Reinforcement Learning

Chapter 3: Model-free RL

Ali Bereyhi

ali.bereyhi@utoronto.ca

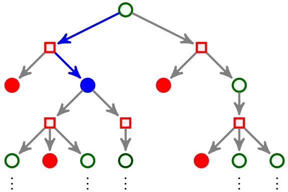
Department of Electrical and Computer Engineering

University of Toronto

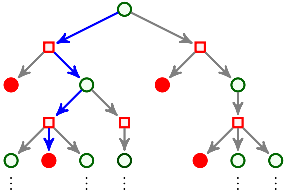
Fall 2025

Monte-Carlo vs TD-0: *Spectrum*

Temporal Difference



Monte-Carlo



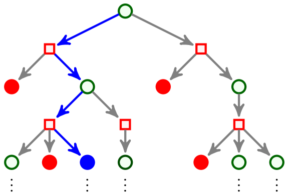
TD-0 and Monte-Carlo are two extreme sides of a *spectrum*

- *In TD-0, we use sample trajectory only for one step*
- *In Monte-Carlo, we use the complete sample trajectory for each state*

Can we draw a solution between these two extreme points?

First Solution: *Bootstrapping with More Steps*

A primary approach to find such a balanced solution is to *extend the idea of bootstrapping to a larger number of steps*



+ *How can we do it?*
– Well! We could simply expand the recursive property of value function

n-Bootstrapping

Looking at a sample return, we can simply write

$$\begin{aligned} G_t &= R_{t+1} + \gamma G_{t+1} \\ &= R_{t+1} + \gamma R_{t+2} + \gamma^2 G_{t+2} \\ &\vdots \\ &= R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^n R_{t+1+n} \gamma^{n+1} G_{t+1+n} \end{aligned}$$

Now, assuming that *at time step t we are at state s*, i.e.,

$$S_t = s, A_t \xrightarrow{R_{t+1}} S_{t+1}, A_{t+1} \xrightarrow{R_{t+2}} \cdots \xrightarrow{R_{t+1+n}} S_{t+1+n}$$

we can bootstrap over a longer part of sample trajectory

n-Bootstrapping

$$S_t = s, A_t \xrightarrow{R_{t+1}} S_{t+1}, A_{t+1} \xrightarrow{R_{t+2}} \cdots \xrightarrow{R_{t+1+n}} S_{t+1+n}$$

In this trajectory, we can expand Bellman equation with deeper recursion

$$v_\pi(s) = \sum_{i=0}^{n} \gamma^i \mathbb{E}\left\{R_{t+i+1} | s\right\} + \gamma^{n+1} \mathbb{E}_\pi \left\{v_\pi\left(S_{t+n+1}\right) | s\right\}$$

So, if we have K sample trajectory, we can estimate value of state s by n steps of bootstrapping, i.e.,

$$\hat{v}_\pi(s) = \frac{1}{K} \sum_{k=1}^{K} \underbrace{\left(\sum_{i=0}^{n} \gamma^i R_{t+i+1}[k] + \hat{v}_\pi\left(S_{t+n+1}[k]\right)\right)}_{\text{computed on sample } k}$$

n-Bootstrapping $\equiv$ *TD-n*

Let's formulate this approach: *for a given sample trajectory and value function estimator $\hat{v}_\pi(\cdot)$, we define n-bootstrapping return at time t as*

$$G_t^n = \sum_{i=0}^{n} \gamma^i R_{t+i+1} + \gamma^{n+1} \hat{v}_\pi(S_{t+n+1})$$

Given that the sample trajectory was started at state $S_t = s$, we can use online averaging and update the value estimator of state s as

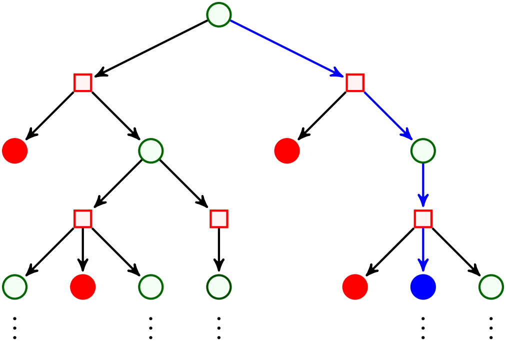
$$\hat{v}_\pi(s) \leftarrow \hat{v}_\pi(s) + \alpha(G_t^n - \hat{v}_\pi(s))$$

