Exploration Image: Credit to Photographer https://www.pexels.com/@valentinantonucci

Simple Reinforcement Learning: Temporal Difference Learning

Andre Violante Follow
Oct 29, 2018 · 9 min read

So recently I've been doing a lot of reading on reinforcement learning and watching David Silver's *Introduction to Reinforcement Learning* video series, which by the way are phenomenal and I highly recommend them! Coming from a traditional statistics and machine learning background, in terms of both grad school and work projects, these topics were somewhat new to me. So for my own personal learning and to share that learning with those interested I thought I'd archive it through a Medium post while trying to make these concepts as simple as possible to understand.

. . .

Why Reinforcement Learning

Why would we want to use reinforcement learning versus some other supervised learning approach? To answer that I came up with this timely example:

> *Let's say you're a big fan of the NFL and you want to predict number of wins for the upcoming season. At the beginning of the season you may look at variables such as: past season wins, number of players injured, forecasted weather conditions, number of 1st year starting players, etc. You then fit a model and predict a 9-win season. You have a good team, but an unproven and new quarterback (only 1 game less compared to last year). After the first 5 games your team has a record of 5–0 and your new quarterback is already being hailed as a future hall-of-famer.*

With this example, using a supervised learning model, you would have to wait till the end of the season before learning and making any changes. However, with reinforcement learning you gain information along the way that help make corrections to your original prediction. This application can work well any time there is a temporal or time step component where you'd like to continuously receive information and make adjustments to your target or estimate.

. . .

Example Data

Now let's look at an example using random walk (Figure 1) as our environment. This is an example found in the book *Reinforcement Learning: An Introduction* by Sutton and Barto. You can actually download the digital 2nd edition online for free at their website. The basic idea is that you always start in state 'D' and you move randomly, with a 50% probability, to either the left or right until you reach the terminal or ending states 'A' or 'G'. If you end in state 'A' you get a reward of 0, but if you end in state 'G' the reward is 1. There are no rewards for states 'B' through 'F'.

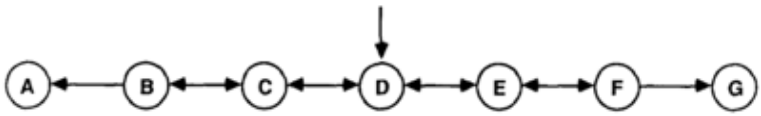

Figure 1: Random Walk Example [1]

The purpose of using reinforcement learning on this example is to see if we can accurately predict the values of each of these states through a model-free approach. The ground truth values are simply the probabilities for each state to gain a reward with respect to ending in state 'G'. So below I've included a table indicating the labels for each of the states that we'll measure our estimates against. Let's talk about our models and what temporal difference is now that we know our data / environment and the target values.

```
-----------------------
| state | probability |
-----------------------
| a     | 0.0         |
| b     | 0.167       |
| c     | 0.333       |
| d     | 0.50        |
| e     | 0.667       |
| f     | 0.833       |
| g     | 1.0         |
-----------------------
```

. . .

Temporal Difference (TD)

Simple Introduction

So to discuss these algorithms I'm going to try and explain them in a simple way. Feel free to reference the David Silver lectures or the Sutton and Barto book for more depth. Temporal difference is an agent learning from an environment through episodes with no prior knowledge of the environment. This means temporal difference takes a model-free or unsupervised learning approach. You can consider it learning from trial and error.

Let's Learn Greek

You will notice in this post some notation and we'll discuss 3 algorithms: TD(0), TD(1) and TD(λ). I will show the equations for each of these and we'll explain them, but lets quickly define the notation for at least some of the hyper parameters (greek letters that are sometimes intimidating).

1. **Gamma (γ):** the discount rate. A value between 0 and 1. The higher the value the less you are discounting.
2. **Lambda (λ):** the credit assignment variable. A value between 0 and 1. The higher the value the more credit you can assign to further back states and actions.
3. **Alpha (α):** the learning rate. How much of the error should we accept and therefore adjust our estimates towards. A value between 0 and 1. A higher value adjusts aggressively, accepting more of the error while a smaller one adjusts conservatively but may make more conservative moves towards the actual values.
4. **Delta (δ):** a change or difference in value.

. . .

TD(1) Algorithm

So the first algorithm we'll start with will be TD(1). TD(1) makes an update to our values in the same manner as Monte Carlo, at the end of an episode. So back to our random walk, going left or right randomly, until landing in 'A' or 'G'. Once the episode ends then the update is made to the prior states. As we mentioned above if the higher the lambda value the further the credit can be assigned and in this case its the extreme with lambda equaling 1. This is an important distinction because TD(1) and MC only work in episodic environments meaning they need a 'finish line' to make an update.

Now lets look at the algorithm and try and make sense of this. Gt (Figure 2) is the discounted sum of all the rewards seen in our episode. So as we're traveling through our environment we keep track of all the rewards and sum them together with a discount (γ). So lets act like we're reading this out loud: the immediate reward (R) at a given point (time, t+1) plus the discount (γ) of a future reward (Rt+2) and so on.

