Temporal Difference Learning on Random Walks

Ansel Lim
ansel@gatech.edu
https://github.com/ansellim/TDlearning.git

***Abstract*—This paper summarizes the fundamental mathematical ideas of temporal difference learning as presented in Richard S. Sutton's 1988 paper in *Machine Learning*. Temporal difference learning is implemented and demonstrated on a simple dynamical system of bounded random walks which was presented in Sutton's paper. The key findings of Sutton's paper are reproduced and explained.**

***Index Terms*—temporal difference learning, random walks, reinforcement learning**

I. Introduction to TD learning

A. TD learning versus supervised learning

In this paper we replicate the key numerical experiments presented by Richard S. Sutton in his seminal paper titled "Learning to Predict by the Methods of Temporal Differences", which was published in the journal *Machine Learning* in 1988. The main thrust of the paper is how temporal difference learning differs from supervised learning, and how temporal difference learning is superior to supervised learning on multi-step prediction problems. Multi-step prediction problems are problems where, instead of acquiring full information about the correctness of a prediction immediately after making that prediction, the learning agent only acquires partial information about the correctness of the prediction; the conclusive correctness of the prediction only becomes evident at the end of all steps in the problem.

The fundamental idea of temporal difference (TD) learning is that the difference (error) between temporally successive predictions drives learning. This incremental nature of TD learning contrasts it from conventional supervised learning which focuses on the difference between predicted and actual outcomes. Sutton (1988) has suggested that on multi-step problems TD methods are advantageous because they are computationally cheaper, converge faster, and produce more accurate predictions.

B. Notation

TD learning is suited to multi-step prediction problems. We will adopt the following notation for multi-step prediction problems in this paper.

A multi-step prediction problem consists of a sequence of multiple observations spread out in time (represented by a sequence of vectors $x_1, x_2, \cdots, x_m$), as well as a final outcome z, which is a real-valued scalar. Each observation x_t (for $t = 1, \cdots, m$) determines a prediction P_t, so that we have a temporal sequence of predictions $P_1, P_2, \cdots, P_m$, all of which attempt to estimate the final outcome z (for notational purposes, $P_{m+1} \triangleq z$. For simplicity, assume that the prediction model is a linear model with shared weights, so that we can write $P_t = w^{\mathrm{T}} x_t$ for all t. Notice that the prediction model is linear in both the observation vectors and the weights vector. Furthermore, the prediction at time t depends only on the weights vector w and an observation x_t at a single time point.

Following this paradigm, all learning procedures on multi-step problems seek to optimize the weights vector w. Suppose that the update to the weights vector w is achieved incrementally. For each observation x_t, an increment to w, denoted Δw_t, is calculated. If we accumulate the increments over a sequence of observations, and then update the weights vector at the end of the sequence, then we can write the update rule for the weights vector as follows:

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

C. TD(1) learning

As mentioned previously, TD lea
February 21, 2022rning procedures focus on the difference between temporally successive predictions. The question then is, "which successive predictions do we compare?" As an introduction to TD learning methods in general, we shall consider the simplest method in this family, TD(1) learning.

The most intuitive method, which is called TD(1) learning, simply evaluates the error of a prediction P_t by comparing it to the prediction made at the next immediate time step P_{t+1}. In fact, this procedure is equivalent to the Widrow-Hoff rule, a classical supervised-learning procedure. In fact, TD(1) learning may be seen to be an incremental implementation of the Widrow-Hoff rule. Let us now derive the update rule for TD(1) learning from the Widrow-Hoff rule.

Let us recall that the Widrow-Hoff rule is characterized by the following increment

$$\Delta w_t \triangleq \alpha(z - P_t)\nabla_w P_t = \alpha(z - w^{\mathrm{T}} x_t)x_t \tag{2}$$

where α is the learning rate and the quantity $(z - P_t)$ represents the error associated with the prediction at time t.

