Replication of some results from Sutton's TD paper

Khalil Nouili

Abstract

This paper aims to replicate some results from the Richard Sutton's 1988 paper[SUT88] "Learning to Predict by the Methods of Temporal Differences."

Introduction

Learning to predict from past experience is an ever evolving topic. Until today there are new published papers that suggests cutting edge techniques in order to deal with this problem. Richard Sutton as well suggests temporal differences method in his paper while comparing it to the "traditional" approach.

This paper focuses on comparing some results presented by Sutton's TD paper through the random walk example. This paper contains:

- Problem description: understanding the random walk environment.
- A quick look about traditional supervised learning method: Widrow-Hoff and temporal difference.
- A quick look about multi-step and single-step environments.
- Experiments and analysis.
- Conclusion.

Environment description: Random walk

The random walk is the game environment in which we are doing our experiments.

The game rules

The beginning of the game starts at the state D. The player can either go to the right or left. Once the player reached state A or G, the game will be finished with a reward z = 0 or z = 1 respectively.

In this part, I am going to formulate this problem into a Markov decision process for a better understanding:
Agent: The player / The computer
States: {A,B,C,D,E,F,G} with A and G as terminal states.
Actions: Move to the right or left. In our case it's 50% chance either to move right or left.
Rewards: Rewards are received once terminal states are reached. z = 1 if state G was reached, z = 0 otherwise (state A was reached)

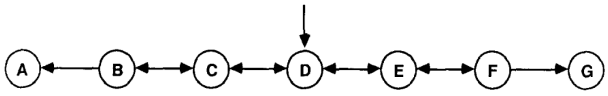

Figure 1: Random walk example game environment

Implementation

Each state will be one hot encoded (e.g., x_D= (0,0,1,0,0)). The experiment is on a training set which is composed of 100 sequences with each sequence contains 10 sub-sequences. A sub sequence represents one game.
(e.g., a game sub-sequence can be (x_D, x_C, x_B, x_A) which will be translated into the following matrix with a reward z = 0 :

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Pseudocode

Algorithm 1: Generate random walk dataset

```
Input : training set size, sequence size
output: The random walk dataset
dataset ← []
for i ← 0 to training set size 1 do
    sequences ←[]
    for j ← 0 to sequence size 1 do
        subSequence ←[]
        newPosition ←2
        while -1 <newPosition <5 do
            currentState ←[0, 0, 0, 0, 0]
            currentState[newPosition] ←1
            subSequence ←subSequence +
             currenState
            newPosition ←
             randomIntBetween(newPosition −
             1, newPosition + 1)
        end
        sequences ←sequences + subSequence
    end
    dataset ←dataset + sequences
end
```

Supervised learning and TD(λ)

[SB14]

Thanks to the TD(λ) and supervised learning methods, our model can learn from the past and predict the future.
In this paper we are going to compare Widrow-Hoff procedure vis a vis TD(λ) but first an in-depth knowledge about these two methods should be acquired.
Before diving deeper into these methods, let's talk about the terminologies used:

- w_t: weights of the learner at timestamp t
- x_t: input (the sequence of states in this case)
- α: learning rate: the step by which the learner is learning to reach the local/global minimum error value.
- m: the size of the training set
- z: the label of the inputted data (the reward in our case)
- $0 \leq \lambda \leq 1$: weights the observations with recency

Widrow Hoff

This supervised learning method behaves the same way as any supervised learning method. It treats the prediction problem by splitting the data into item-label pair. The learner begins by trying to predict the outcome via its weights (initialized by 0.5 for all states) using the following equation:

$$P_t = w^t x_t$$

Once the learner predicts the outcome, it needs to calculate the error between the prediction made and the item label using the following equation :

$$z - P_t = z - w^t x_t$$

Finally a change in the weights must be updated thanks to the error calculated:

$$\Delta_w \leftarrow \alpha(z - P_t)\, x_t$$
$$w \leftarrow w + \sum_{t=1}^{m} \Delta_w$$

As seen by the equations, this approach takes only into consideration the current item-label pair and ignores completely the sequential structure of the data pattern. To overcome this issue the TD(λ) family come into the light.

TD(λ)

As said in the last section, the temporal-difference methods capture the sequential pattern in the data and it does not use only the current item-label pair but also last dates observations.
Temporal-difference cares about changing the weights incrementally between each successive predictions rather than the overall error.
The weights update is sensitive to prediction alteration with a recency weighting as this equation suggests:

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

As a way to compute the sum presented in the previous equation through an incremental process, the following equation might be helpful:

$$e_{t+1} = \nabla_w P_{t+1} + \lambda e_t$$

The more λ is closer to zero, the more the predictions are determined by the most recent events.
As a matter of fact the weight changes will become like this:

$$\Delta_w = \alpha(P_{t+1} - P_t) + \nabla_w P_t$$

Comparing the two methods

As a matter of fact, Widrow-Hoff method is one special case from TD(λ) where $\lambda = 1$.

Single step and multi step training

In this section we will distinguish two kinds of prediction-learning problems:

Single step predictions

At every timestamp, the label of the data is revealed at once and the weights are updates at each time

Multi step predictions

the learner corrects and updates its weights on multi step basis while revealing the partial information of the correctness at each time step.

