Chapter 07 – n-step Bootstrapping

Author: Gianmarco Scarano

gianmarcoscarano@gmail.com

1. Introduction

We know that Monte-Carlo approach uses all time steps, hence leading to high variance. Temporal Difference instead, uses a single timestep, but sometimes it's not enough to acquire significant state change.

This leads us to an intermediate approach, called **n-step bootstrapping.**

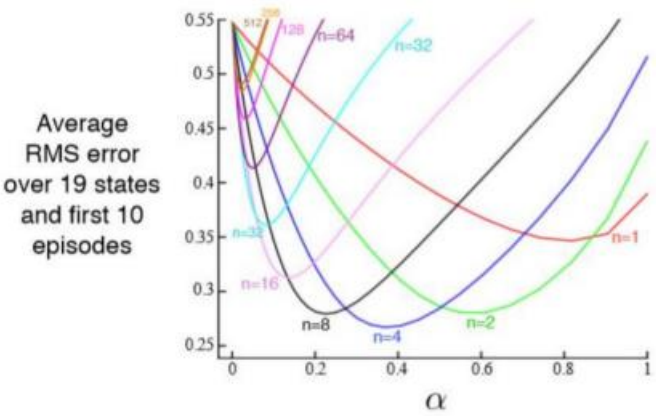
2. N-step bootstrapping

In N-step, returns are still using the temporal difference, but use more than one single reward.

We can think of n-step as follows:

- If we take 1 single step, then we are exactly doing Temporal Difference
- If we take ∞ steps, then we are achieving Monte Carlo
- Otherwise, we simply use n steps (with $G_t^{(n)}$ being the n-step return).
  - $n = 1 \mid y = G_t^{(1)} \Rightarrow r_t + \gamma V(s_{t+1}) \Rightarrow \boldsymbol{TD}!$
  - $n = 2 \mid y = G_t^{(2)} \Rightarrow r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2})$
  - …
  - $n = \infty \mid y = G_t \Rightarrow y_{MC} \Rightarrow \boldsymbol{MC}!$

In this method, we still do our updates using the tabular (moving average) form / function approximation.



In some cases, extremes (like MC and TD) can perform way worse than for normal n values.

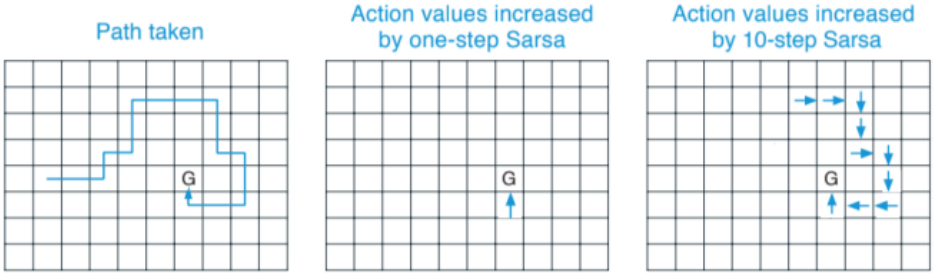
Let's remember that α (which is a number between 0 and 1) is used when we compute the following formula:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_i + \gamma Q(s',\cdot) - Q(s,a))$$

2.1 n-step SARSA

Simply summing all the n discounted rewards, then using SARSA as usual:

$$y = Q_{(n)}{}^{\wedge\pi} = r_t + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n Q^{\wedge\pi}(s_{t+n}, a \sim \pi(s_{t+n}))$$



In the 3rd panel, we see that we propagate the information through the states. So, it's juts not 1 single step, but 10 steps.

3. Choosing n

n = 2 $G^{(2)}$

n = 4 $G^{(4)}$

y = 0.5$G^{(2)}$ + 0.5$G^{(4)}$

The problem now is how to we pick n? Based on what? Actually, we could answer this question by simply averaging n-step returns over different lengths and combining this information.

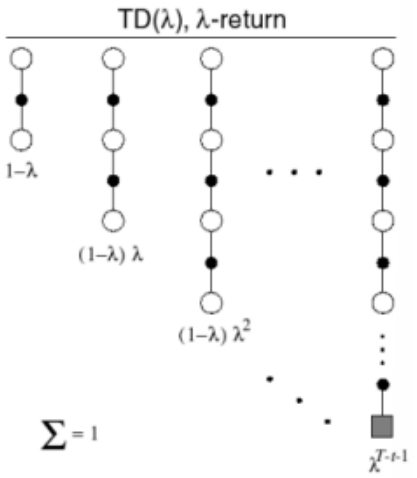
We can combine then, information from all timesteps using λ-returns.