This is what we call TD-n method of learning

This is obviously more general than TD-0! In TD-0, we had simply $n = 0$

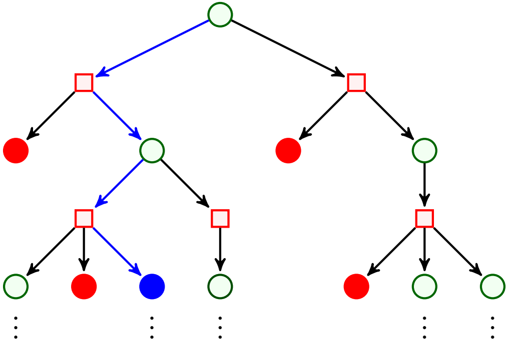
Backup Diagram: *Sampling with n-Bootstrapping*

With n-bootstrapping we sample $(n+1)$-step trajectories from action-state tree of environment



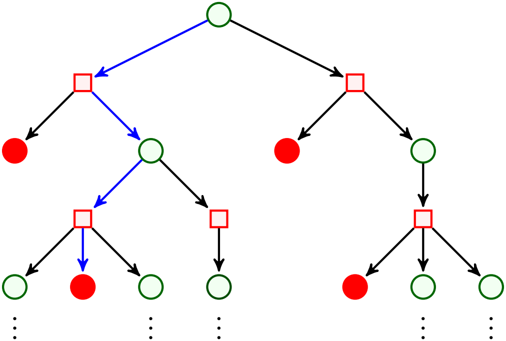
Backup Diagram: *Sampling with n-Bootstrapping*

With n-bootstrapping we sample $(n+1)$-step trajectories from action-state tree of environment



Backup Diagram: *Sampling with n-Bootstrapping*

With n-bootstrapping we sample $(n+1)$-step trajectories from action-state tree of environment



TD-n: *Policy Evaluation*

We can extend our TD-0 evaluation algorithm to TD-n

```
TDn_Eval(π):
```

1: *Initiate estimator of value as* $\hat{v}_\pi(s) = 0$ *for all states*
2: ***for*** *episode* $= 1 : K$ ***do***
3: *Initiate with a random state* S_0 *and act via policy* $\pi(a|s)$
4: *Sample a trajectory until either a terminal stated or some terminating* T

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

5: ***for*** $t = 0 : T - n - 1$ ***do***
6: *Compute* $G = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n+1} v_\pi(S_{t+n+1})$ ⋆
7: *Update as* $\hat{v}_\pi(S_t) \leftarrow \hat{v}_\pi(S_t) + \alpha\left(G - \hat{v}_\pi(S_t)\right)$
8: ***end for***
9: ***end for***

TD-n: *Policy Q-Evaluation*

Same-wise, we can extend our algorithm for action-value computation

`TD`n`_QEval(`π`):`

1: *Initiate estimator of value as* $\hat{q}_\pi(s, a) = 0$ *for all states*
2: **for** *episode* $= 1 : K$ **do**
3: *Initiate with a random state-action pair* (S_0, A_0) *and act via policy* $\pi(a|s)$
4: *Sample a trajectory until either a terminal stated or some terminating* T

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

5: **for** $t = 0 : T - n - 1$ **do**
6: *Compute* $G = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n+1} v_\pi(S_{t+n+1})$ ⋆
7: *Update as* $\hat{q}_\pi(S_t, A_t) \leftarrow \hat{q}_\pi(S_t, A_t) + \alpha(G - \hat{q}_\pi(S_t, A_t))$
8: **end for**
9: **end for**

TD-∞: *Going Back to Monte-Carlo*

+ *You told us that we look for a solution between TD-0 and Monte-Carlo! I see TD-0 is TD-n with $n = 0$, but where does Monte-Carlo stand?*
– Well! You may see already that *Monte-Carlo* is *TD-∞*

