Reinforcement Learning
Eligibility Traces - A unified view

Contents

There are two ways to consider eligibility traces in reinforcement learning :

- The theoretical approach (or *forward view*) emphasizes the bridge that can be built between TD(0) and Monte-Carlo methods. Indeed, as we'll see, when TD methods are augmented with eligibility traces, they produce a family of methods spanning a spectrum that has MC methods at one hand and TD(0) at the other.
- The practical approach (or *backward view*) provide a more mechanistic view of the same principle. From this perspective, eligibility traces are a temporary recording of an event (such as the visit of a state-action couple).

After justifying the intuition for the forward view, we'll detail both methods before proving their theoretical equivalence (under certain conditions).

1 n-step predictions

Consider the task of estimating $V^\pi(\cdot)$ from episodes sampled from π. In MC methods, we would perform a back-up for each sampled state based on the entire reward sequence (from the state to the end of the episode). On the other hand, for TD(0) methods, the update is only based on the next reward (and the current estimation of a following state-action couple as a proxy).

It would appear that these methods each describe one end of a more intermediate approach, with backups based on an intermediate number of rewards. More formally, consider the back-up applied to state s_t as a result of the state-reward sequence :

$$s_t, r_{t+1}, \ldots, r_T, s_T$$

In a MC way, the estimate $V_t(s_t)$ of $V^\pi(s_t)$ is updated in the direction of the *complete return* :

$$R_t = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{T-t-1} r_T \tag{1}$$

which would be the target of the back-up. On the other hand, the target of the TD(0) algorithm will be the 1-step return :

$$R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1}) \tag{2}$$

Equivalently, the *n*-step target will write :

$$R_t^{(n)} = \sum_{i=0}^{n-1} \{\gamma^i r_{t+i+1}\} + \gamma^n V_t(s_{t+n}) \tag{3}$$

A n-step backup is defined to be a back-up toward the n-step return. In the tabular case, it writes :

$$\Delta V_t(s_t) = \alpha \left[R_t^{(n)} - V_t(s_t) \right] \quad \alpha > 0 \tag{4}$$

Such a back-up comply with the *error reduction property* :

Error Reduction Property

$$\forall V(\cdot), \quad \max_{s \in \mathcal{S}} \left| \mathbb{E}_\pi \left[R_t^{(n)} | s_t = s \right] - V^\pi(s) \right| < \gamma^n \max_{s \in \mathcal{S}} |V(s) - V^\pi(s)| \tag{5}$$

This namely implies that the expected return of size n under $V(\cdot)$ is a better approximation of $V_\pi(\cdot)$ than $V(\cdot)$ itself ! Under appropriate conditions, this results is used to proved that the n-step backups converge to correct predictions.

It is important to understand that this kind of updates are never used in practice, since we'd have to wait n-steps (and therefore keep them in memory) before performing any kind of backup ! This becomes problematic for large n, and even so for smaller n when it comes to control tasks.

2 The forward view

Back-ups can be done not just toward any n-step return, but towards *any average of them* ! For instance :

$$R_t^a = \frac{1}{2} R_t^{(1)} + \frac{1}{2} R_t^{(2)}$$

This conserves the error reduction property, and is known as a *complex backup*. The TD(λ) algorithm can be understood as a particular way of averaging n-step backups. This average will contain all the n-step returns, each of them being weighted proportionally to λ^{n-1}, with $\lambda \in (0, 1)$. (along with a normalizing factor). The return is known as the λ-return and writes :

$$R_t^{(\lambda)} = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^n R_t^{(n)} \tag{6}$$

Remark : For an episodic MDP, we'll can write :

$$\begin{aligned} R_t^{(\lambda)} &= (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)} + \sum_{T-t}^{\infty} (1 - \lambda) \lambda^n R_t(n) \\ &= (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)} + \lambda^{T-t-1} R_T \end{aligned} \tag{7}$$

For $\lambda = 1$, we clearly have that $R_t^{(1)} = R_T$: this is the constant-α Monte-Carlo method !!