Since $P_{m+1} \triangleq z$, we can rewrite the error quantity $(z - P_t)$ as a sum of differences of successive predictions

$$z - P_t = \sum_{k=t}^{m} (P_{k+1} - P_k) \tag{3}$$

which is then combined with (1) and (2) to obtain the update rule (after simple algebraic manipulation of the sums)

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

which implies that the increment Δw_t may be written as

$$\Delta w_t = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \nabla_w P_k. \tag{5}$$

Realize that the increment Δw_t (at time t) depends only on the current prediction P_t and the immediate next prediction P_{t+1}, and on the sum of previous gradients from the start of the sequence until the current time step.

D. TD(λ) learning

The TD(λ) family of learning procedures is a straightforward extension of the TD(1) paradigm just presented. In the TD(1) procedure, the differences between a pair of successive predictions P_{t+1} and P_t are weighted equally in the increment Δw_t (and, therefore, also the sum of increments $\sum_t \Delta w_t$), regardless of whether this pair of successive predictions occurred earlier or later in the sequence of observations. In that sense, the alterations to the predictions P_t of observation vectors are time-invariant.

We may modify this paradigm by *discounting* remote events relative to more recent events. Intuitively, the increment Δw_t computed at time step t should make a larger alteration to predictions P_k which are more recent, compared to predictions which are further back in the past. This idea may be concretized with an exponential weighting with recency. We introduce an exponential weighting to (5) to get a new increment formula for the weight increment at time t:

$$\Delta w_t = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k. \tag{6}$$

The alterations to predictions occurring x steps to the past are weighted according to λ^x, where the discount factor λ is a nonnegative number smaller than or equal to 1. Because $0 \leq \lambda \leq 1$, predictions occurring further back (smaller k, larger $t-k$) receive smaller alterations (smaller λ^{t-k}) compared to more recent predictions (larger k, smaller $t-k$, larger λ^{t-k}).

Notice that TD(0) corresponds to the special case where the sum of gradients $\nabla_w P_t$ reduces to only the gradient of the current time step. That is,

$$\Delta w_t = \alpha(P_{t+1} - P_t) \nabla_w P_t \tag{7}$$

II. General methodology of TD learning in random walks

The rest of the paper demonstrates the application of the family of TD(λ) procedures to a simple example of bounded random walks. A bounded random walks example is a dynamical system with an evolving state which is observed and partially revealed over time. Although the random walks example is a simple dynamical system, it serves to illustrate how TD methods efficiently use their experience and previous predictions to converge quickly to accurate solutions. TD methods may be applied to other, more complex dynamical systems, by following similar principles and approaches presented here.

A. Dynamical system of bounded random walks

Consider a simple dynamical system which generates bounded random walks. The reader may refer to Fig. 2 in [1] for an illustration of the system. Here, we will provide a brief written description of the system. The system has only 7 states which are labeled alphabetically from A to G. The 7 states are ordered from left to right in alphabetical order. The leftmost and rightmost states, i.e. states A and G, are terminal states with outcome values (z) of 0 and 1 respectively. The walk always begins in the state in the center, namely state D. At every state, the agent randomly moves to a neighboring (adjacent) state to the left or to the right, and it does so with equal probability. No rewards are obtained from these movements. The exceptions to this random leftward and rightward movement are at the terminal states (the leftmost state A and the rightmost state G)—where no further movement is possible—and the state immediately adjacent to each of these terminal states. These "border" states are state B (adjacent to state A) and state F (adjacent to state G). At each "border" state immediately adjacent to either terminal state, the agent has probability 1.0 of moving to the corresponding terminal state.

What are we trying to learn here? We can model this as a multi-step prediction problem. We are interested in determining the probability of achieving the ideal terminal state G when we are at any of the other non-terminal states. Here, we should recall that each time point, the observation is the state we are currently in. At each time point, we can form a prediction P_t which is parameterized by a weight vector w and the state x_t which we are currently in. This prediction value P_t attempts to predict the value of the final outcome (z) of the sequence. The quality of the predictions is determined by how close the predictions are to the true probabilities of terminating in G. The true probabilities of ending up in terminal state G, for each of the states B, C, D, E, and F, are 1/6, 2/6, 3/6, 4/6, and 5/6 respectively. Ideally, we would want our predictions P_t to be as close to these values as possible.

