. The shape of the curve is similar the one seen in the paper however, the ranges of RMSE are different and I am getting higher ranges as compared to fig3 in paper. My hypothesis is that algorithm is sensitive to the data it sees during training. If it sees enough examples for uniform distribution of sequences possible, it will be able to reduce the gap between RMSE achieved via training and actual probability values. This was observed when I changed the seed used to generate different kinds of data set and on each dataset the RMSE achieved was different. Example of this is when data generated was changed (by using seed=33) we get below figure.

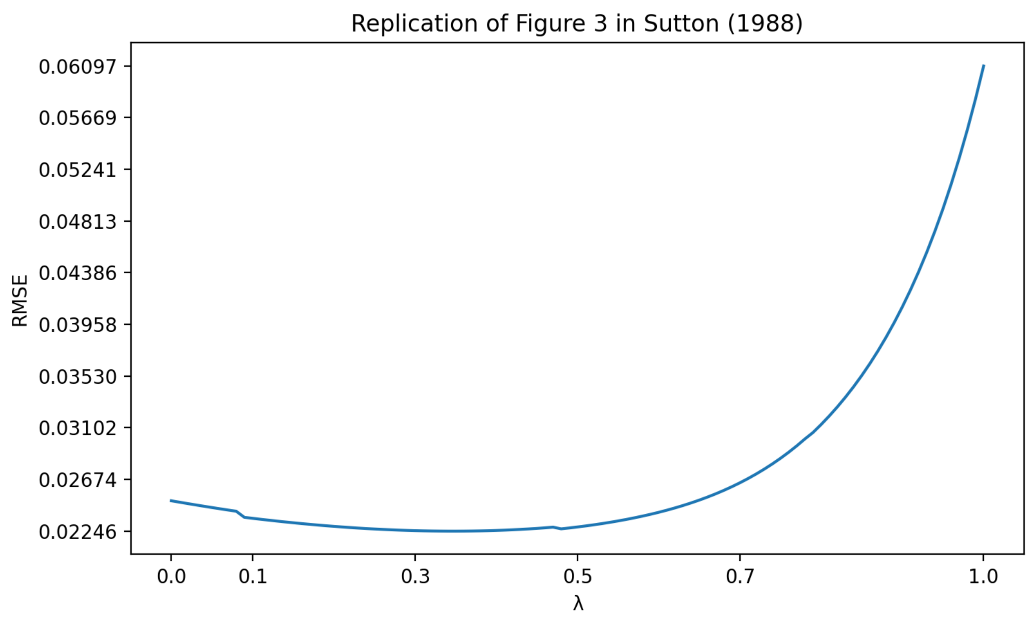

fig3: Replication of fig3 in Sutton (1998). RMSE with different λ values when $\alpha = 0.001$, Dataset2(seed=33)

Note that the ranges for RMSE in **fig3** is different than **fig**

Experiment 2: TD learning - Learning rate when training presented once

The second experiment concerns the question of learning rate when the training set is presented just once rather than repeatedly until convergence. Same data set is used for various λ.

The full experiment setup is following, First, Each training set was presented once to each procedure. Second, Weight updates were performed after each sequence (rather than waiting until end as in Experiment 1). Third, Each learning procedure was applied with a range of values for the learning-rate parameter α. To avoid any bias for either toward right-side or left-side terminations all components of the weight vector were initially set to 0.5

The result is shown in **fig4**. Here we used λ values 0, 0.3, 0.8 and 1. As pointed out in Sutton Paper, α values had a role in determining best performance. This also showed that optimal performance was achieved at $\alpha = 0.3$ and $\lambda = 0.3$ roughly. This part was relatively straight forward once Figure 3 learner was implemented. The only changes needed were to introduce iterating over