You can see that we discount (γ) more heavily in the future with γ^T-1. So if γ=0.2 and you're discounting the reward at time step 6, your discount value γ become γ^6–1 which equals 0.00032. Significantly smaller after just 6 time steps.

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$$

Figure 2: Sum of Discounted Rewards

Now we'll make an update to our value estimates V(S). **Its important to know that when starting out you really don't have a good starting estimate. You initialize using either random values or all zeros and then make updates to that estimate**. For our random walk sequence we initialize values for all the states between 'B' and 'F' to zero `[0,0,0,0,0,0,1]` . We'll leave the terminal states alone since those are known. Remember we're only trying to predict the values for the non-terminal states since we know the value of the terminal states.

We'll use the sum of discounted rewards from above, Gt, that we saw from our episode and we'll subtract that from the prior estimate. This is called the TD Error. Our updated estimate minus the previous estimate. Then we multiple by an alpha (α) term to adjust how much of that error we want to update by. Lastly we make the update by simply adding our pervious estimate V(St) to the adjusted TD Error (Figure 3).

$$V(S_t) = V(S_t) + \alpha(G_t - V(S_t))$$

Figure 3: TD(1) Update Value toward Action Return

So thats 1 episode. We did 1 random walk and accumulated rewards. We then took those rewards at each time step and compared it to our original estimate of values (all zeros). We weight the difference and adjust our prior estimate. Then start over again. You just learned the TD(1) or MC update!

. . .

TD(0) Algorithm

Now that we've explained TD(1), TD(0) will be much easier to understand. Lets look inside the parenthesis fo Figure 3 cause thats the only difference. Instead of using the accumulated sum of discounted rewards (Gt) we will only look at the immediate reward (Rt+1), plus the discount of the estimated value of **only 1 step ahead** (V(St+1)) (Figure 4).

$$V(S_t) = V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

Figure 4: TD(0) Update Value toward Estimated Return

This is the only difference between the TD(0) and TD(1) update. Notice we just swapped out Gt, from Figure 3, with the one step ahead estimation.

When we use estimates to update estimates we call this bootstrapping. This type of technique has a higher bias then TD(1) or MC because you're making estimates from estimates versus estimates from seeing an entire episode. However, this tends to have lower variance. Another benefit of TD(0) is that it can learn environments that do not have terminal states where TD(1) can not.

. . .

TD(λ) Algorithm

Lets say we want to make value updates before the end of an episode (TD(1)) and use more than a 1 step ahead (TD(0)) for our estimation. This is where TD(λ) comes in. You should know that there are 2 implementations of TD(λ): forward and backward view. The forward view looks at all n-steps ahead and uses λ to essentially decay those future estimates. For this post we'll stay with the backward view of TD(λ), but it has been proven that both forward and backward view are equivalent and Sutton shows that here if interested.

The backward view of TD(λ) updates values at each step. So after each step in the episode you make updates to all prior steps. The question is how do you weight or assign credit to all prior steps appropriately? The answer is by using something called Eligibility Traces (ET). ET basically keeps a record of the frequency and recency of entering a given state (Figure 4). It assigns credit to states that are both visited frequently as well as visited recently with respect to our terminal state. The lambda (λ) and gamma (γ) term are used to discount those traces.

$$E_0(s) = 0$$

$$E_t(s) = \gamma\lambda E_{t-1}(s) + 1(S_t = s)$$

Figure 4: Eligibility Traces

If you think about this intuitively we know that state 'F' will be visited frequently and recently with respect to ending in our final state 'G' and therefore will receive many updates. Therefore ET will assign more credit to state 'F' in proportion to the TD Error (Figure 5, αδtEt(s)). However, state 'B' will not be visited as often with respect to terminating in state 'G' so the value of that state will not be updated as often and will remain close to 0 where it was initialized. From here we make continuous updates to our prior estimates.

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

$$V(s) \leftarrow V(s) + \alpha \delta_t E_t(s)$$

Figure 5: Update with Respect to TD Error (δt) and Eligibility (Et(s))

. . .

Summary

So thats temporal difference learning in a simplified manner, I hope. The gist of it is we make an initial estimate, explore a space, and update our prior estimate based on our exploration efforts. The difficult part of Reinforcement Learning seems to be where to apply it, what is the environment, how do I set up my rewards properly, etc, but at least for now you understand the exploration of a state space and making estimates with an unsupervised model-free approach.

. . .

References

1. Sutton, Richard S. "Learning to Predict by the Methods of Temporal Differences." *Machine Learning*, vol. 3, no. 1, 1988, pp. 9–44., doi:10.1007/bf00115009.

2. Silver, David, director. *RL Course by David Silver—Lecture 4: Model-Free Prediction*. *YouTube*, YouTube, 13 May 2015, www.youtube.com/watch?v=PnHCvfgC_ZA.

3. Free download from the RL Founding Father Richard Sutton, Reinforcement Learning: An Introduction

4. This allows you to write LaTex and save as image. There may be better ones out there but this is what I used. Online LaTex Editor