Experimentations and analysis

In this section, we will compare and analyse the performance of TD(λ) with different values of λ in both single and multi step environment.
The experimentations are conducted using the bounded random walk example with 100 training sets with each training set consists of 10 sequences.
The root mean squared error between the learner predictions and ideal predictions is used in these experiments.

The repeated presentations training paradigm

In this first experiment, the Δ_w are accumulated over sequences and the weights are updated over the complete presentation of a training set. The idea behind the repeated presentation in this process is repeating the weights update until they converge to a certain value. To get a better idea about this technique, a pseudo-code is provided in the next sub section.
This experiment will be conducted on λ with values [0, 0.1, 0.3, 0.5, 0.7, 0.9, 1] and α = 0.01.

Pseudo-code

Algorithm 2: repeatedPresentationMethod

Input : dataset, lambdas, ideaPredicitons
output: errors generated by different λs
$errorsList \leftarrow []$
$learningRate \leftarrow 0.01$
$epsilon \leftarrow 0.001$

for λ ***in*** $lambdas$ **do**
  $accumulatedErrors \leftarrow 0$
  for $trainingSet$ ***in*** $dataset$ **do**
    $currentWeights \leftarrow$[0.5, 0.5, 0.5, 0.5, 0.5]
    $oldWeights \leftarrow$[0, 0, 0, 0, 0]
    while *all elements in [oldWeights - currentWeights] are* $>$*epsilon* **do**
      $\Delta_{WeightAfterATrainingSet} \leftarrow$[0, 0, 0, 0, 0]
      $oldWeights \leftarrow$currentWeights
      for $sequences$ ***in*** $trainingSet$ **do**
        error $\leftarrow [0, 0, 0, 0, 0]$
        $\Delta_w \leftarrow [0, 0, 0, 0, 0]$
        z $\leftarrow theRewardOfSequence$
        for $i \leftarrow 0$ **to** *sub sequence size* 1 **do**
          error $\leftarrow \lambda$ *error + subSequence[i]
          output $\leftarrow$ $TheSumOf(subSequence[i] * weights)$
          if *TerminalStateReached* **then**
            Δ_w += α* (z-output) * error
          end
          else
            nextOutput $\leftarrow$ $TheSumOf(subSequence[i+1] * weights)$
            Δ_w += α * (nexOuput - output) * error
          end
        end
        $\Delta_{WeightAfterATrainingSet}$ += Δ_w
      end
      weights += $\Delta_{WeightAfterATrainingSet}$
    end
    accumulatedErrors += MSE(weights-ideaPredicitons)
  end
  errorsList.Add(errors$_accumulated$)
end

Analysis
As figure 2 suggests, the performance of the learner decreases with λ increase. This can be explained since the temporal difference methods with low values of λ take more into consideration future experiences and tries to find the sequential pattern from the past ones.

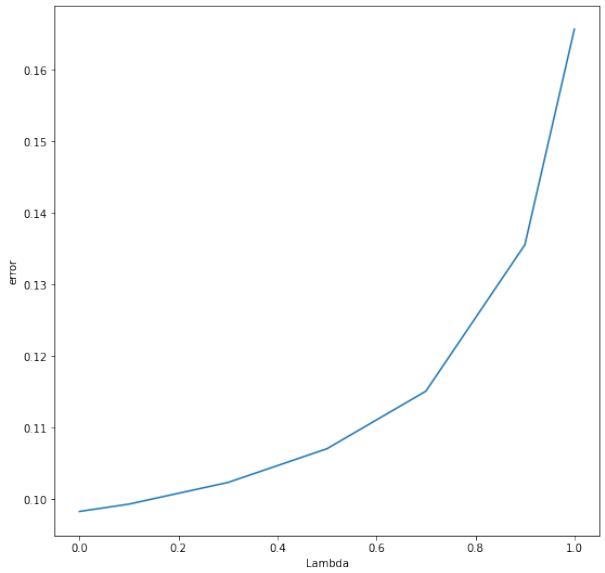

Figure 2: Replication of figure 3 results from Sutton's paper

Trying different learning rates when training set is presented only once

In this experiment, different combinations of learning rates and λ will be tested over the training set just once until convergence. The weights are initially unbiased and set to 0.5.
Here λ takes values in [0, 0.3, 0.8, 1] while the learning rate is in range between 0 and 6.

