Desperately Seeking Sutton: Project 1

Kelly "Scott" Sims

Abstract*—this article is an examination and reconstruction of Richard Sutton's 1988 paper "Learning to Predict by the Methods of Temporal Differences". In it, Sutton reviews two prediction algorithms; conventional supervised learning and temporal difference. He goes on to propose TD methods as being superior with respect to dynamic environments and demonstrates such with a simple Markov experiment. The contextual paradigm of superiority between the two algorithms is bounded by two key attributes, computational efficiency and accuracy. Here, we aim to achieve statistical repeatability of these experiments and provide further analysis of results.

I. Introduction

This article centers around Temporal Difference methods and their efficacies for prediction problems. It is a replication of experiments performed in Sutton's paper *Learning to Predict by the Methods of Temporal Differences (1998).* In it he argues that *TD methods* are not only more accurate than conventional supervised learning methods, but they're more computationally efficient while leading to faster convergence. Historically, conventional *supervised learning* methods have been used due to the ease of interpretable results and understanding. Sutton points out, however, that this is merely the case because TD methods "were never studied independently". Ultimately, he formally proves convergence of TD methods and demonstrates, experimentally, their superiority with respect to prediction problems involving dynamic systems.

The conventional *supervised learning* paradigm is one in which a mapping function is learned through iterative updates of a weight vector. These updates are a function of the error between prediction and true observation. More formally, *supervised learning* aims to optimize an objective function by minimizing prediction error. Upon convergence, optimization should produce a model that can make predictions with some thershold of accuracy. There are many optimization functions used, and for different reasons. However that is outside the scope of this article. Instead, the main thing to note is that optimization is solely dependent on input to output mappings of i.i.d observations. No sense of sequence or "*temporal difference*" is accounted for during training. A common example of this is the prediction of housing prices when given inputs (features) of square footage, number of rooms, number of bathrooms, zip code, etc. Given these inputs to an optimized model, it should produce a close to accurate price.

A problematic consequence of this global approach is the lack of sequential information. In practice, *supervised learning* trains predictive models from sampled data. If the given data sampled, for example, were over the course of 10 years for a specific area, this sampling would fail to account for rate of changes of inflation, equity, and/or market reactions. According to the law of large numbers, sampling multiple times from this large time span dataset would yield a more central price. This might be sufficient for general pricing. But for the sake of accuracy with resepect to current pricing trends, this method would fall short.

As an anecdotal example, we want to predict how much a house will sell for if it's expected to to sit on the market for at least two months. The house is in a particular zip code where resale values have been increasing at a constant rate year over year. However, a recession has recently hit the housing market and prices are in decline. *Supervised learning* would fail to account for this more recent sequential drop in price prediction. It would still provide a central number based off of all historical data available. This paradigm is one in which Sutton argues that *TD methods* would be more accurate. Using *TD methods*, continual price drops month after month would update pricing expectations with respect to this downard trend immediately.

Expanding on this novel example, week one might predict the price to be $450k. But after another week of recession struggles, comparable homes in the neighbhorhood have been selling at 10% below asking. With TD, we can update our prediction to account for this drop and adjust the $450k price point immediately. Conversely, with *supervised learning*, a substantial amount of price drop data would have to be accumulated over time and the model retrained, in order to realize a significant effect of the recession.

For completeness, *TD methods* exploits the Markov property whereas *supervised learning* does not. *TD methods* can be in the form of bootstrapping (creating estimates from estimates), sampling (what *supervised learning* does), or a combination of the two. When trying to predict in a dynamic system Sutton's experiments attempt to show, in the simplest case, that *TD methods* in fact perform better than conventional learning because of this exploitation. With that, we recreate his analysis to asses repeatability and confirmation of his results.

II. Supervised Learning vs TD(λ)

Complete mathematical proof and rigor behind the concepts surrounding this article is beyond scope. We only introduce key concepts and equations here for continuity and the minimization of ambiguity. *Supervised learning* as described by Sutton, and how it is generally regarded, is learning with respect to input/target pairs (X_i, y_i) where X_i is input features and y_i is the target value. A mapping function consisting of weights is learned during training such that a linear combination of these weights and X_i results in a prediction P_t. Sutton (1988) notates this linear combination as

$$P_t = w^T x_t = \sum_i w(i)\, x_t(i) \tag{1}$$

The error between P_t and y_i $[(y_i - P_t)]$ is the subject of optimization. Supervised learning aims to reduce the error between prediction and ground truth for each observation in X_i. Being that the observations, X_i and target values, y_i are constant throughout the learning process, error in predictions is attributed to values of the weights themselves. *Supervised learning* converges to a solution with minimum mean-squared error. That is, it best fits only the observed data. Again, with no rigor provided, the weight update procedure is

$$w \leftarrow w + \sum_{t=1}^{m} \Delta w_t \tag{2}$$

Where w is each weight in the vector w_i and Δw_t is the gradient of the weight vector with respect to component *t*.

