GIT HASH for code: `dcd6eb131cfda50e9d6f693f8d977e5c795415ce`

Prediction Experiments on TD(λ)

Tharun Saranga
College of Computing
Georgia Institute of Technology
Atlanta, USA
tarun.saranga116@gmail.com

***Abstract*—This paper deals with the Reinforcement Learning approach called TD(λ) or Temporal Difference method. The Temporal Difference method is an incremental learning procedure for prediction. Unlike conventional prediction techniques that follow on the difference between the predicted and actual outcome, the TD method assigns credit by using the difference between the temporally successive predictions. The usefulness of the TD method is discussed and try to understand its behavior by using a simple random walk as an example.**

Keywords*—*TD(λ), Temporal Difference method, incremental learning, prediction

I. Introduction

In prediction methods, generally we have to feed in data with labels informing the class it belongs and supervise the algorithm to predict when given with similar type of data. This is usually not scalable for huge datasets and not viable for real-time, real-world scenarios. These are called Supervised learning methods. The Unsupervised learning methods do not require any input from the user to separate the class of the data. The algorithm learns from the given set of data and predicts the future using the past experience. Learning to predict is now-a-days mostly used in predicting game moves and stock prices. These kind of pattern recognition problems can be treated as prediction problems. For these type of problems only the data is provided, not intervention of the supervisor is required in the learning process. The Temporal Difference methods are specialized incremental learning procedures for prediction problems. The conventional methods assign credit by using the difference between the predicted and the actual outcome. The Temporal Difference method uses the temporally incremental difference between two successive states. The TD methods are incremental in nature making the computationally less expensive and are more efficient in using the previous experience than conventional prediction techniques.

II. TD(λ) Learning Procedures

Let us analyse the mathematical concepts and why TD method is superior to conventional supervised learning methods. A typical multi step supervised learning method is fed with a sequence of data of form $x_1, x_2, x_3, x_4 ... x_m, z$ where each x_t is the observation vector available at time t in the sequence and z is the outcome of the sequence. $P_1, P_2, P_3 ... P_m$ are the predicted estimates of z from a particular state. The prediction P_t is the function of the observation vector and another parameter called weights, w. All learning procedures are stipulated as a set of rules for updating the weights parameter w. For experienced gained by the observation a change is made to the weights w as Δw_t. After complete processing of a sequence the changes are accumulated and are added to the initial weights vector.

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

A supervised learning method treats the data as pairs of observation-outcome pairs. A prototypical supervised learning update would be.

$$\Delta w_t = \alpha\,(z - P_t)\, \nabla_w P_t$$

After computing the partial derivative and substituting for P_t we get this.

$$\Delta w_t = \alpha\,(z - w^T x_t)\, x_t \tag{2}$$

But this has a certain disadvantage to it. The method requires that the outcome of the sequence is already known which in most practical cases is difficult. In situations with a training set with already known outcomes it might be useful but in the real world learning case it fails. This is where Temporal Difference learning methods comes to rescue. Instead of using the difference between the predicted and actual outcome, it uses the difference between the two consecutive predictions to make the weight updates. Now the *equation 2* is changed to.

$$\Delta w_t = \alpha\,(Pt + 1 - Pt) \sum_{k=1}^{t} \nabla_w P_k \tag{3}$$

This incremental computation doesn't require to memorize the whole sequence which saves memory and computation power. Now to improve the performance even more the TD(λ) method is used. It just adds a decaying weighted parameter λ that controls the TD(λ) method's behaviour.