Pseudo-code

Algorithm 3: multipleAlphas

Input : dataset, lambdas, ideaPredicitons
output: errors generated by different λs
$errorsList \leftarrow []$
$epsilon \leftarrow 0.001$

for λ ***in*** $lambdas$ **do**
  $accumulatedErrors \leftarrow 0$
  for $alpha \leftarrow 0$ **to** 0.65 0.5 **do**
    for $trainingSet$ ***in*** $dataset$ **do**
      $currentWeights \leftarrow$[0.5, 0.5, 0.5, 0.5, 0.5]
      $oldWeights \leftarrow$[0, 0, 0, 0, 0]
      $\Delta_{WeightAfterATrainingSet} \leftarrow$[0, 0, 0, 0, 0]
      $oldWeights \leftarrow$currentWeights
      for $sequences$ ***in*** $trainingSet$ **do**
        error $\leftarrow [0, 0, 0, 0, 0]$
        $\Delta_w \leftarrow [0, 0, 0, 0, 0]$
        z $\leftarrow theRewardOfSequence$
        for $i \leftarrow 0$ **to** *sub sequence size* 1 **do**
          error $\leftarrow \lambda$ *error + subSequence[i]
          output $\leftarrow$ $TheSumOf(subSequence[i] * weights)$
          if *TerminalStateReached* **then**
            Δ_w += α* (z-output) * error
          end
          else
            nextOutput $\leftarrow$ $TheSumOf(subSequence[i+1] * weights)$
            Δ_w += α * (nexOuput - output) * error
          end
        end
        weights += $\Delta_{WeightAfterATrainingSet}$
        accumulatedErrors += MSE(weights-oldWeights)
      end
      $learningRateError$.Add(accumulatedErrors)
    end
  end
  lambdaErrors.Add(learningRateError)
  errorsList.Add(errors$_accumulated$)
end

Analysis
The results shown in figure 3 suggests that α has a huge impact on the learner performance.
For each λ value, there is a certain α value that minimizes the error and builds a better model.
As the last experiment analysis, we see that for every α value, the tradition approach of the supervised learning method (Widrow - Hoff) is performing worst than any other learners.

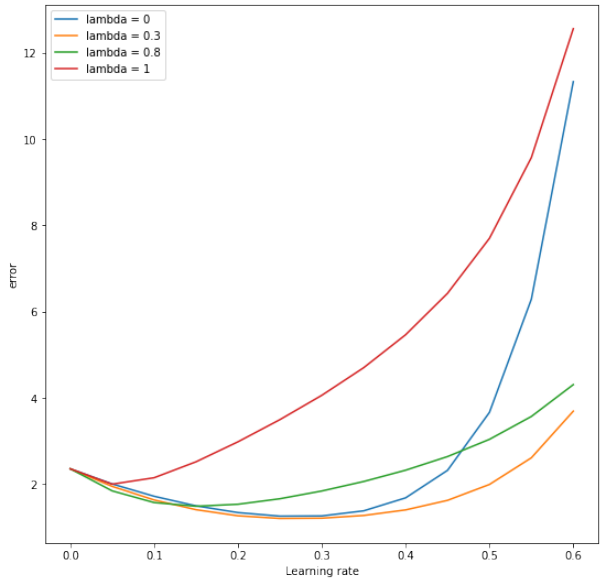

Figure 3: Replication of figure 4 results from Sutton's paper

Best error level achieved for each λ

This experiment is the same as the previous one. The difference here is the use of a larger range of λ values and the result would be getting the best α that corresponds to the minimum error values for each λ.

Analysis
Figure 4 shows that the learner has the minimum error value while λ = 0.3 and the more we move further from it, the more the error gets bigger

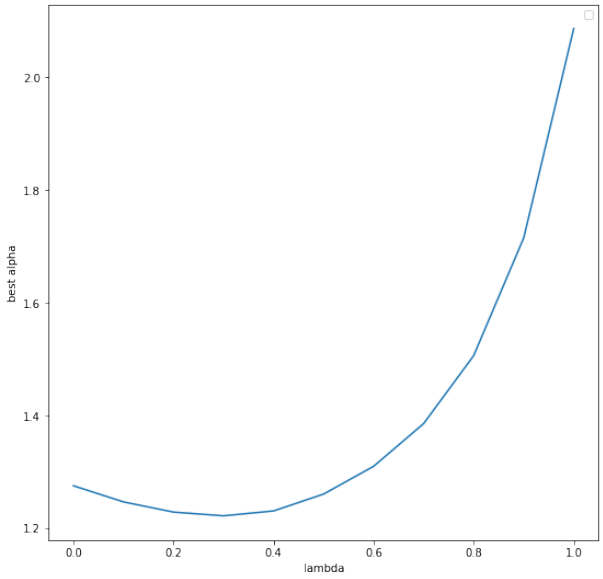

Figure 4: Replication of figure 5 results from Sutton's paper

Conclusion

In this paper, an introduction to the random bounded walk environment, as well as an in-depth knowledge about the tradition supervised learning method: Widrow-Hoff and Temporal difference was made.
After these explanations, some experiments and analysis were conducted in order to compare the performance of these two methods in the random-walk environment both in multi-step and single step training.
At the end from those experiments, not only temporal-difference method was performing better than widrow-Hoff and with the right value of λ, a minimum error value can be reached but also it included an incremental weight updates which was more effective than Widrow-Hoff in the space complexity.

References

[SUT88] RICHARD S. SUTTON. "Learning to Predict by the Methods of Temporal Differences". In: *Machine learning* (1988), pp. 9–23. DOI: https://link.springer.com/content/pdf/10.1007/BF00115009.pdf.

[SB14] Richard S. Sutton and Andrew G. Barto. "Temporal-Difference Learning". In: *Reinforcement Learning: An Introduction* (2014), pp. 143–161.