4. λ-returns and TD(λ)

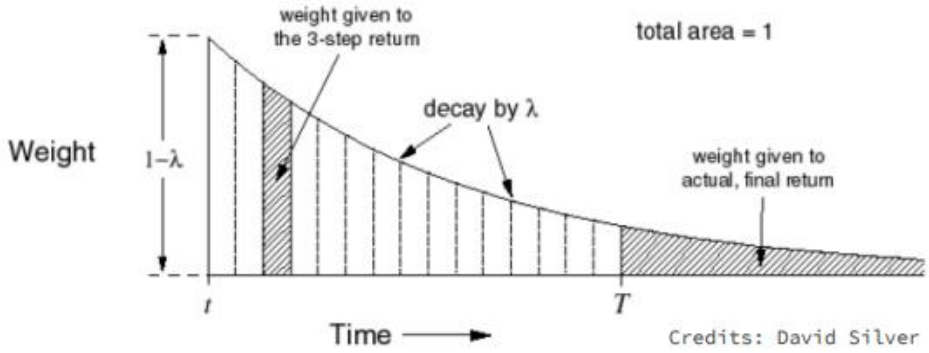
In λ-returns, we combine all n-step returns, returning the following object G_t^{λ} using the following properties:

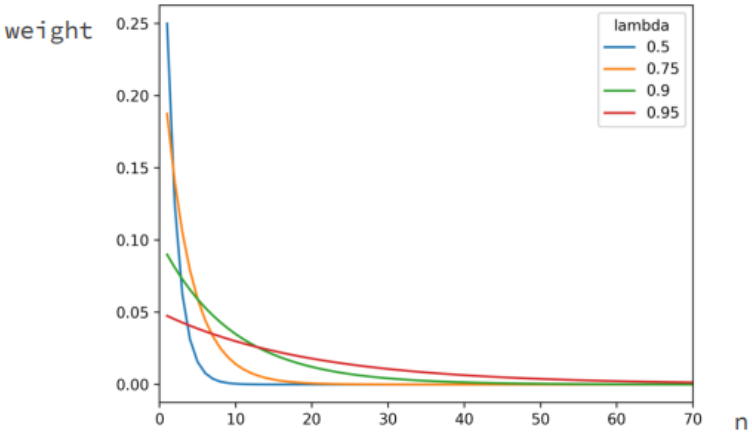
- Using a certain weight: $(1-\lambda)\lambda^{n-1}$
- Doing a weighted average, such that:

$$y = G_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} \cdot G_t{}^{(n)}$$



As for the weights, we have that this weight will decrease in Time (T) by a value of λ. We have a simple plot here that shows what's happening:





As we can see here, if we choose a λ value which is 0.5, we are not taking into consideration returns already from step 8. If we put $\lambda = 0.95$ we are still considering weighted returns up to step 55.

4.1 Forward and Backward-View T(λ)

We can combine all n-steps using λ-returns G_t^{λ}, but we are back at the problem of Monte-Carlo updates, where we have to wait for termination. We can work around this problem by reason about it in different way: through Backward-View via **eligibility traces**.

5. Elegibility Traces

Instead of waiting for what's going to happen next, we will remember what happened in the past.

We will assign a credit to states (or states and actions) as press does with football players when a match finishes.

We'll use two heuristics:

- **Frequency**: Most frequent states get the credit
- **Recency**: Most recent states get the credit

By combining these two, we will keep an **eligibility trace** for all the states s in S.

1. Initialize a trace of a certain state to 0 as: $e_0(s) = 0$
2. Compute traces as:

$$e_t(s) = \gamma\lambda e_{t-1}(s) + \boldsymbol{I}(s_t = s)$$

From the computation number 2, we can already see that the $\boldsymbol{I}$ function is called indicator function, which is equal to 1 if the condition inside is true, 0 otherwise. Also, γ = **Recency** and $\boldsymbol{I}(s_t = s)$ = **Frequency**.

When γ is a value close to 0, it means that the agent focuses on past experiences regardless of how recent they are. As γ approaches 1, it places more emphasis on recent experiences. This is why γ is linked to recency because it determines the weight given to recent state visits when updating the eligibility trace.

