CS 7642 Project 1: TD(λ)

Ali Alrasheed - Ajar3@gatech.edu
Git hash: ce685d526b439ceb565e0822c575a6d9fb6f274c

Abstract

This paper shows an effort to replicate the result of the TD(λ) algorithm introduced in "Learning to Predict by the Methods of Temporal Differences" by (Sutton, 1988). In particular, the random walk example is implemented to replicate the results in figure 3, 4, and 5 in the original paper. The difference between the result will be discussed along with the pitfalls encountered while reproducing the experiments described in the original paper.

1. Introduction

This paper discusses the problem of learning to predict the outcomes of unknown systems or models through experiences to anticipate how they behave in the future. Similar to how humans learn from experiences, a machine can be taught to utilize experiences to predict otherwise unknown system or models.For example, a machine could be taught to predict what the weather could be like tomorrow, and what the stock market return could be in a particular economic settings, and whether a particular chess position is likely to lead to a win.

This paper focuses on TD(λ) technique which is one of the well-known Reinforcement learning algorithms that was introduced in (Sutton, 1988). It is a supervised learning technique that is specialized in solving predictions problems. It uses the idea of *Temporal Difference* (TD) which is a class of incremental learning processes. Instead of updating the predictions based on the error of the current prediction and the final sensor input (outcome), TD updates the predictions based on the difference between the current and previous predictions. This allows learning after each transition as opposed to waiting for the final outcome. Therefore, *Temporal Difference* (TD) consumes less memory, converges faster, and has less RMS error than Windrow-Hoff technique.

1.1. Supervised-learning approach: Widrow-Hoff rule

The supervised-learning approach works by associating a set of observations-outcome sequence. This sequence usually comes from experiences in a form of $x_0, \dots, x_m, z$ where is x_m is the observation at time step m, and z is the actual (real) outcome. These observations are used by the learner to provide prediction of the z in a form of $P_0, \dots, P_m$, where P_m is the prediction at time step m. Generally, P_t is a function of the current observation at time t x_t , and a learnable weight ω. For simplicity, Sutton's paper proposed a prediction based on a linear function of x_t and ω as follows:

$$P(\omega, x) = \omega^T x \tag{1}$$

Therefore, the learner procedure is done through the update ω. Since the supervised-learning approach relies on the final outcome to update ω, the update is accumulated and updated only once as it can be seen from equation 2. It will be shown later how TD can be used to update the weights ω after each observation as opposed to waiting until the final outcome.

$$\omega \leftarrow \omega + \sum_{t=1}^{T} \Delta\omega_t \tag{2}$$

In the supervised-learning approach, the update rule for the weight ω depends on the error between the actual outcome z and the current prediction P, and the gradient of the prediction with respect to the weight ω as follows:

$$\Delta\omega_t = \alpha(z - P_t)\nabla_\omega P_t \tag{3}$$

Using P_t as it is given in equation 1, $\nabla_\omega P_t$ is just x_t, therefore, equation 3 can be reduced to:

$$\Delta\omega_t = \alpha(z - P_t)x_t \tag{4}$$

This is known as the delta rule or Widrow-Hoff rule. It is simple and robust. However, It suffers from several drawbacks. For example, since the update equation 4 depends on the final outcome, updating the weight can only be done after the complete sequence is known. In addition, the update cannot be computed incrementally since all the predictions need to be remembered which is not memory-efficient. Therefore, Sutton, 1988 proposed a method called Temporal-Difference which will be introduced in the next sub-section.

1.2. Temporal-Difference (TD)

Temporal-Difference can produce the same result as Widrow-Hoff rule, but it can be computed incrementally. This can be done by breaking $(z - P_t)$ in equation 4 into a sum of successive prediction as follows:

$$(z - P_t) = \sum_{k=t}^{m}(P_{k+1} - P_k) \tag{5}$$

where $P_{m+1} \equiv z$ Therefore, by combining equations 2, 3, and 5:

$$\omega \leftarrow \omega + \sum_{t=1}^{m} \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \nabla_\omega P_k \tag{6}$$

Converting equation 6 into delta rule as in 3, the new update equation can be given as follows:

$$\Delta\omega = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \nabla_\omega P_k \tag{7}$$

This is known as TD(1) which equivalent to Widrow-Hoff rule. However, TD has several advantages over Widrow-Hoff rule. First, TD can be computed incrementally which means that the the complete sequence does not have to be completed in order to learn (update the weight). This is particularly useful in situations where the sequence is infinite or not complete. Second, TD requires less memory than Widrow-Hoff rule since TD does require storing the all the predictions as opposed to Widrow-Hoff rule.