Say the environment is episodic: we can say that if we bootstrap very deep; then, at some point we hit a terminal state, i.e.,

$$\lim_{n\to\infty} v_\pi \left(S_{t+n+1}\right) = 0$$

This concludes that at any time t in an episodic environment

$$G_t^\infty = \sum_{i=0}^{\infty} \gamma^i R_{t+i+1} = G_t$$

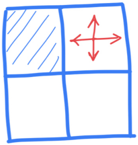
and hence TD-∞ will update its estimator by

$$\hat{v}_\pi \left(s\right) \leftarrow \hat{v}_\pi \left(s\right) + \alpha\left(G_t - \hat{v}_\pi \left(s\right)\right)$$

TD-n: *A Discrete Spectrum*

We can look at TD-n as a discrete spectrum between Monte-Carlo and TD-0

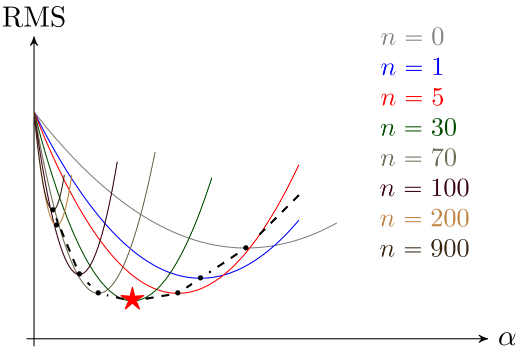
Example: *Dummy Grid World with Random Walk*



Once you got home, try to get back to our dummy world and use TD-n method for multiple n to compute the values for uniform random policy ☺

Typical Behavior: *Variation Against Depth*

If you do some practice with random walk, you will see following curves for different choices of n



We observe a minimum against n: this is a typical behavior! *Any illustration?*

Averaging Different Depth

+ *How deep we should then bootstrap?*
– Well! We could try to find the best one for each setting, or *we could average the result of multiple bootstrapping depths*

For example, we can

1: *Initiate* $\cdots$
2: **for** *episode* $= 1 : K$ **do**
3: $\cdots$
4: **for** $t = 0 : T - 3$ **do**
5: *Compute* $G_0 = R_{t+1} + \gamma v_\pi (S_{t+1})$
6: *Compute* $G_2 = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi (S_{t+3})$
7: *Set* $G = (G_0 + G_2) / 2$
8: *Update as* $\hat{v}_\pi (S_t) \leftarrow \hat{v}_\pi (S_t) + \alpha (G - \hat{v}_\pi (S_t))$
9: **end for**
10: **end for**

Averaging Different Depth: λ-*Return*

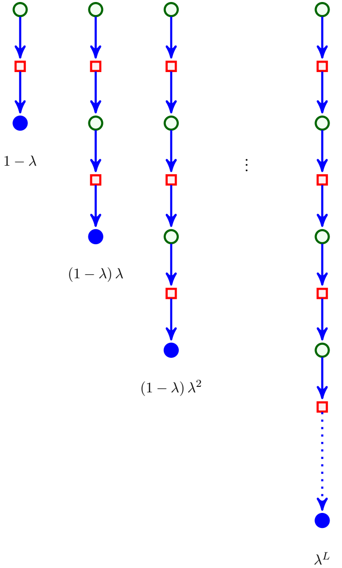
+ *Why should that be a good idea?*
– Because *if the good one is within the average; then, it could dominate and improve* estimation
+ *Then, which ones we should take? We still have no clue!*
– Let's take all of them! We can do it by *geometric weights!*

λ-Return

For any $0 \leqslant \lambda \leqslant 1$, *the* λ*-return at time* t *over* L *steps is defined as*

$$G_t^\lambda = (1-\lambda) \sum_{n=0}^{L-1} \lambda^n G_t^n + \lambda^L G_t^L$$

Averaging Different Depth: λ-*Return*



For each state in the sample trajectory, we can

- *Compute* 0*-bootstrapping return*
  - ↳ *Give it weight* $1-\lambda$
- *Compute* 1*-bootstrapping return*
  - ↳ *Give it weight* $(1-\lambda)\lambda$