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

If lambda = 1 this will be equivalent to *equation 3*. This is TD(1) method. For λ < 1 the weight changes occur different than those made by any supervised learning method. The effect of λ vanishes when λ = 0

III. Problem Definition

To understand more about the TD(λ) method a Random Walk example has been chosen to work with. A bounded random walk is a sequence of random actions that move from one states to another until a boundary is reached. Let us consider a seven state system to simulate the random walk as shown in the figure 1. Random Walk is a sequence of steps that we take in this seven state system. The seven states are A, B, C, D, E, F and G. A and G are the terminal states where the walk terminates. We assume that the walk always starts at the center state D. The states other than the terminal states have equal probabilities for the walk to move to left state or the right state. As the walk sequence continues and eventually ends in the terminal state it is terminated. When it reaches the terminal state G it is terminated with an outcome $z = 1$. When it reaches A the outcome $z = 0$. The equal probability of the intermediate states makes sure that there is an equal probability for ending in left most state or right most state. The learning method estimates the expected value of a particular state that shows the probability of ending in right most state G. The learning method being used is the TD(λ) method which is implemented and analysed in this paper.

As described in the problem definition we consider the five states as A, B, C, D, E, F and G. Let us assume an observational vector x_i that corresponds to each non-terminal state. The vector represents the state which it is in. It is a unit vector with 5 components which are zero except for the component that belongs to the current state. x_i consists of four components as zeros and fifth as 1. As an example an observational vector x_D would be represented as $x_D = [0, 0, 0, 1, 0, 0, 0]^T$. A random walk sequence might be DEFEFG with its vector representation as x_D, x_E, x_F, x_E, x_F, 1. The 1 at the end is the outcome of the sequence for reaching the right most state. The prediction value for a state i at the time t would be $P_t = w^T x_t$. It is the vector multiplication of the weights parameter w transposed and the observation vector. w is a vector representing the weight of each state with the weights of the terminal states as zero. Two computational experiments are performed as below.

IV. Experimental Setup

The environment used to perform this is Python programming environment with numpy library for matrix computations, matplotlib for plotting and random to generate random sequences. A set of 100 test cases with each containing 10 random walk sequences with observation vector and outcome as discussed above. Seven different values are used for lambda and the weight increments are updated using the TD(λ) procedure with ideal prediction values as 1/6 1/3 1/2 2/3 and 5/6 for B, C, D, E and F .

Experiment I

In this experiment the weight vector was updated after a complete presentation. The Δw is accumulated and used to update w only after the complete presentation of a training set. The training set was repeatedly presented until there is no significant changes to it. The performance of the procedure is measured using Root Mean Squared Error between the predictions on the training set and the ideal predictions.

First a set of 100 training set with 10 sequences each was created. Second, the weight vector was initialized with 0.5 for non-terminal states. Next, each training set was presented repeatedly, accumulating the weight differences calculated using TD(λ) rules for a fixed alpha. Next, the accumulated weights are added to the weight vector and repeated until the weights change very less. These steps are performed for each lambda value and the RMSE is calculated. The results are plotted with lambda on x-axis and average RMS Error on the y-axis. The graph plotted is shown in the Figure 2.

The error with $\lambda < 1$ is significantly less. As the lambda value approaches 0 the error decreased. The error seems to be a monotonous curve.

The graph is very similar to the graph in Sutton's experiment, except the error seems to blow up as the λ value increases. The range of error values is inconsistent with the range presented in the Sutton's experiment.

The ambiguity with this experiment is choosing the α value for the experiment as it was not described in the Sutton's experiment. The alpha value for this experiment was chosen by trying different α values which gave the best error result.

Experiment II

The second experiment is to observe the effect of learning rate α on the performance of the algorithm. The experiment is performed with several values of α on different values of λ using the same data as the previous experiment. In this experiment the data is just presented once. The RMSE of each λ is plotted on to a graph.

First a set of 100 training set with 10 sequences each was created. Second, the weight vector was initialized with 0.5 for non-terminal states. Next, each training set was presented once, accumulating the weight differences after each sequence, calculated using TD(λ) rules for the given list of alphas. Next, the accumulated weights are added to the weight vector. These steps are performed for each λ value and the RMSE is calculated. The results are plotted with α on x-axis and average RMS Error on the y-axis with curves showing results from different λ values as shown in the Figure 3.

The learning rate α has a huge effect on the performance of the algorithm. The TD(1) is the worst performer for all values of alpha. TD(0.3) seems to be the best performer. For all $\lambda < 1$ the TD method performed best over a wide range of α values.