B. Setup of numerical experiments

The dynamical system described in the previous subsection was implemented using just standard Python and NumPy.

From an implementation perspective, the states B to F were considered to be the unit basis vectors of length 5 (with the 1 appearing at a different position for each state), whereas the terminal states A and G were modeled by the integers 0 and 1 respectively. For example, state B is the tuple $x_B = (1, 0, 0, 0, 0)^{\mathrm{T}}$. A sequence of events, or a random walk, is an ordered collection of state tuples. As an example, if the walk DCDEFG occurred, then this would be represented by a sequence of vectors/tuples $[x_D, x_C, x_E, x_F, 1]$. Stated another way, a walk is a sequence of states x_t at each time step t which ends with one of the two terminal states.

Based on the rules of the dynamical system described in the previous subsection, a collection of 100 training sets, each consisting of 10 sequences of states, was generated for our numerical experiments.

The prediction model constructed was linear in the observation vectors (states). The predictions $P_t = w^{\mathrm{T}} x_t$ at each time step t were parameterized by a weight vector w of length 5.

Depending on the experiment, the weight vector was randomly initialized using a random number generator (Experiment 1), or initialized to the same value, in particular 0.5, in all five of its components to avoid bias toward left or right-sided terminations (Experiment 2). In each training set, the objective of the learning algorithm was to update this value of w according to an increment quantity that was calculated at each new time step, corresponding to a state in a sequence. After the appropriate update procedures (discussed later), the final (asymptotic) weights were used to compute the final predictions P_m. The root mean squared error (RMSE) between the predictions P_m and the ideal predictions $(1/6, 2/6, 3/6, 4/6, 5/6)^{\mathrm{T}})$ was then computed. The aforementioned learning methodology was repeated for each of the 100 training sets to obtain an average RMSE.

The process of arriving at the final weights from the initialized weights relies crucially on incremental weight updates. Weight updates utilize the formulae shown in (1) and (6). Specifically, each individual increment Δw_t was computed at each observation (state) according to (6), which is straightforward to implement. The application of (1), in particular *when* the weight update (using sums of increments) should be performed, is less straightforward from an implementation perspective. From an implementation perspective, the sums of increments $\sum_t \Delta w_t$ may be applied to w in three possible settings:

1) w is updated only at the end of a collection (training set) of multiple sequences, $\{x_1^{(s)}, x_2^{(s)}, \cdots, x_m^{(s)}\}_{s=1}^{s=S}$. Note that different sequences in the collection may have different lengths (number of observations) since they may terminate after different numbers of steps. The training set may be seen once or multiple times.
2) w is updated at the end of each sequence of observations $x_1, \cdots, x_m$, after accumulating Δw_t over that sequence
3) w is updated after each observation x_t in a sequence $x_1, \cdots, x_m$ in a training set

All three settings detailed above perform the weight update $w \leftarrow w + \sum_t \Delta w_t$, however they differ in *when* this weight update is performed. The timing of the weight update affects the programming logic, which controls how the Δw_t's are accumulated *until* the weight update is performed and $\sum_t \Delta w_t$ is reinitialized (reset) to 0 to allow for the next round of accumulation of Δw_t's.

Sutton [1] performed two different experiments which differed in *when* the weight update was performed and how many times the training set was presented to the learner. The details of these experiments and the findings which I have reproduced are contained in the following sections. Note that in both experiments, TD(λ) learning is applied to the same training data. Experiments 1 and 2 differ materially only in how weight updates are scheduled and how the training set is presented.

III. Experiment 1: Training paradigm with repeated presentations of training sets

A. Implementation details

Recall again that we have 100 training sets, each containing 10 sequences of random walks. In the first experiment, the weight update was performed only at the end of each presentation of a training set. This corresponds to setting (1) described previously. For different values of λ, the TD(λ) procedure was performed for different values of α. For each (λ, α)-configuration, the RMSE was calculated for each of the 100 training sets and then averaged. Note that for each training set, the RMSE was calculated using two vectors: (i) the prediction vector that resulted from application of the asymptotic weight vector w at the end of the training set, and the (ii) ideal predictions vector (true probabilities of right-sided termination in state G).