1.3. TD(λ)

TD(λ) is the general case of TD introduced in the previous section. It provides a exponential weighting to the predictions which gives higher modification weighs for more recent predictions. TD(λ) equation can obtained by adding λ to equation 7 as follows.

$$\Delta\omega = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_\omega P_k \tag{8}$$

A significant advantage of exponential weighting is that it can be also computed incrementally as well. Therefore, the sum in equation 8 can be written as:

$$e_t = \sum_{k=1}^{t} \lambda^{t-k} \nabla_\omega P_k \tag{9}$$

$$= \nabla_\omega P_t + \lambda \sum_{k=1}^{t-1} \lambda^{t-1-k} \nabla_\omega P_k \tag{10}$$

$$= \nabla_\omega P_t + \lambda e_{t-1} \tag{11}$$

Therefore, the final update equation that will be used in this paper can be written as:

$$\omega \leftarrow \omega + \alpha(P_{t+1} - P_t) e_t \tag{12}$$

It must be mentioned that there are two distinct TD(λ) cases, TD(1) and TD(0). When λ is 0, the ω_t and therefore the prediction P is updated based on only one step change in the prediction (Maximum likelihood). However, when λ is 1, the ω_t and therefore the prediction P is updated based change from the first prediction to the actual outcome (Widrow-Hoff procedure). The advantage of TD(λ) is now clear as it having a value of λ between 0 and 1 will provide an update geometrical weighted from the one-step change to the T-step change in predictions.

2. Random Walk Example

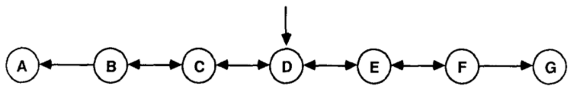

Figure 1. Random walk system from the original paper).

Figure 1 illustrates a random walk example that will be used to evaluate the TD(λ). The sequence starts from D and randomly walks right or left (with equal probability) until it reach a terminating state (A or G). An example of a valid sequence would be DCDBFG. The prediction in this example is defined as the probability of terminating in G from all other states. This means that the outcome z should be 0 when the current state is A, and 1.0 if the current state is G. To reiterate, the prediction P_t is defined as follows:

$$P(\omega, x) = \omega^T x \tag{13}$$

where x = x_B,...,x_F is the observation vector for non-terminating states [B,C,D,E,F]. The vector x uses the one-hot encoding system meaning that x contains zeros for all the states except the current state which should contain 1. For example, x_B is the observation that the current state is B which means that x should be equal to $(1, 0, 0, 0, 0)^T$. Similarly, x_D should be equal to $(0, 0, 1, 0, 0)^T$. Therefore, the gradient of P is given as:

$$\nabla_\omega P_t = x_t \tag{14}$$

By combining equations 11, 12, 13, and 14, we have all necessary information to update the weight. These equations will be used to update the weight for the experiment conducted in section 3.

2.1. Generation of Random Walk sequences

Random walk sequences are needed to generates different training sets in order to conduct several experiments with TD(λ) algorithm. The pseudo-code for generating random walks training sets is illustrated below:

Algorithm 1 Prepare Training Sets

```
function GENERATE_DATASETS(num_datasets, num_seq)
    training_sets = []
    for i in 1,2,...,num_datasets do
        training_set = []
        for j in 1,2,...,num_seq do
            seq = [D]
            $curr\_state = D$
            while s != A and s != G do
                if $rand() < 0.5$ then
                    $curr\_state$ = next state to the right
                else
                    $curr\_state$ = next state to the left
                end if
                seq.append($curr\_state$)
            end while
            training_set.append(seq)
        end for
        training_sets.append(training_set)
    end for
    return training_sets
end function
```

3. Experiments and Results

In this section, several experiments will be demonstrated to duplicate the result in figure 3,4 and 5 in the original paper. The RMS (Root Mean Square) error were used to evaluate the performance in the experiments. The true probabilities of the terminating in state G are $0, \frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{5}{6}, 1.0$ for state A,B,C,D,E,F,G respectively. All non-terminating state (B,C,D,E,F) were initialized to 0.5 to remove any biases.

It must be noted that for all the experiments that will be presented, 100 training sets will generated with 10 sequences per training. The average RMS error will calculated and plotted.

3.1. Experiment 1: Repeated Presentation Training Paradigm

This experiment focuses on evaluating the performance of of TD with different λ values when the training sets are provided infinitely until convergence. First, instead of updating the weight after each transition in the sequence, the weights ω is accumulated and only updated after the end of each sequence. Second, for each training set, the sequences was repeatedly used to updated the weight ω until there is no significant changes in the weight. In addition, different value of λ was evaluated by calculating their RMS error. The pseud-code of experiment 1 can be seen below.