⋮

- *Compute* $(L-1)$*-bootstrapping return*
  - ↳ *Give it weight* $(1-\lambda)\lambda^{L-1}$
- *Compute* L*-bootstrapping return*
  - ↳ *Give it weight* λ^L

TD_λ: *Policy Evaluation*

We can evaluate policy by averaging its λ-returns over multiple episodes

$\texttt{TD\_Eval}_\lambda(\pi)$:

1: *Initiate estimator of value as* $\hat{v}_\pi(s^n) = 0$ *for* $n = 1 : N$
2: **for** *episode* $= 1 : K$ **do**
3:     *Initiate with a random state* S_0 *and act via policy* $\pi(a|s)$
4:     *Sample a trajectory until either a terminal stated or some terminating* T

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

5:     **for** $t = 0 : T-1$ **do**
6:         *Set* $G \leftarrow G_t^\lambda$ *towards end of trajectory*
7:         *Update as* $\hat{v}_\pi(S_t) \leftarrow \hat{v}_\pi(S_t) + \alpha(G - \hat{v}_\pi(S_t))$
8:     **end for**
9: **end for**

TD-λ: *Special Cases*

It is easy to see that $\lambda = 0$ and $\lambda = 1$ are again two extreme cases: *assume we are dealing with an episodic environment*

- *with* $\lambda = 0$
  - ↳ *All weights are zero but that of* G_t^0 *which is weighted one*

$$G_t^\lambda = R_{t+1} + \gamma G_{t+1}$$

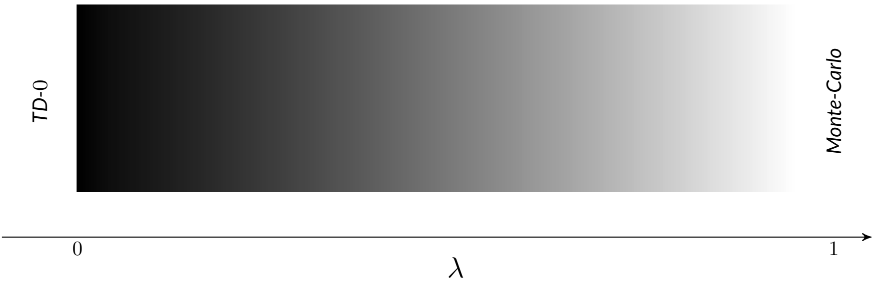
  - ↳ *this is basic bootstrapping: TD$_0$ is hence simply TD-0*
- *with* $\lambda = 1$
  - ↳ *All weights are zero but that of* G_t^{T-t} *which is weighted one*

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

  - ↳ *this basic Monte-Carlo: TD$_1$ is Monte-Carlo*

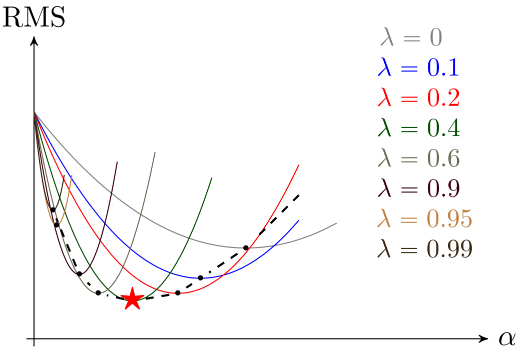
TD$_\lambda$: *A Continuous Spectrum*

We can look at TD$_\lambda$ as a continuous spectrum between Monte-Carlo and TD-0