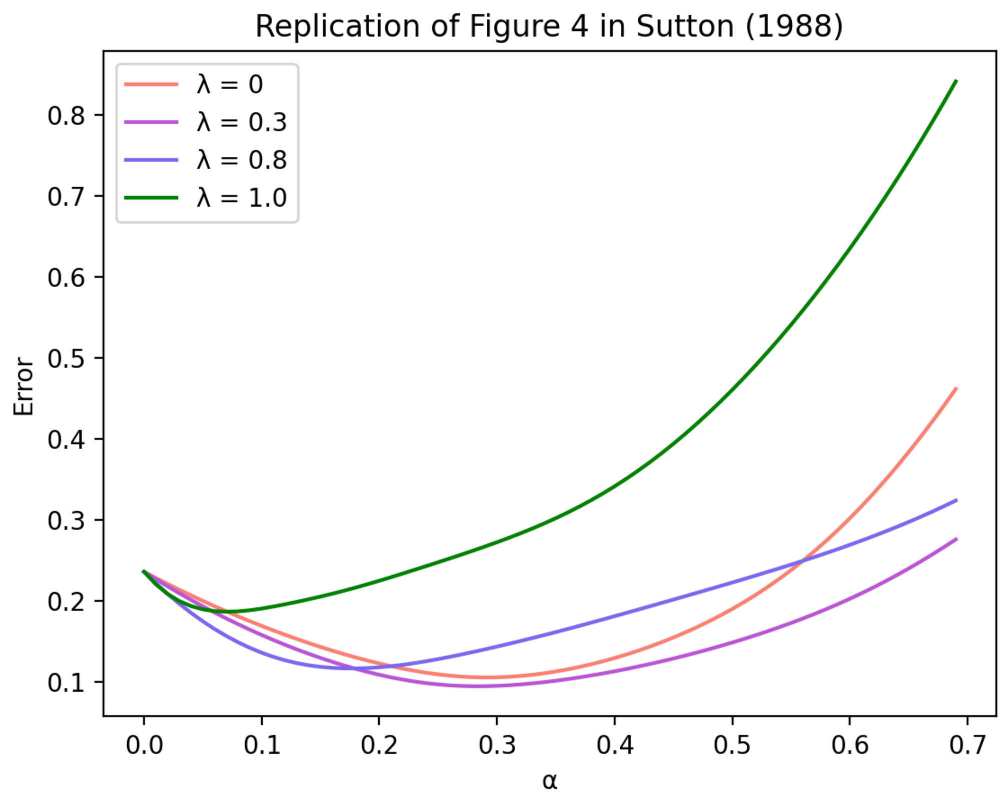

fig4: Replication of fig4 in Sutton (1998), training set shown only once. RMSE for various λ and α

different values of alpha, updating weights after every observation vs waiting till end of sequence and finally removing the convergence checking code since we only show dataset once.

In this experiment the learning method was much more robust and even when different training data was generated (using different seeds), they all got same convergence in terms of RMSE w.r.t λ and α. My hypothesis is that this can be understood by thinking about overfitting. In experiment 1 we presented same data over and over again so the learner might be over-fitting to data. In this case, we only present data once and hence we avoid overfitting

Experiment 3: TD learning - Best RMSE error for each lambda

The idea behind this experiment is to find out which exponential weighting i.e. which λ would give the best i.e. lowest RMSE error assuming we are allowed to choose any learning rate α. Implementation was again relatively straight forward once Experiment 2 was already implemented. In order to achieve this, the only additional step was to take a min for each lambda on the RMSE error across various alpha.

