**CS7642 Reinforcement learning and decision making**

Project 1: Desperately Seeking Sutton

Name: Shuai Jia

Email: sjia61@gatech.edu

Git hash:

**Problem description**

Learning to predict is the basic problem for reinforcement learning and artificial intelligence. In another word, in an unknown or partially known system, using the past experience to predict potential behaviors in the future. For instance, the cloud shape, sizes, humidity, temperature, animals’ behaviors, could be used to predict whether there will be rain or not.

The best known and widely studied learning to predict paradigm is the supervised learning, in which a pair of representation and outcome are provided to train a model via specially designed algorithms, e.g. linear regression and support vector machine. Even though it is easy to understand and analyze, this pairwise approach ignores the sequential structure of the problem. These sequences are quite important to understand the inside of the problem and to achieve better learning. For instance, to predict next Monday’s weather, multi-step observations of Monday, Tuesday, Wednesday, Thursday, Friday, Saturday and Sunday are all important. Therefore, how to sequentially learn from multi-steps of a real problem is highly needed.

**Introduction**

The earliest temporal difference (TD) learning method appeared in Samuel’s celebrated checker-playing program at 1959. After that, it was also applied to various programs and problems and found it worked very well. Nevertheless, this method has remained poorly understood. Theoretical model and theorem have not been established. In 1988, Sutton was the first to view TD in a simpler way, as a method to efficiently learn and predict arbitrary events, not just goal-related things. It made it is capable to evaluate isolated step learning and to achieve formal results. Two significant advantages of TD learning over traditional supervised-learning method are found. The first is it is incremental and easier to compute, which means less memory is used in computation. The second is that it tends to make more efficient use of their experience: it converges faster and predict better results. In this work, we take an example of random walking to show how to use it.

**Random walking**

As shown in Figure 1, we start from state D and move randomly to left or right. Once we arrive the terminal states, A and G state in this example, we will stop and the receive rewards (0 for A and 1 for G). At each step, the walk moves to a neighboring state, either to the right or the left with equal probability. If either state (A or G) is arrived, the walk terminates. For example, a possible step sequence is DCDEFG, and its reward is 1.

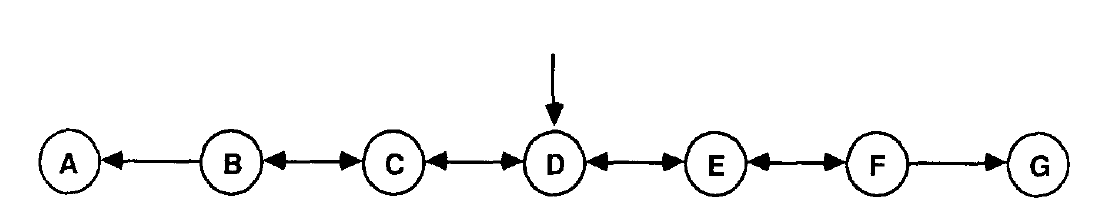


Figure 1. A generator of bounded random walk.

Generally, considering a multi-step observation in the form of x1, x2, x3, … xm, z, where each xt is a vector of observations available at time t in the sequence, and z is the final reward of this sequence. In the random waling example, we use the center 5 states as its observation. For example, the D state in represented as (0,0,1,0,0). Thus an observation sequence could be represented sequentially by the order of observation. Please note the terminal state is a real value, 0 or 1 here. For each observation-outcome sequence, TD method produces a corresponding sequence of predictions P1, P2, P3,….Pm, each of which is an estimate of z. These predictions are usually based on a vector of modifiable weights w. Here, to make it simple, we assume Pt is in a linear function with xt and w.

(Equation 1)

**Temporal learning**

**TD incremental learning**

TD produces exactly the same result as Widrom & Hoff rule, and can be computed incrementally. The error z – Pt could be represented as a sum of changes in predictions (Equation 2)

(Equation 2)

Combined with Widrom & Hoff rule and equation 1, it is easy to derive the following equation 3.

(Equation 3)

Given the above equations, it is easy to turn out that on multi-step prediction problem, the linear TD(1) procedure produce the same per-sequence weight changes as the Widrom & Hoff rule.

**TD (λ) learning**

The hallmark of TD methods is their sensitivity to changes in successive predictions rather than to overall error between predictions and the final outcome. To achieve better predictions than Widrom & Hoff rule, Sutton introduced a family of TD learning procedures which make greater alterations to more recent predictions. Specially, an exponentially reduced weight of lambda is applied to the predictions of observation vectors occurring k steps in the past.

where 0<=λ<=1 (Equation 4)When λ = 1, TD(1) produces the same result as Widrom & Hoff rule. When λ< 1, TD(λ) produces weight changes different from those made by any supervised-learning methods. When λ = 0, the weight increment is determined only by its effect on the prediction associated with the most recent observation.

**Results and discussion**

**True values for weights in random walk**

The true values of weights, namely ideal predictions, could be derived from the Markov process (Equation 5). For this random walking example, Q is the transition matrix, Qij is the transition probability from state i to state j. I is the identical matrix. H is the transition vector from all non-terminal state to terminal state G.

(Equation 5)

The calculated h = [1/6, 1/3, 1/2, 2/3, 5/6].