Typical Behavior: *Variation Against λ*

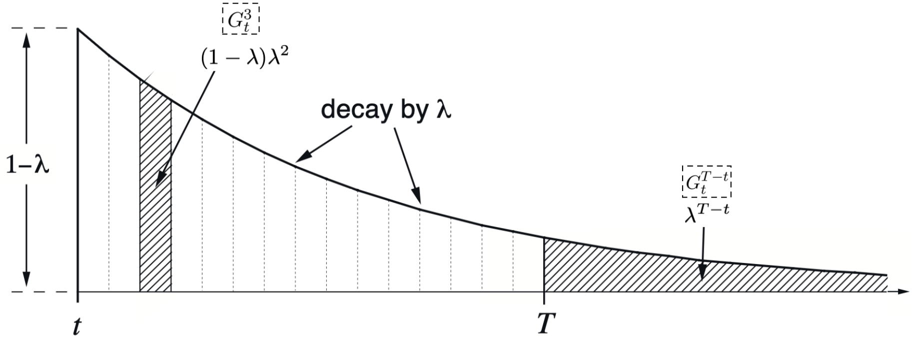
If you do some practice with random walk, you will see following curves for different choices of λ



We observe a minimum against λ: this is analog to TD-n behavior!

TD$_\lambda$: *Weighting Function*

Let's look at the weights decay in λ-return[1]



[1] *From Sutton and Barto's book in Chapter 12*

Assigning Credit for Future by *Weighting*

+ *What is the intuition behind computing the λ-return with those weights?*
- This is a very valid question! Let's see!

Let's look back at TD$_\lambda$ approach: at each time t in the trajectory, we compute

$$G_t^\lambda = \sum_{n=1}^{T-t-1} w_t^n G_t^n$$

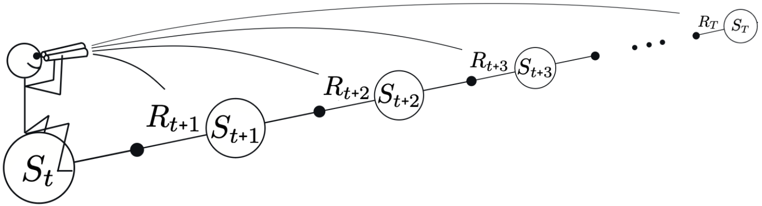
and then update the value of the current state S_t based on these future returns

the more far away this future is $\leftrightsquigarrow$ the less weight its return gets

In fact, we are assigning credit to our current state which will impact our future update: the more we go forward in time, the less this impact will be

TD$_\lambda$: *Forward View*

We can imagine this time advancement *as building impact towards future*[2]



But this approach is hard to be implemented online

we need to wait till the end of episode to compute the λ-returns!

[2] *This figure is taken from Sutton and Barto's book in Chapter 12*

TD$_\lambda$: *Backward View*

+ *But, it does not seem to be another way for credit assignment?*
– Well maybe we could apply the same idea backward
+ *How can we do it?*
– We can invoke the idea of eligibility tracing

Eligibility Tracing in Nutshell

At each time, when we update the value of a state in a sample trajectory, we also update the value of previous states we already met, with a weight decaying as we go back in time

Let's make an algorithm for that!

Eligibility Tracing

```
ElgTrace(S_t, E(·)):
1: Eligibility tracing function has N components, i.e., E(s) for all states
2: for n = 1 : N do
3:     Update E(s^n) ← γλE(s^n)  ⇜ choosing γλ for equivalency to forward view
4: end for
5: Update E(S_t) ← E(S_t) + 1
```

Say we initiate $E(s) = 0$ for all states and get to the following trajectory

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

Assume that we set $\gamma\lambda = 0.1$; then we have

1. *At $t = 0$, we only change $E(S_0) = 1$* ⇜ fresh memory ≡ high impact
2. *At $t = 1$, we change $E(S_0) = 0.1$ and $E(S_1) = 1$*

Eligibility Tracing

```
ElgTrace(S_t, E(·)):
1: Eligibility tracing function has N components, i.e., E(s) for all states
2: for n = 1 : N do
3:     Update E(s^n) ← γλE(s^n)   ⇜ [choosing γλ for equivalency to forward view]
4: end for
5: Update E(S_t) ← E(S_t) + 1
```

Say we initiate $E(s) = 0$ for all states and get to the following trajectory

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

Now say that we see at $t = 2$ the same state as in $t = 0$, i.e., $S_0 = S_2$

3. *At $t = 2$, we change $E(S_1) = 0.1$ and $E(S_0) \leftarrow 0.1E(S_0) + 1 = 1.01$*