The λ-return algorithm performs updates towards the λ-target :

$$\Delta V_t(s_t) = \alpha \left[R_t^{(\lambda)} - V_t(s_t) \right] \tag{8}$$

and defines the **forward view** of TD(λ).

3 The backward view

The forward view is anti-causal, since at each step we are using information generated by future ones. We describe hereinafter the backward view of TD(λ), providing an incremental mechanism for approximating the forward view. We add to every state an additional memory variable called its eligibility trace, noted $e_t(s) \in \mathbb{R}^+$, $\forall s \in \mathcal{S}$.

On each step, the eligibility traces for all state decay by a factor $\gamma\lambda$, and the eligibility trace for the current step is incremented by 1 :

$$\forall s \in \mathcal{S}, \quad e_t(s) = \begin{cases} \gamma\lambda \cdot e_t(s) & \text{if } s = s_t \\ \gamma\lambda \cdot e_t(s) + 1 & \text{otherwise} \end{cases} \tag{9}$$

This kind of trace is known as an *accumulating trace*. Traces indicate which state is eligible for undergoing learning changes, and to which extent.

At a state s_t, the TD(0) error for state-value prediction is :

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$$

which computation triggers a proportional update to the recently visited states :

$$\Delta V_t(s) = \alpha\delta_t e_t(s), \quad \forall s \in \mathcal{S} \tag{10}$$

which can either be applied online or offline. The backward view of TD(λ) is therefore backward orientated in time. At each moment, we look at the TD error and assign it backward to each prior state, according to the previously visited states.

As before, $\lambda = 0$ gives rise to the simple TD rule, and $\lambda = 1$ to an online MC version for TD(λ). However, this MC method can be used online and for discounted continuing task, increasing the range of applicability for MC-like algorithms.

4 Equivalence between the two formulation

We wish to prove that the off-line TD(λ), as defined in the section above, achieves the same weight updates as the offline λ-return algorithm. We'll let :

- $\Delta V_t^{\lambda}(s_t)$ be the update of $V(s_t)$ according to the λ-return algorithm.
- $\Delta V_t^{TD}(s)$ its equivalent for the TD(λ) algorithm.

Our goal is to show that all the updates over one episode is the same for the two algorithm, ie :

$$\sum_{t=0}^{T-1} \Delta V_t^{TD}(s) = \sum_{t=0}^{T-1} \Delta V_t^{\lambda}(s) \mathbb{1}_{ss_t}, \quad \forall s \in \mathcal{S} \tag{11}$$

Recall that we can write an eligibility traces as $\forall s \in \mathcal{S}$:

$$e_t(s) = \sum_{k=0}^{t} (\gamma\lambda)^{t-k} \mathbb{1}_{ss_k} \tag{12}$$

Therefore :

$$\begin{aligned} \sum_{t=0}^{T-1} \Delta V_t^{TD}(s) &= \sum_{t=0}^{T-1} \alpha\delta_t e_t(s) \\ &= \sum_{t=0}^{T-1} \alpha\delta_t \left\{ \sum_{k=0}^{t} (\gamma\lambda)^{t-k} \mathbb{1}_{ss_k} \right\} \\ &= \sum_{k=0}^{T-1} \alpha \mathbb{1}_{ss_k} \sum_{t=k}^{T-1} (\gamma\lambda)^{t-k} \delta_t \end{aligned} \tag{13}$$

We now focus on the left-hand side of (11). Since we have :

$$\begin{aligned}
\frac{1}{\alpha}\Delta V_t^\lambda(s_t) &= (R_t^\lambda - V_t(s_t)) \\
&= -V_t(s_t) \\
&\qquad + (1-\lambda)(r_{t+1} + \gamma V_t(s_{t+1})) \\
&\qquad + (1-\lambda)\lambda(r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2})) \\
&\qquad + \dots \\
&= -V_t(s_t) \\
&\qquad + (1-\lambda)\left(\sum_{k=0}^{\infty}\lambda^k\right) r_{t+1} + (1-\lambda)\gamma V_t(s_{t+1}) \\
&\qquad + \dots \\
&= -V_t(s_t) \\
&\qquad + (r_{t+1} + \gamma V_t(s_{t+1}) - \lambda\gamma V_t(s_{t+1})) \\
&\qquad + \dots \\
&= \quad (\gamma\lambda)^0(r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)) \\
&\quad +(\gamma\lambda)^1(r_{t+2} + \gamma V_t(s_{t+2}) - V_t(s_{t+1})) \\
&\quad + \dots
\end{aligned}$$