$\boldsymbol{I}(s_t = s)$, instead, checks if the current state s_t is equal to the state s for which you are calculating the eligibility trace. If s_t is equal to s, then I $\boldsymbol{I}(s_t = s)$ equals 1, otherwise it equals 0.

In other words, this function checks whether the agent is currently in the state s and adds a $+1$ to this state. The more we add, the more it means we've visited this state.

We will use eligibility traces as scaling factor for the Temporal Difference error.

If a single step TD error is explicated as follows:

$$\delta_t = r_t + \gamma V^{\wedge\pi}(s_{t+1}) - V^{\wedge\pi}(s_t)$$

Now, for all the states s in S, we update the value function estimate as:

$$V^{\wedge\pi}(s) \leftarrow V^{\wedge\pi}(s) + \alpha \cdot \delta_t \cdot e_t(s)$$

The explanation of the Eligibility Traces + TD error + Value Function estimate can be summed up in this way:

δ_t tells the agent whether the current state is better or worse than expected, where $V^{\wedge\pi}(s_t)$ represents the predicted value of the current state under policy π and $V^{\wedge\pi}(s_{t+1})$ represents the predicted value of the next state under the same policy π. r_t represents the immediate reward going from s to s_{t+1}.

When we update our Value Function estimate, it will be updated through the δ_t calculated before, the α term (which is the step size of the update: Smaller α means the updates are more conservative, while a larger α leads to more aggressive updates) and the eligibility trace for that state.

If the TD error δ_t is positive, it means that the agent received a reward that was better than expected when transitioning from the current state to the next state. In this case, we will increase the value estimate of the current state $V^{\wedge\pi}(s_t)$ to reflect this better-than-expected outcome.

Everything can be translated in the other way around: A negative δ_t error will return a negative outcome.

α is a really important parameter, since it tells us if we want the updates to be prominent or really small, maintaining the system stable.

Finally, the combination of the TD error and the update rule with α helps the agent learn to predict the expected cumulative future rewards accurately and adjust its value estimates for different states based on its experiences in the environment.

In other hands, this is still based on the Temporal Difference method and it's still biased because of the bootstrapping.

5.1 λ values

- If $\lambda = 0$, then:

$$e_t(s) = \gamma\lambda e_{t-1}(s) + \boldsymbol{I}(s_t = s)$$

Which means that $e_t(s)$ is simply equal to the indicator function and thus the value function will be updated as in Temporal Difference because all other terms will be multiplied by 0:

$$V^{\wedge\pi}(s) \leftarrow V^{\wedge\pi}(s) + \alpha \cdot \delta_t$$

- If $\lambda = 1$, then the credit is maintained until the end of the episode. As a result, over the course of an episode, the total update to each state is the same as the total update for MC, since the value function is update just at the end of the episode. Here, we are looking at all states (multiplied by 1).

If we consider a state $\boldsymbol{s}$ visited only <u>once</u> at time <u>k:</u>

- $e_t(s) = 0$ if $t < k$
- $e_t(s) = \gamma^{t-k}$ if $t \geq k$

The total updates that **it** cumulates are: $\alpha(G_k - V^{\wedge\pi}(s_t))$

$$\sum_{t=0}^{T-1}\alpha\delta_t e_t(s) = \alpha\sum_{t=k}^{T-1}\gamma^{t-k}\delta_t = \alpha(G_k - V^{\wedge\pi}(s_t))$$

$$\delta_k + \gamma\delta_{k+1} + \gamma^2\delta_{k+2} + \ldots + \gamma^{T-1-k}\delta_{T-1} =$$

$$r_k + \gamma V^{\wedge\pi}(s_{k+1}) - V^{\wedge\pi}(s_k) +$$

$$\gamma r_{k+1} + \gamma^2 V^{\wedge\pi}(s_{k+2}) - \gamma V^{\wedge\pi}(s_{k+1}) +$$

$$\gamma^2 r_{k+2} + \gamma^3 V^{\wedge\pi}(s_{k+3}) - \gamma^2 V^{\wedge\pi}(s_{k+2}) +$$

$$\ldots$$

$$\gamma^{T-1-k} r_{T-1} =$$

$$r_k + \gamma r_{k+1} + \gamma^2 r_{k+2} + \gamma^{T-1-k} r_{T-1} - V^{\wedge\pi}(s_k) = G_t - V^{\wedge\pi}(s_k)$$

More generally speaking, if λ is whatever value, then the total updates it cumulates are:

$$\alpha(G_k{}^{\lambda} - V^{\wedge\pi}(s_t))$$

5.2 Forward/Backward view equivalence

In the forward and backward view, we do offline and online updates. There's a theorem which states that the sum of offline updates is identical for forward-view and backward-view.