Algorithm 2 Repeated Presentation Training Paradigm

```
function EXPERIMENT_1(training_sets,$\epsilon$)
    initialize $\alpha$
    for each $\lambda$ do
        for each training_sets do
            $\omega$ = [0,0.5,0.5,0.5,0.5,0.5,1.0]
            $\Delta\ \omega$ = [0,0,0,0,0,0,0.0]
            $et$ = [0,0,0,0,0,0,0.0]
            while true do
                for seq in training_set do
                    $et \leftarrow \nabla_\omega P_t + \lambda e_{t-1}$
                    $\Delta\omega \leftarrow \Delta\omega + \alpha(P_{t+1} - P_t)et$
                end for
                if $\epsilon \geq$ max(| $\Delta\ \omega$ |) then
                    break
                end if
                $\omega \leftarrow \omega + \Delta\omega$
            end while
            append RMS error per training set
        end for
        append avg RMS error per $\lambda$
    end for
    return avg RMS error for all $\lambda$
end function
```

Figure 2 (equivalent to figure 3 in the original paper) depicts the RMS error for 8 different values of λ. The overall trend is consistent with figure 3 in Sutton's paper. It can be seen from the figure that any λ value less than 1 produced lower error than when λ is 1 (Windrow-Hoff).

This can be explained by the fact that Windrow-Hoff procedure mainly reduced the training set error but it cannot necessarily generalize well for other training sets. It can be also observed that TD(0) converges to the minimum error among the other value of λ. This means that given the training data can be presented infinitely, TD(0) can guarantee the minimum error as it uses the maximum likelihood estimate of Markov process. Generally, reducing the value of λ will reduce the error if the data is repeatedly presented.

In addition, although figure 2 has the same trend as figure 3 in Sutton's paper, figure 2 has much lower RMS error overall. The RMS error range in Sutton's paper was from 0.17 to 0.25, whereas the replicated experiment produced an error range between 0.12 to 0.19. There are several reasons for this. First, the training sets are randomly generated, different training sets will provides slightly different plot. Second, the convergence criteria was not explicitly described. In this paper, the maximum change of the individual elements of weights is are less 0.001 ,it is considered converged. Third, in Sutton's paper, it is mentioned that α must be small in order for the weights to converges but not specifically mention what value of learning rate was used. In this paper, α of 0.01 was used.

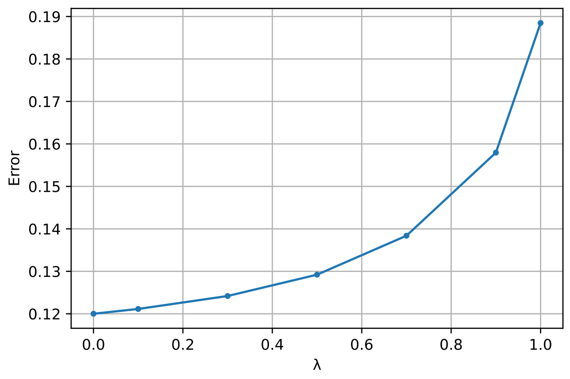

Figure 2. Repeated data training: RMS error vs different value of λ and constant $\alpha = 0.01$

3.2. Experiment 2: Single Presentation Training Paradigm

The second experiment concerns about evaluating the performance of TD(λ) when trained on finite data. It consists of two parts. The first part shows the RMS error for different value of α and λ. The second part is similar to part one but it illustrate the RMS error of different λ with its best value of α (the one that produced the the least RMS error). In other words, as opposed to experiment 1, the weight ω in experiment 2 is updated after each transition in the sequence rather than only at the end. In addition, each sequence is only encountered once as opposed in experiment 1 where each training set was repeated used until the weight converges. The pseudo-code 3 shows how part 1 in experiment 2 is implemented (equivalent to figure 4 in the original paper).

Algorithm 3 Repeated Presentation Training Paradigm

```
function EXPERIMENT_1(training_sets)
    initialize $\alpha$
    for each $\lambda$ do
        for each $\alpha$ do
            for each training_sets do
                $\omega$ = [0,0.5,0.5,0.5,0.5,0.5,1.0]
                $et$ = [0,0,0,0,0,0,0.0]
                for seq in training_set do
                    $et \leftarrow \nabla_\omega P_t + \lambda e_{t-1}$
                    $\omega \leftarrow \omega + \alpha(P_{t+1} - P_t)et$
                end for
                append RMS error per training set
            end for
            append Avg RMS error for given $\lambda$ and $\alpha$
        end for
    end for
    return avg RMS errors for all $\lambda$ and $\alpha$
end function
```