Therefore in the hand :

$$\begin{aligned}
\frac{1}{\alpha}\Delta V_t^\lambda(s_t) &= \sum_{k=t}^{\infty}(\gamma\lambda)^{k-t}\delta_k \\
&= \sum_{k=t}^{T}(\gamma\lambda)^{k-t}\delta_k
\end{aligned} \tag{14}$$

Therefore

$$\sum_{t=0}^{T-1}\Delta V_t^\lambda(s_t)\mathbb{1}_{ss_t} = \sum_{t=0}^{T-1}\alpha\mathbb{1}_{ss_t}\sum_{k=t}^{T-1}(\gamma\lambda)^{k-t}\delta_k = \sum_{t=0}^{T-1}\Delta V_t^{TD}(s) \tag{15}$$

hence proving (11) and the equivalence of the backward and forward view for off-line learning. This result stills holds for on-line learning if $\alpha << 1$ (and therefore we can assume the state value function $V_t(\cdot)$ to stay constant along an episode). More generally, we can expect for TD(λ) and eligibility traces to be similar.

5 Algorithms

5.1 Sarsa(λ)

We wish to use eligibility traces for other task than evaluation - like control. As usual, we will focus on learning $Q_t(s,a)$. We'll need a trace for each (state $\times$ action) couple. We'll just have to generalize the TD(λ) approach for the action-value function. The recursion for the eligibility traces are :

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1 & \text{if } s = s_t,\ a = a_t \\ \gamma\lambda e_{t-1}(s,a) & \text{otherwise} \end{cases} \tag{16}$$

and the update will write :

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a) \quad \forall (s,a) \in \mathcal{S}\times\mathcal{A} \tag{17}$$

with :

$$\delta_t = r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \tag{18}$$

Sarsa(λ) is still an *on-policy* algorithm - we'll evaluate a tabular version of $Q(\cdot,\cdot)$ while following π. Control is performed by ε-greedy of Gibbs sampling, like in TD(0).

Sarsa(λ)

1. *Initialize* $\{Q(s,a)\}_{(s,a)\in\mathcal{S}\times\mathcal{A}}$
2. *Repeat* (each episode)
   a) Initialize $(s,a)\in\mathcal{S}\times\mathcal{A}$
   b) Take a, observe s, r
   c) Select a' from a soft policy derived from $Q(\cdot,\cdot)$
      $\delta_t = r + \gamma Q_t(s',a') - Q_t(s,a)$
      $Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a) \quad \forall (s,a)\in\mathcal{S}\times\mathcal{A}$
      $e(s,a) \leftarrow 1 + e(s,a)$
      $e(s,a) \leftarrow \gamma\lambda \cdot e(s,a)$
   d) $s' \leftarrow s$, $a' \leftarrow a$

5.2 Watkins's Q(λ)

For adapting eligibility traces to off-line methods like Q-learning, a somehow more thorough reflection needs to be conducted. Suppose we are backing up from (s_t, a_t). The agent select the greedy action during the next two steps, but on the third it follows an exploratory action (non-greedy). Here, we can not use more than a 3-step return !

Unlike Sarsa(λ), Watkin's Q(λ) does not look ahead all the wat to the end of the episode to backup, but only until the next exploratory actions. Aside from this, Watkin's Q(λ) is extremely close to TD(λ). If (a_{t+n}) is the first exploratory action, then the longest backup is toward :

$$r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n \max_{a\in\mathcal{A}(s_{t+n})} Q(s_{t+n}, a) \tag{19}$$

Therefore, the backward view for Watkin's Q(λ) is also very simple : the trace's update will write :

$$e_t(s,a) = \mathbb{1}_{ss_t}\mathbb{1}_{aa_t} + \begin{cases} \gamma\lambda e_{t-1}(s,a) & \text{if } Q_{t-1}(s_t,a_t) = \max\limits_{a\in\mathcal{A}(s_t)}(Q_{t-1}(s_t,a_t) \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