$$\Delta w_t = \alpha (y_i - P_t) \nabla_w P_t \tag{3}$$

We only make note here of the update procedure on the weights during training without formal proof or derivation. The importance of highlighting this update in-place is so that the connection between *supervised learning* and *TD(λ)* can be realized. Alpha in equation 3 is the learning rate. This is a tune-able hyperparameter that dictates how much we want to move our weights in the direction of the gradient.

TD(λ) on the other hand converges to a solution of maximum likelihood Markov model (*silver 2015*). It is an averaging of values over various n-step returns. Here, n-step returns simply means evaluating returns over a discrete number of steps. If n = 1, we transition one step in the future and evaluate our values. If n = 5, we transition 5 steps in the future and evaluate. *TD(λ)* allows us to evaluate multiple n-step returns simultaneously and average them.

TD methods [TD(λ)] have a similar update structure for its weights (4). The main thing to note however is that there is not a linear function mapping inputs to outputs. The value of a particular state is defined as the expected reward from being in that state and transitioning either in a finite or infinite horizon. Therefore, we need not find weights for an extrapolated mapping function. We simply initialize state values, and update those values with actual returns experienced from the dynamics of the environment continually until convergence.

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

With no formal proof, note that when lambda is equal to 1, equation 4 is equivalent to equation 3. Therefore, TD(1) produces the same weight updates as *supervised learning*. This is an important distinction to keep in mind when evaluating the experiment. Because TD(1) and *supervised learning* are equivalent, the results of TD(1) compared to all other values of lambda are reflective of the performance of *supervised learning* compared to all other values of lambda.

III. The Experiment

Sutton's experiment to highlight the efficacies of TD methods was a simple one in principle. He proposed a bounded random walk. Given seven different states [A, B, C, D, E, F, G], a walk always starts at state D. The only actions allowed during this walk is a one step transition either to the left or right. The action space is one of equal probability in that, there's a 50% chance of going left or 50% chance of going right. The state you are in bares no influence on the action taken. A walk is deemed terminal whenever either state A or G has been reached. Transitions between states [B, C, D, E, F] yield zero reward. In fact no reward, either positive nor negative, is achieved unless state G has been reached. Reaching terminal state A nets 0 reward while state G nets a reward of 1. Figure 1 shows the exact depiction of the experiment as presented in Sutton's paper.

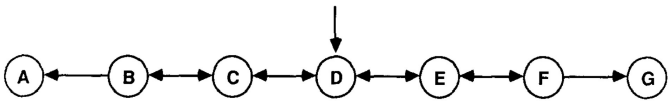

Figure 1: Original depiction of the Markov process as designed by Sutton from his paper "Learning to Predict by the Methods of Temporal Differences"(1988)

Given the basis of this random walk experiment, we seek to find the value of each state. Value used here, is a relative term with respect to the two aforementioned prediction algorithms. Although, both allude to the same thing, the context in which it is used differs. In *supervised learning*, the value of state is simply the probability of ending up in state G for any given state (or 1-probability for state A). An input state X_B, X_C, X_D, X_E, X_F is mapped directly to a prediction P_t. This prediction is the measure of "value". Meanwhile, *TD methods* correlate value with "how good is it to be in a particular state". It is an estimate of expected future reward given the current state. Continuing on, the use of the term weights or values is one in the same and will be used interchangeably.

From this scenario, two experiments were conducted. The first experiment centering around the proof of *TD methods* performing better than *supervised learning* in a dynamic

environment. The second experiment seeks to highlight the underperformance of *supervised learning* over several permutations of learning rate and prediction methods. In either case, data ingestion during training remained the same.

Data is randomly generated in the form of 100 training sets. Each set contains 10 sequences of walks. There were no restrictions in place dictating how long a walk could be. Therefore, each training set contained walks of random variable length. For both experiments, all 1000 sequences were used during training. The difference in implementation between the experiments came with respect to weight updates and convergence objectives. Experiment one seeks full convergence while weight vectors are updated only after seeing all training data. Experiment two presents all data only once while updating weight vectors after seeing one complete sequence. Each experiment will be outlined with greater detail in subsequent sections

IV. Experiment Setup and Implementation

The environment utilized for the experiments was Python version 3.7.4 running on Linux Ubuntu 18.04 LTS kernel. The first initiative was the implementation of a Random Walk Data Generator. Using the numpy library version 1.17.2, a class was built that would generate data sets at a user defined amount. Each dataset would contain variable number of sequences, again as defined by a user. Given that Sutton specifically outlined using 100 training sets containing 10 sequences, this ultimately was used. The structure of a single training set is as follows. An entire sequence of a random walk is contained in a Python namedtuple "Walk" which contains two named attributes; Route and Reward.