The result is show below:

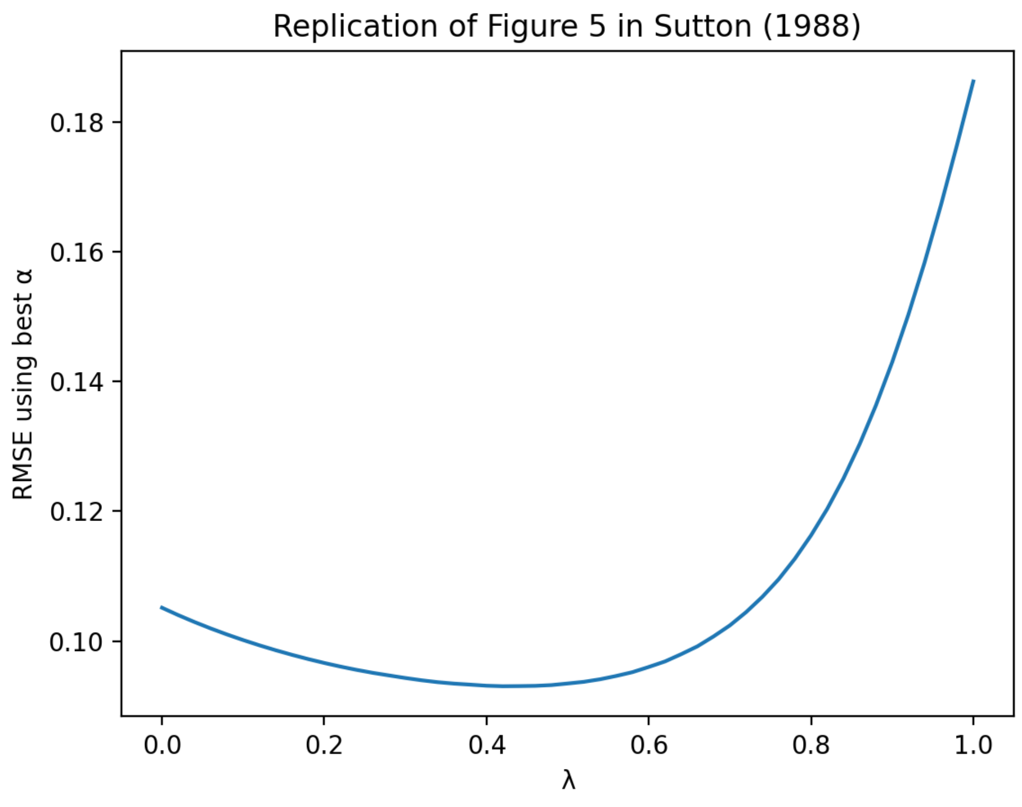

fig5: Replication of fig5 in Sutton (1998)

Conclusion/Analysis:

Implementation has 3 parts. First is data generation, this is implemented in *random_walks* method. For *random_walks* equal probability for going left or right was assumed. Second, is given training sequences, lambda, alpha and z values, output deltas to be added to weights. This is implemented in *learn_weight_updates*. Both these are re-used across 3 experiments. The third part is what is varied across experiments.

Experiment 1 was most ambiguous for me. Mainly the authors didn't talk about alpha values. This was resolved by trying out different alphas systematically (binary search) for which convergence was seen. Convergence was checked similar to HW1/HW2 where we keep track of deltas during run and if the change in delta is below threshold, then we stop. Experiment 1 results didn't quite match the Sutton paper results specially the range of RMSE that I was able to achieve. As discussed in that section before, my conclusion is that this is due to the Repeated Learning paradigm led to overfit the data i.e. it made the learner too sensitive to data. Thus when training data was changed, we saw drastic changes to RMSE curves. Also, showing entire sequences before updating weights also seem to have played a role in overfitting.

Experiment 2 and 3 were more incremental once implementation for 1 was written. For both experiment 2 and 3, results were much similar to Sutton Paper. My understanding is that this is due to that fact that showing data only once and not waiting for entire sequences before updating weights made the learning algorithm more robust by not letting it overfit data. And hence didn't change when training data was changed.

Main issues/pitfalls were lack of providing certain parameters, confusing write where experiments setup and results were little jumbled up while describing etc. For parameters, systematically searching for correct values helped me get over them. For write up reading multiple times and piazza discussions helped.