First, all traces are decayed by $\gamma\lambda$ or brought back to 0. Then we increment by 1 the current state×action couple. We can then apply the update rule :

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a) \quad \forall (s,a)\in\mathcal{S}\times\mathcal{A} \tag{21}$$

with

$$\delta_t = r_t + \gamma \max_{a\in\mathcal{A}(s_{t+1})} Q(s_{t+1},a) - Q_t(s_t,a_t) \tag{22}$$

Watkin's Q(λ)

1. *Initialize* $\{Q(s,a)\}_{(s,a)\in\mathcal{S}\times\mathcal{A}}$
2. *Repeat* (each episode)
   a) Initialize $(s,a)\in\mathcal{S}\times\mathcal{A}$
   b) Take a, observe s, r
   c) Select a' from a soft policy derived from $Q(\cdot,\cdot)$
      $\delta_t = r_t + \gamma \max_{a\in\mathcal{A}(s_{t+1})} Q(s_{t+1},a) - Q_t(s_t,a_t)$
      $Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a) \quad \forall (s,a)\in\mathcal{S}\times\mathcal{A}$
      $e(s,a) \leftarrow$ (20)
   d) $s' \leftarrow s$, $a' \leftarrow a$

Unfortunately, cutting off traces every time a non-greedy action is selected is drying many of the advantages of using eligibility traces. There will indeed rarely be backups of more than one or two steps.

5.3 Peng's Q(λ)

Peng's Q(λ) is an alternate version of $Q(\lambda)$ aimed to remedy to the few long back-ups performed by Watkin's Q(λ). It can be thought as an hybrid between Sarsa(λ) and Watkin's Q(λ). Conceptually, it uses a mixture of back-ups without distinctions between greedy and exploratory actions. Each component back-up s over many step of actual experience, and all but the last are capped by a final minimization over actions. Therefore, they are never on or off policy : the earlier transitions are on policy, where the last one uses the greedy policy.

If there exist no proof of convergence for Peng's Q(λ), it often shows better practical performances that Watkin's Q(λ).

6 Advanced topics

6.1 Replacing traces

In some cases, significantly better performances can be reached by using a slightly modified king of trace known as replacing trace. It operates as follows :

$$e_t(s) = \begin{cases} 1 & \text{if } s = s_t \\ \lambda\gamma e_{t-1}(s) & \text{otherwise} \end{cases} \qquad \forall s \in \mathcal{S} \tag{23}$$

meaning that the trace of the currently visited state is lifted up to 1 instead of being incremented by 1. In the un-discounted case, this approach identifies to the MC first-visit algorithm.

6.2 Variable λ

We could generalize our previous approach to more general λ, by allowing it to vary from one step to another. For evaluation purposes, we could use the following trace update :

$$e_t(s) = \begin{cases} \lambda_t\gamma e_{t-1}(s) + 1 & \text{if } s = s_t \\ \lambda_t\gamma e_{t-1}(s) & \text{otherwise} \end{cases} \qquad \forall s \in \mathcal{S} \tag{24}$$

Intuitively, the idea is to vary λ_t to be a function of the state (or more precisely of the confidence in the state's value estimation.

→ If the state has been visited many times, and the state-value function is supposed to be well-known at this state, we might want to use it fully. We'll switch λ to be near 0, therefore re-setting all traces to 0 after the update.

→ Similarly, states whose value estimates are highly uncertain will be given λ values near 1. Their TD update will therefore have little effect (they will be skipped until a high-confidence state is reached again).

7 Conclusion

Eligibility traces in conjunction with TD error provide an efficient, incremental way of shifting and choosing between MC and TD methods. The question of the choice of λ can be tricky, and has no theoretical answer. It indeed greatly depends on the nature of the policy being learned (reward sparsity, episode length, ..). Although, choosing for a good compromise ($\lambda \simeq 0.75$) between TD and MC usually leads to a well-behaved learning.