Updating with Eligibility Tracing: *Intuition*

+ *What is the use of this algorithm?*
- We can simply *update all previous states each time t weighted by eligibility traces*

We can 0-bootstrap at each time, i.e., compute error as

$$\Delta_t = \underbrace{R_{t+1} + \gamma \hat{v}_\pi \left(S_{t+1}\right)}_{G_t^0} - \hat{v}_\pi \left(S_t\right)$$

and update any state $s = S_0, \ldots, S_t$ that has non-zero trace of eligibility as

$$\hat{v}_\pi \left(s\right) \leftarrow \hat{v}_\pi \left(s\right) + \alpha \Delta_t E_t \left(s\right)$$

with $E_t \left(s\right)$ denoting the eligibility trace that we have updated up to time t

TD_λ vs Eligibility Tracing

+ *Is there any concrete reason beside simple intuition that this is a good idea?*
– We can actually see that this is an online form of basic TD_λ

In fact, we can show by telescopic sum that

$$\sum_{t=0}^{T-1} \Delta_t E_t(s) = \sum_{t=0}^{T-1} \left(G_t^\lambda - \hat{v}_\pi(S_t) \right) \mathbf{1}\{S_t = s\}$$

This means that at the end of episode, we are updating the same!

- *If we set $\lambda = 0$ in the eligibility tracing*
  - ↳ *all eligibility trace remains 0 for all states: only we have $E(S_t) = 1$*
  - ↳ *we are back to TD-0 as it was with TD_0*
- *If we set $\lambda = 1$ in the eligibility tracing*
  - ↳ *we are back to Monte-Carlo approach as in TD_1*

TD$_\lambda$ with Eligibility Tracing: *Policy Evaluation*

We can evaluate policy by averaging its λ-returns over multiple episodes

`ElgTD_Eval`$_\lambda(\pi)$:

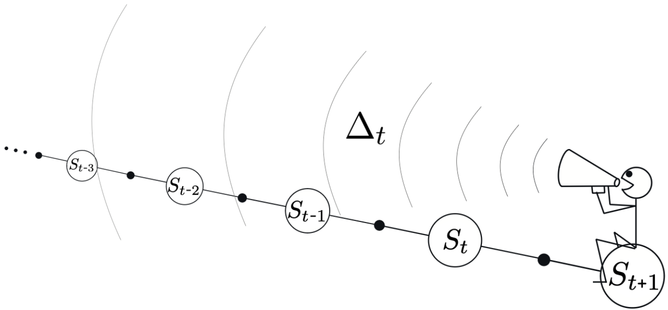
1: *Initiate value estimator and eligibility traces as $\hat{v}_\pi(s) = 0$ and $E(s) = 0$ for all s*
2: ***for** episode $= 1 : K$ **do***
3: *Initiate with a random state S_0 and act via policy $\pi(a|s)$*
4: *Sample a trajectory until either a terminal stated or some terminating T*

$$S_0, A_0 \xrightarrow{R_1} S_1, A_1 \xrightarrow{R_2} \cdots \xrightarrow{R_{T-1}} S_{T-1}, A_{T-1} \xrightarrow{R_T} S_T$$

5: ***for** $t = 0 : T-1$ **do***
6: $E(\cdot) \leftarrow$ `ElgTrace`$(S_t, E(\cdot))$
7: *Compute $G = R_{t+1} + \gamma \hat{v}_\pi(S_{t+1})$ and find error $\Delta = G - \hat{v}_\pi(S_t)$*
8: ***for** $n = 1 : N$ **do***
9: *Update $\hat{v}_\pi(s^n) \leftarrow \hat{v}_\pi + \alpha E(s^n) \Delta$*
10: ***end for***
11: ***end for***
12: ***end for***

Eligibility Tracing: *Backward View*

We can imagine eligibility tracing *as backward assignment of credits*[3]



Now we can update values each time and don't need to wait till end of episode!

[3] *This figure is taken from Sutton and Barto's book in Chapter 12*