Four important comments need to be made.

1) Likely because the number of times the weight update is performed (relative to the number of individual sequences in a training set) is quite small compared to setting (2), this method required multiple presentations of a training set to allow for sequential updates and eventual convergence of the weight vector w, so that the asymptotic weights, and therefore the predictions $P_t = w^{\mathrm{T}} x_t$, approached the ideal predictions with multiple presentations. In Sutton's paper, the requirement for multiple presentations until convergence is referred to as the "repeated presentations" paradigm.
2) With multiple presentations in this training paradigm, it was important that the learning rate α was small, otherwise the weight vector could easily diverge. Therefore, in our implementation of the experiment, I chose only to perform the grid search for α for 30 values between $\alpha = 0$ and $\alpha = 0.30$.
3) The initialization of the weight vector was designed to be random, with values being drawn from the uniform distribution. In fact, whether or not pre-specified constants (e.g. 0.5) or random values were used, similar final asymptotic values of w were arrived at.
4) Predictions represent probabilities. However, there was no restriction on the elements of the weight vector becoming negative or very large positive numbers in the training process. Therefore, when a prediction was generated ($P_t = w^{\mathrm{T}} x_t$) at the end of each training set, negative or very large values (greater than unity) could appear in the prediction P_t vector. Sutton did not mention if negative probability predictions were replaced by zero [in effect constituting a rectified linear unit (ReLU) filter] or if prediction values larger than 1 were

replaced by 1. I implemented the experiment with and without this process of filtering out invalid probability predictions.

B. Results of implementation

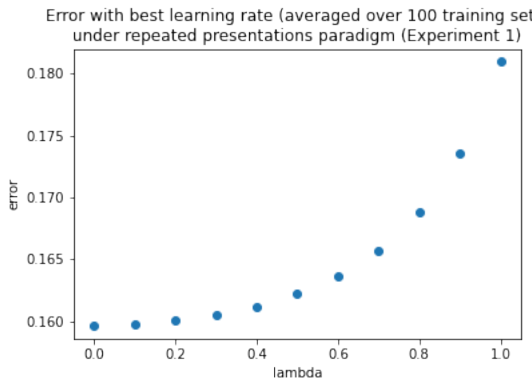

Fig. 1. Root mean squared error with the best learning rate α for various values of λ, averaged over 100 training sets under the repeated presentations paradigm (Experiment 1) without filtering. For each value of λ, a grid search over 30 values of α equally distributed in the range $[0, 0.30]$ was performed, and the smallest average error (among α's) for each λ is presented.

Fig. 1 shows the results of our implementation without filtering out invalid probability predictions. (The graph generated with filtering is virtually identical.) The general shape of the graph demonstrates that the error appears to increase monotonically with increasing λ, as we approach $\lambda = 1$ which is equivalent to an incremental rewrite of the supervised learning algorithm, the Widrow-Hoff procedure. In fact, TD(0) learning appears to perform the best. The relationship that error has with λ is entirely consistent with (corrected) Figure 3 in Sutton's paper.

The results make sense. Temporal difference learning is superior to the supervised learning algorithm because temporal difference learning seeks to minimize error on future experience rather than optimize on the training set. In fact, as Sutton shows in the theoretical part of his paper, TD(0) learning converges with the maximum-likelihood estimate of the underlying Markov process.

Although we have replicated the general pattern of error with different values of λ, the absolute values of the errors in Fig. 1 are not identical to Sutton's paper. This may be due to a number of reasons which are enumerated below.