**Question 1 with repeated presentation paradigm (repeat Figure 3)**

To learn weight changes and predictions in random walk, we first randomly generate 100 training sets with each containing 10 training sequences. Each training sequence has a final outcome, the reward, 0 or 1. The training examples are generated with the following pseudo-code.

**Pseudo-code to generate training sets**

Input: None

Output: 100 trainingsets

Init training\_sets, reward\_sets

for i in range (100):

Init training\_seqs, reward\_seqs

for j in range (10):

Init training\_seq

while cur\_state not in [‘A’, ‘G’]:

next\_state = random choice of getNeighbour(cur\_state)

If next\_state not in [‘A’, ‘G’]:

training\_seq.append(next\_state)

elif next\_state == ‘A’: reward = 0

elif next\_state == ‘G’: reward = 1

cur\_state = next\_state

training\_seqs.append(training\_seq)

end

training\_sets.append(training\_seqs)

end

end

In the repeated presentation paradigm, 10 training sequence in one training set are repeatedly feed to the TD learning procedure with certain λ value and learning rate until convergence. The initial weight as set as 0.5 for all states. The pseudo-code is shown here. Specially, each step in one sequence will update weight by equation 4. However, the weight does not update until the end of this sequence. Then the rest sequences will feed to the learning procedure. A RMSE will be generated by comparing the final weight and true weight. Repeat above process for 100 training sets to get 100 RMSE and their average is plotted in Figure 2. It shows that TD(1) (same result with Widrom & Hoff rule) is higher than that of other λ values. TD(0) has the lowest mean error because it concerns only the most recent state and such result is consistent with the maximum likelihood estimation for the Markov process.

**Pseudo-code to calculate weight in repeated presentation paradigm**

Input: lambda list

Output: average\_err

Init alpha (learning rate), training\_sets, epilson

for each lambda do:

for each training set do:

Init weight w = [0.5, 0.5, 0.5, 0.5, 0.5]

while True:

copy w as w\_old

for each training\_seq do

Init err, delta\_w

for each state do:

calculate Err, and delta\_w

end

add sum delta\_w

update w with sum of delta\_w

if change < epilson: break

calculate err for this training set

end

calculate average\_err

end

A screenshot of a cell phone

Description automatically generated

Figure 2. Average error with repeated presentation paradigm.

**Question 2 with single presentation paradigm (repeat Figure 4)**

Same with repeated presentation paradigm in the above, 100 training sets are generated for training purpose. The initial weights are set as zero for all states. However, in the single presentation paradigm, the procedure with certain λ value and learning rate will update weight at the end of each sequence. It is only feed the training set once, no matter it is converged or not, to achieve final weight. 100 training sets will generate 100 learned weights and their average error is plotted in Figure 3 with learning rate and λ values. It is easily found that the learning rate has a significant effect on the final result. Learning rate in the range of 0.2 to 0.3 usually generates lower error. In addition, no matter how small is the learning rate, TD(1) always get the worst outcome. The corresponding pseudo-code is shown here.

A close up of a map

Description automatically generated

Figure 3. Average error with learning rate in single presentation paradigm for λ = 0, 0.3, 0.8, 1.0.

**Pseudo-code for TD(λ) with single presentation paradigm**

Input: lists of λ values and alpha

Output: average\_err with alpha for certain λ value

for each lambda value do:

for each alpha value do:

for each training set do:

Init weight = [0.5, 0.5, 0.5, 0.5, 0.5]

for each training seq do:

Init err and delta\_w

for each state do:

Calculate err and delta\_w

update w with delta\_w

calculate error for this training set

calculate average error with lambda and alpha pair

end

end

**Question 3 with single presentation paradigm (repeat Figure 6)**

Because learning rate has a significant impact on the TD learning outcome. To achieve best learning, it is better to optimize both λ values and learning rates. In this question, for TD learning procedure with certain λvalue, we first optimized its leaning rate with same method and algorithm in question 2. Finally, we plotted the average weights error using optimized learning rate with λ values in Figure 4. The TD(1) has the poorest predictions. The best λ value for single presentation paradigm is around 0.3.

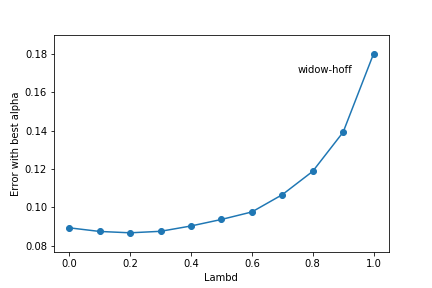


Figure 4. Average error with best learning rate in single presentation paradigm.

**Conclusions**

TD method is able to learn incrementally and predict better for multi-observation outcome sequence. The TD(λ) makes this method is powerful to learn steps difference and predict better. TD(1) has the same result with Widrom & Hoff rule in conventional supervised-learning. However, by optimizing λ value and learning rate, it is able to achieve better learning and predictions.

**Reference**

R.S.Sutton, Learning to Predict by Methods of Temporal Difference, Machine learning 3: 9-44, 1988

M.K. Littman, Algorithms for Sequential Decision Making, PhD dissertation, 1996