From the best α value selected from the above experiment, the learning method is implemented with a single presentation of the data. The RSME values are plotted as in Figure 4.

First a set of 100 training set with 10 sequences each was created. Second, the weight vector was initialized with 0.5 for non-terminal states. Next, each training set was presented once, accumulating the weight differences after each sequence, calculated using TD(λ) rules for a fixed optimal α which is taken from the previous experiment. Next, the accumulated weights are added to the weight vector. These steps are performed for each lambda value and the RMSE is calculated. The results are plotted with λ on x-axis and average RMS Error on the y-axis.

Using the best α from above method the error has dropped significantly for lambda<1. The optimal value of λ is 0.2. For λ=1 the error is still significantly larger than for λ<1.

The graph is similar to that of Sutton's with the exception of the error ranges and the lower part of the curve is not as pronounced as in Sutton's. As lambda value increases the error increases rapidly.

Graphs

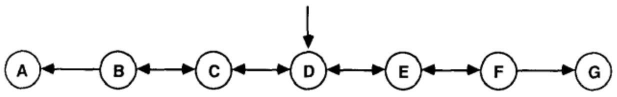

Figure 1

A system of bounded random-walk. This is used to generate the sequences. All walks begin at D and end at either A or G. The intermediate states have a 50% chance of moving either to the left or the right.

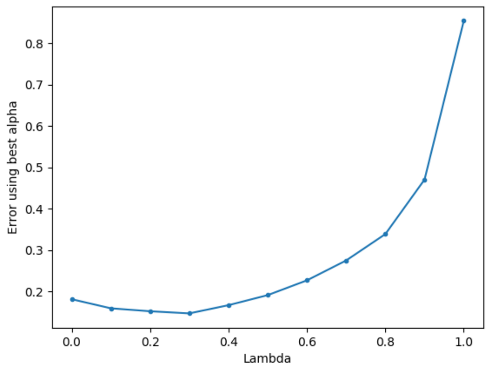

Figure 2

Average RMS error on the random-walk example trained repeatedly until convergence on 100 training sets each with 10 sequences. At λ= 1 is the error by the supervised learning method.

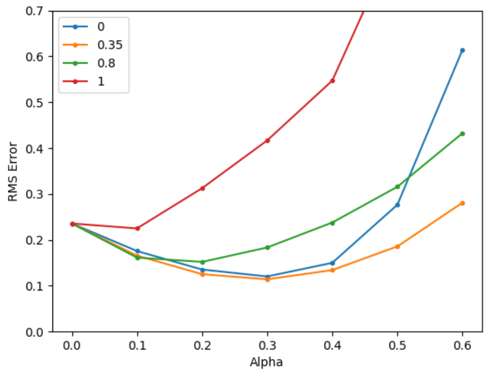

Figure 3

Average RMS error on the random-walk for varying alpha value with different lambdas calculated with single representation of 100 training sets each with 10 sequences. The red line represents the λ= 1.

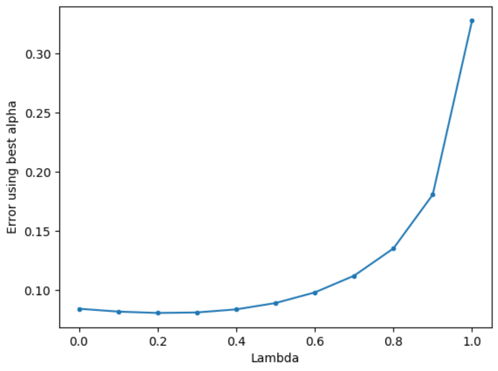

Figure 4

Average RSM error at best α value on random-walk problem. The errors are averaged over 100 training sets of the errors calculated by the TD(λ) procedure. The α value was selected from those shown in Figure 2 to obtain lowest error for that λ value.

Conclusion

The Temporal difference methods perform better than supervised learning methods. The error difference is significantly large. TD methods take less computational power to converge to the same solution because of its iterative nature.