1) Our data is almost certainly different from Sutton's. Only 100 training sets were used, and each training set only contains 10 sequences. The stochasticity of the data generation process that arises from different random number generators and random number generation algorithms may result in discrepant error terms. Furthermore, for larger values of λ, the TD(λ) procedure is sensitive to overfitting; it learns the data with little capacity for generalization.
2) While I performed grid search for α over the $[0, 0.30]$ interval, it is likely that Sutton used different values of α. The paper mentions that for small α the weight vector would always converged to the same final values, and so we know that Sutton used small values of α. However the exact values of α are not available in Sutton's paper.
3) Sutton describes at a high level the algorithmic principle of repeated presentation of training sets until convergence of the weight vector. Two parameters which are not explicitly stated by Sutton are the maximum number of iterations set in the code (to prevent inadvertent infinite loops that may occur even with small α) and the numeric threshold for determining convergence. In our implementation, due to computational time constraints, a maximum of 1000 presentations was set, and the numeric threshold (tolerance) for determining convergence was the default tolerance used by the `np.isclose()` method, namely 1×10^{-8}.

IV. Experiment 2: Paradigm of single presentation of data with weight updates after each sequence

A. Implementation details

Experiment 2 followed the same general setup outlined in Section IIB of this paper. In particular, it is worth mentioning that the same dataset was used, and that the same general approach holds: for different values of λ, we trained a learner under different learning rate regimes to predict the true probabilities of termination in state G. For each of the 100 training sets, we computed the root mean squared error between predicted probabilities (calculated using asymptotic weights and state vectors) and the ideal predictions. The best hyperparameter configurations of (λ, α) were identified by comparing the average root mean squared error of different (λ, α)-configurations.

Note that Comment 4 in Section IIIA, which relates to removal or non-removal of invalid values in the predicted probabilities vector, applied to Experiment 2 as well. Experiment 2 was conducted under two settings: with and without filtering of invalid probability predictions.

Experiment 2 differs from Experiment 1 in a few important ways:

1) The training set is presented just once to the learner, instead of multiple times until convergence of the weights.
2) During each training set, the learner updates the weights vector w after each sequence in the training set, rather than only (once) at the end of seeing the sequences in the training set.
3) Because there is no concern about multiple presentations leading to explosion of the weights vector w, the learning rates α need not be constrained to very small values. That is, for any given λ, a wider grid of 100 α values over the interval $[0, 0.60]$ was used for the grid search for the α associated with the best error.
4) Because the training set is only presented once, and no convergence requirement is imposed on the learner, there is a risk that random initialization of w similar to what was performed in Experiment 1 may bias the learner

toward termination of the random walk on one of the two sides. Therefore, at the beginning of each training set the weights vector w is initialized to 0.5 in all of its elements.

Although it is clear from Sutton's paper that in Experiment 2, the learner updates w with $\sum_t \Delta w_t$ (which was calculated using the Δw_t's in a sequence) at the end of each sequence in a training set, it is not clear if the accumulator variable $\sum_t \Delta w_t$ is reinitialized to zero with each update.

1) A naive argument could be made that the $\sum_t \Delta w_t$ persists within a training set, since we are looking at the asymptotic behavior of w within a training set, and in that sense w is derived from a learning process applied to a training set which is in effect a concatenation of several sequences. The problem with this interpretation is that numerical overflow may be a larger concern
2) The standard interpretation that the $\sum_t \Delta w_t$ is reset to zero with each update of w may be a more natural assumption.

Given the ambiguity about when the accumulator $\sum_t \Delta w_t$ is reset, I implemented Experiment 2 in two ways: Version 1 (the former interpretation) and Version 2 (the latter).

B. Results of implementation

Similar to our work in Experiment 1, in Experiment 2, we looked at the profile of average RMSE with different values of the discount factor λ. For each discount factor, the best average error among all values of α in the grid search is calculated.

We found that under both settings of filtering and without filtering of invalid probability predictions, we had a similar relationship between error and λ. Because both settings gave similar results, we will only show the figures belonging to the setting without filtering. Version 1 gave rise to a fairly deep U-shaped relationship between error and λ, with the minimum error being achieved between $\lambda = 0.4$ and $\lambda = 0.6$ (Fig. 2). In contrast, Version 2 produced only a very gentle U-shaped curve with a shallow minimum of the error at about $\lambda = 0.2$ to $\lambda = 0.3$ (Fig. 3), which in our opinion is closer to Sutton's result in Figure 5 of his paper. This suggests that the second interpretation of the timing of resetting the accumulator is closer to the truth.