Route, is the randomly generated walk, vectorized as an M x 7 numpy array. M being the number of transitions taken during the walk. Each row is a unit vector of 6 zeros and a single 1, representative of the current state. Starting from state D, a number is randomly sampled from a uniform distribution between zero and one [0, 1] using numpy. If the number sampled is less than 0.5, the action of a left transition is applied and another unit vector is created representative of state C. The opposite is true for a number randomly drawn greater than or equal to 0.5. A unit vector would be created representing state E. This process continues until either state A or G is reached. An example walk of X_D, X_C, X_D, X_E, X_F, X_G is illustrated in Figure 2.

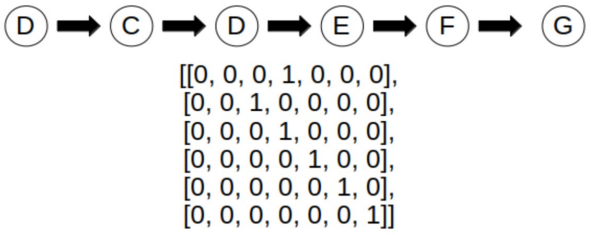

Figure 2: Example of random walk that transitions across multiple states and how it is represented in vector format. Index for each row in the array is noted as states A, B, C, D, E, F, G

The Reward attribute of the namedtuple is a single vector of length (M – 1). It maintains the rewards received for each transition as randomly generated. This, trivially, is a vector of all zeros except in the instance of a walk ending in state G. The subsequent vector for the walk in Figure 2 would be [0, 0, 0, 0, 1]. This was done for ease of indexing during implementation.

Semi-gradient TD(λ) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0,1]$
Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  Initialize S
  $\mathbf{z} \leftarrow \mathbf{0}$ (a d-dimensional vector)
  Loop for each step of episode:
    Choose $A \sim \pi(\cdot|S)$
    Take action A, observe R, S'
    $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \nabla\hat{v}(S,\mathbf{w})$
    $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
    $S \leftarrow S'$
  until S' is terminal

Figure 3: Pseudo code from Barton & Sutton Reinforcement Learning [pg. 293]

The algorithms themselves were reproduced using Python 3.7.4 and Numpy as well. Instead of using the exact gradient updates as notated in Sutton's paper, the "eligibility trace" algorithm was utilized. The pseudocode which can be seen in Figure 3, is taken from the book *Reinforcement Learning (Barto & Sutton 2nd Edition)*. The only change in actual implementation was the placement of the weight updates. This will be dicussed further in later sections.

A. Experiment I

The first experiment was centered on performance of TD methods versus *supervised learning*. Sutton noted during training, weights were accumulated until termination of the 10th walk in a set. The weights were then updated as given by (4). The sets were repeatedly iterated over until convergence. This simply means that all states were initialized to values of 0. For a given sequence, the value updates were stored for each transition. State values were then reset to 0 for the next sequence and the process repeated. After the 10th sequnce, the weight updates are performed and the updated values (weights) were the initial starting values for the next loop of the sequences. Upon convergence, the root mean squared error was calculated between the final state values and the ideal values as specified by sutton [1/6, 1/3, 1/2, 2/3, 5/6]

The aformentioned steps were performed for all 100 training sets so that the result was 100 different root mean squared errors. These RMSEs were then averaged to give the final error. These steps were repeated for varying levels of lambda between 0 and 1, inclusive. Sutton failed to specify two key aspects in his implementation. Equation (4) is dependent on lambda and learning rate, alpha. The paper only makes note of a "best alpha" value in the results, but doesn't clarify what it was. Therefore, multiple permutations of lambda and learning rate alpha were utilized, in search of the reproduction of Sutton's results. The alpha that best matched his results was found to be 0.2.

The second ambiguity was the convergence criteria. He only states in the implementation that each training set is repeated until convergence, but does not clarify what this was.

The method used in our results was to maintain the previous iteration of all 10 sequences' weight values as a separate variable. After performing the weight update step, the difference between the previous weights and the new weights was calculated. If this difference was less than or equal to 0.001, then convergence was declared. The results from this experiment can be seen in Fig 4. The inset graph is the results from Sutton's original experiment.