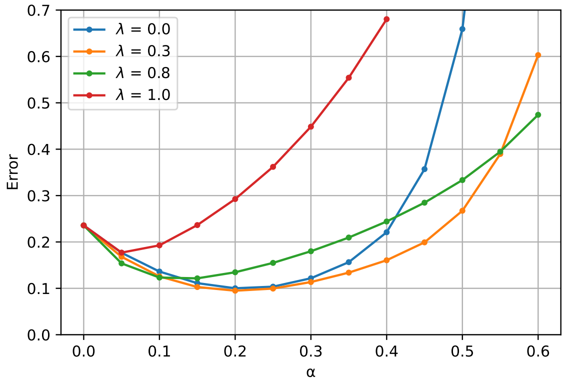

Figure 3. Finite training data: RMS error for different value of λ and α

Figure 3 (equivalent to figure 4 in Sutton's paper) depicts the effect the the learning rate α on the final learning outcome (error). It is obvious from the figure that the minimum error per λ is very sensitive to the learning rate α.In addition, it can be seen that TD(1) produces worst error than any value of λ less than 1. Furthermore, what is interesting is that TD(0) does not perform the best result compared to other value of λ.

The result of Figure 3 has the same overall trend as to figure 4 in Sutton's paper. However, the result 3 seems to produce much higher error for high value of α compared to figure 4 in Sutton's paper. Interestingly, using 5 sequences instead of 10 sequences per training set produced much closer result to the original paper. This could be due to different randomization approaches than the one used by numpy. Another cause could be that original paper used 5 sequence instead of 10 per training set.

Furthermore, the pseudo-code 4 shows how part 2 in experiment 2 is implemented (figure 5 in the original paper).

Algorithm 4 Repeated Presentation Training Paradigm

```
function EXPERIMENT_1(training_sets)
    initialize $\alpha$
    for each $\lambda$ do
        for each $\alpha$ do
            for each training_sets do
                $\omega$ = [0,0.5,0.5,0.5,0.5,0.5,1.0]
                $et$ = [0,0,0,0,0,0,0.0]
                for seq in training_set do
                    $et \leftarrow \nabla_\omega P_t + \lambda e_t$
                    $\omega \leftarrow \omega + \alpha(P_{t+1} - P_t)et$
                end for
            end for
            append min RMS error for given $\lambda$ and $\alpha$
        end for
    end for
    return min RMS errors for all $\lambda$ and $\alpha$
end function
```

Figure 4 (equivalent to figure 5 in Sutton's paper) is also generated by the single presentation training paradigm. However, figure 4 plots the error using best α versus different value of λ. This figure is very similar to figure 3 (equivalent to figure 4 in Sutton's paper) but it is more condensed. It is evident that and value of λ strictly less than 1 produced less average RMS error than when λ is 1 (Windrow-Hoff). It can also be observed that the lowest error in figure 4 is not TD(0) as in figure 2, but a value of λ between 0.1 and 0.4. One possible reason is that TD(0) is much slower in propagating the update through all elements of the weights. This can be avoided in experiment 1 since the sequences are repeated until the weight converges. Again, figure 4 preserves the same overall trend of figure 5 in Sutton's paper but it has a lower overall error. Of course, the randomness of generating the training could play a role on this. However, the probable reason for achieving lower error is that more α values were explored to obtain the minimum error in the replicated experiment. An α value between 0.0 to 0.6 with 0.05 increments were to used to obtain the lowest RMS error per λ. The original paper have not mentioned what value of α were used to obtain the lowest RMS error.

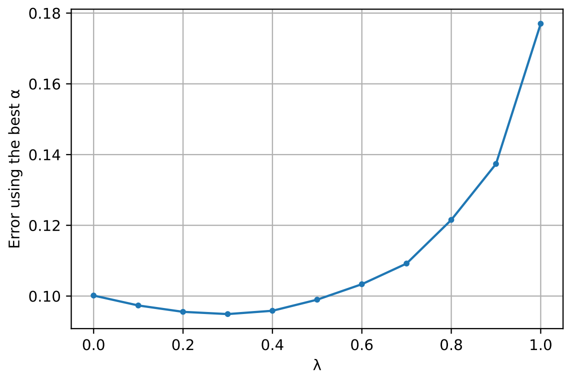

Figure 4. Finite training data: best α RMS error vs different value of λ

4. Conclusion

This report duplicates the random walk experiment in Sutton, 1988. Some experiments were conducted to evaluate the performance of TD compared to the Windrow-Hoff techniques. It was shown that TD method can generate the same learning as Windrow-Hoff through TD(1). However, TD converges faster and it uses less memory. Furthermore, the first experiment primarily shows when the training data is repeated presented, the lower the λ, the lower the prediction RMS error. Similarly, experiment 2 shows that TD(0) does not produce the lowest error when the training data is presented only once, but a value of λ between 0.1 to 0.4.

References

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, *3*(1), 9–44.