We then looked at a grid of α and λ configurations, similar to Sutton's Figure 4, and evaluated the mean RMSE. Under the setting without filtering, Versions 1 and 2 of Experiment 2 produced virtually identical results. Fig. 4 shows the results under Version 2. Under the setting with filtering out of invalid prediction probabilities, the curves were concave down rather than the general convex shape seen in Fig. 4 as well as Sutton's Figure 4, supporting a hypothesis that Sutton did not filter out invalid probability predictions in his original experiments.

References

[1] Sutton, R.S. "Learning to predict by the methods of temporal differences". Machine Learning 3, 9–44 (1988). https://doi.org/10.1007/BF00115009

[2] Sutton, R. S., & Barto, A. G. (2018). "Reinforcement learning: An introduction". MIT press.

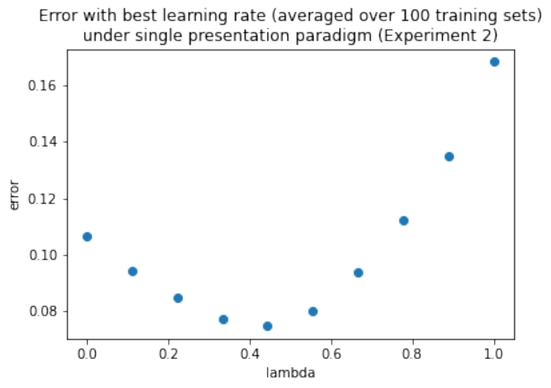

Fig. 2. RMSE with the best learning rate α for various values of λ, averaged over 100 training sets under the single presentation paradigm (Experiment 2, Version 1) without filtering. For each λ, a grid search over 100 values of α equally distributed in the range $[0, 0.60]$ was performed, and the smallest average error (among α's) for each λ is presented.

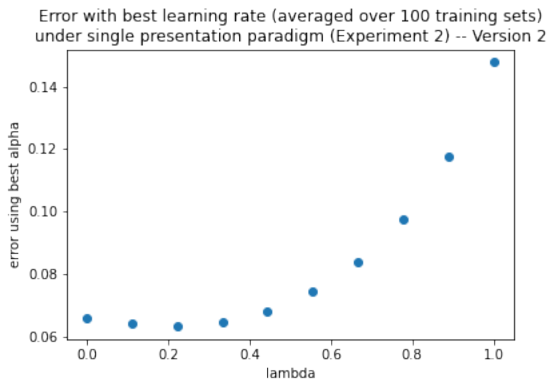

Fig. 3. RMSE with the best learning rate α for various values of λ, averaged over 100 training sets under the single presentation paradigm (Experiment 2, Version 2) without filtering. The methodology is identical to Fig. 2 save for the timing of accumulator reinitialization.

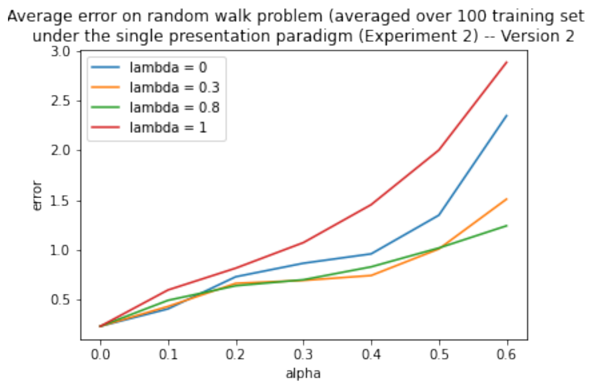

Fig. 4. RMSE with different α and λ hyperparameter configurations, averaged over 100 training sets under the single presentation paradigm (Experiment 2, Version 2) without filtering.