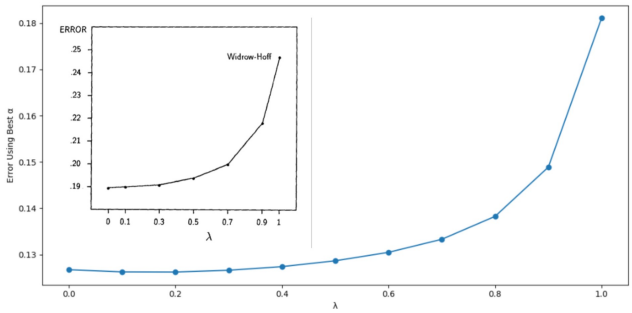

Figure 4: Reproduction of Sutton's first experiment. The inset graph is Sutton's original results

It is clear from Figure 4 that TD(1), which is representative of *supervised learning*, doesn't perform as well as all other methods of *TD(λ)*. A better way to think of these results is to consider TD(1) not as *supervised learning*, but as Monte Carlo updates (*silver 2015*). When lambda is 1, no n-step averaging is taking place with respect to each update. The full return of all steps is accounted for with respect to the weight update procedure. This is no different than what is done with Monte Carlo methods.

For an episodic task, monte carlo accumulates the entire return value for each episode and then averages the results. One downside to this method is that an entire episode must terminate before anything can be learned, so learning is slower. Another problematic attribute of MC is that it is high variance. That is, it overfits whatever training data is presented to it from a sample. It doesn't generalize very well to the true representation of the environment if those samples are skewed. This distinction was noted earlier in the discussion of *supervised learning*. If the random generator produces a training set that is very imbalanced to one terminal state over the other, then MC will tend more to the imbalanced side. It takes in no notion of connectivity or temporal information. For state D, if the random generator created a training set of all sequences teminating in state A, except for one sequence, then the value of D would be far lower than the ideal value due to being averaged over more 0 rewards from state A.

TD methods, however, can learn during an episode. It is an averaging of all n-step returns to better represent a Maximum Likelihood estimate. These methods implicitly build an underlying MDP of the data that accounts for temporal information. The advantage of this method is clearly seen from the results; TD methods outperform *supervised learning*.

Note that the overall errors in our reproduction of Experiment 1 are less than those presented by Sutton. There is no proven explanation as to the difference in magnitudes. Multiple training sets were randomly generated in order to find some combination of training sets that would produce such high errrors. We were unable to do so. We can only offer speculation, but a potential misinterpreation of the steps to reproduce Sutton's experiment could have led us to lower errors. The opposite is likely as well, that Sutton performed some computational error during his experiment that went uncaught. The third thought on the matter comes down to the exact training set used. Sutton could have randomly generated a, typically, unlikely set of sequences that would cause such an increase in error from the norm.

A. Experiment 2

The next experiment was an attempt to highlight the inferior performance of *supervised learning* across a larger domain of the hyperparameter alpha. Experiment 2 is an exercise in gridsearch training. This means that multiple lambda values were tested across multiple alpha values. Only this time the values were intialized to 0.5, and each lambda method was not allowed to converge. Instead, each sequence for a given training set was seen only once. During the transitions of a single sequence, the weights were accumulated. Upon reaching a terminal state, the weights were updated with the accumulated weights, and these new weights served as the initial state values for the next sequence in the training set. After all 10 sequences had been seen, the RMSE was calculated using the same ideal values from Experiment 1. Again the end result is 100 RMSE values for the 100 training sets, that were then averaged to give the final error for a given lambda with a specific learning rate alpha. These steps were repeated for multiple values of alpha for each lambda value.

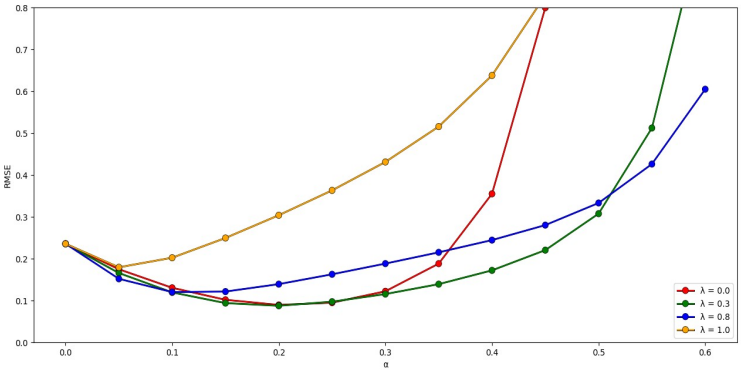

Figure 5: Reproduction of Sutton's second experiment.