Offline updates	$\lambda = 0$	$\lambda \in (0,1)$	$\lambda = 1$
Backward view	TD(0)	TD(λ)	TD(1)
	=	=	=
Forward view	TD(0)	Forward TD(λ)	MC
Online updates	$\lambda = 0$	$\lambda \in (0,1)$	$\lambda = 1$
Backward view	TD(0)	TD(λ)	TD(1)
	=	$\neq$	$\neq$
Forward view	TD(0)	Forward TD(λ)	MC

- Offline = We apply update in batch at the end of the episodes.
- Online = Apply update at every timestep

6. Sarsa-λ

We can embed both forward and backward view into SARSA, having the λ version of it.

Our target y will now become: $(1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\,Q_{(n)}^{\hat{}\pi}$ where $Q_{(n)}^{\hat{}\pi}$ is basically what we had in normal n-step SARSA.

$$Q_{(n)}{}^{\hat{}\pi} = r_t + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n Q^{\hat{}\pi}(s_{t+n}, a\sim\pi(s_{t+n}))$$

In the end then, we can perform our tabular update as follows (remember it's just for that state!):

$$Q(s,a) \leftarrow Q(s,a) + \alpha(y - Q(s,a))$$

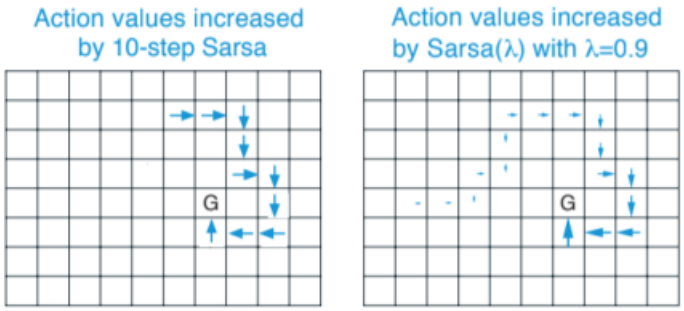
In the backward view, we maintain the eligibility traces for states and actions, initialize them to zero and we do the same thing as before. We update ALL the states and actions.

$e_0(s,a) = 0$ and $e_t(s,a) = \gamma e_{t-1}(s,a) + \mathbf{I}(s_t=s, a_t=a)$
for all s,a

Update for all s,a every time:

$\delta_t = r_t + \gamma Q^{\hat{}\pi}(s_{t+1}, a\sim\pi(s_{t+1})) - Q^{\hat{}\pi}(s_t, a_t)$

$Q(s,a) \leftarrow Q(s,a) + \alpha\delta_t e_t(s,a)$



The information in Sarsa-λ is backpropagated through all the states.

7. N-step Expected Sarsa

The important thing about n-step Expected SARSA is that at the end, instead of sampling an action according to the policy π, I consider the full policy distribution by doing an expectation over the policy.

$$y = Q_{(n)}^{\wedge\pi} = r_t + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n \sum_a \pi(a|s_{t+n}) Q^{\wedge\pi}(s_{t+n}, a)$$

Namely, I'm weighting the Q value by the probability of that exact action.

It's still a stochastic policy, but I'm considering the full possible outcomes of it.

I don't have the transition function, but I can reason about it using the policy itself.

So, we are sampling the reward and the next state from the environment except the last one where we know everything that could happen from the policy, so I consider all the policy actions that the policy could take.

In this case, we would have no sampling but all expectations, here's why $Q(\sigma)$ comes handy.

Q(σ)

Use $\boldsymbol{\sigma}_t$ in [0,1] to denote the degree of sampling at each timestep:

- 1 means full sampling
- 0 means full expectation
- can be a function of the state

At each timestep the TD error is

$$\delta_t = r_t + \gamma(\boldsymbol{\sigma}_t Q^{\wedge\pi}(s_{t+1}, a \sim \pi(s_{t+1})) + (1-\boldsymbol{\sigma}_t) \sum_a \pi(a|s_{t+1}) Q^{\wedge\pi}(s_{t+1}, a)) - Q^{\wedge\pi}(s_t, a_t)$$