The results from experiment 2 can be seen in Figure 5. Note that the error for $\lambda = 0$ increases rapidly around $\alpha = 0.4$ and crosses over $\lambda = 1$ for our experiment. This is believed to be due to the length of the training sets generated. Sutton never clarified how long each sequence was in his original training set, nor did he specifiy if there was a maximum length restriction applied. However, it was discovered by others that

limiting the maximum length of the sequences to a value around 20 transitions would produce the same results as Sutton's original work seen in Figure 6. This intuitively makes sense however. Given there are only a total of 7 states, sequences with a large number of transtions obviously would visit the same states alot. This would cause an n-step average over a state, vast amount of times and dilute the derived value. We chose not to restrict sequence sizes in order to maintain pure randomness. We believe that by imposing a rule on the genration of states (maximum length), there comes a point in generation that the walks are no longer random and become deterministic.

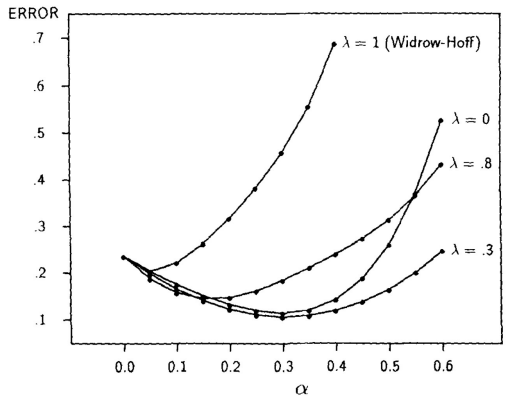

Figure 6: Sutton's original results from Experiment 2

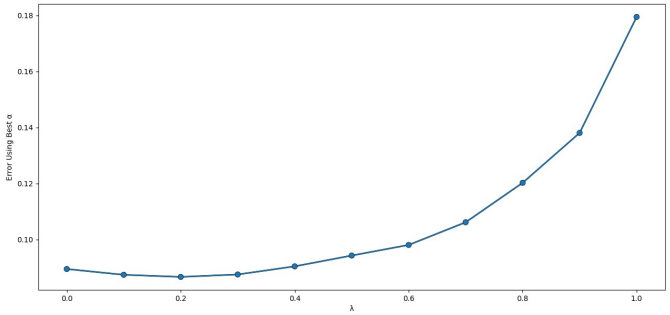

Figure 7: Results from Experiment 2 that shows the alpha values for multiple values of lambda that produced the lowest error for that lambda value.

The general interpretation of the results is as one would expect in any learning situation. Given some learning rate, there is more or less, a sweet spot in which error is the lowest. As learning rate gets higher, the gradients tend to overshoot either a local or the global optimum and error increases. This is perfectly highlighted in Figure 5 and 6. All methods of lambda start to converge until a particular alpha is reached that causes optimization to diverge. In all cases however, each *TD method* less than 1 performed better than TD(1).

Finally, the best alpha value for several lambda values was extracted from experiment 2 and plotted. The results can be seen in Figure 7. Just as it was seen earlier from experiment 1, supervised learning fails to outperform all lambda values less than 1. The main thing to note here is that lambda of 0 is not the optimal method like it was in Figure 2 from the previous experiment. This is an effect of not allowing convergence of each training set. TD(0) is only a one step look ahead, so it takes multiple updates of the weights before the change at state F, for example, is realized at say, state B. Since each sequence was looped through once, not much information was backpropagated through the states.

One major event of misinterpretation of Sutton's experiment 2, on our part, had to do with the weight update step. Sutton stated for Experiment 1 that no weights were updated until all 10 sequnences had been seen. With experiment 2, the weights were updated between sequences. Although the wording was clear that it was a backward view (*silver 2015*) implementation, we initially performed an *exact online* implementation. We were updating the weights inplace between individual transitions in a sequence, not between the sequences themselves. It was only after conferring with fellow students on the appropriate experimental steps that the error was discovered. Upon correcting the updating step, we were able to produce the same general results as Sutton. Surprisingly enough, this one change in placement of the updates greatly changed the results of the experiment. The overall proof these experiments were trying to convey still held true however, that *supervised learning* methods are inferior in multi-step dynamic environments to TD methods.

REFERENCES

[1] *R. Sutton "Learning to predict by the methods of temporal differences"; Sutton, R.S. Mach Learn (1988) 3:9. https://doi.org/10.1007/BF00115009*

[2] *Sutton, R. & Barto, Andrew. Reinforcement Learning 2nd Edition. MIT Press, Cambridge, MA, 2018*

[3] *Silver, D. 2015 Lecture 4: Model-Free Prediction, University College London, delivered May 13, 2015*