DTU

02465: Introduction to reinforcement learning and control

Eligibility traces and value-function approximations

Tue Herlau

DTU Compute, Technical University of Denmark (DTU)

Lecture Schedule

Syllabus: `https://02465material.pages.compute.dtu.dk/02465public`
Help improve lecture by giving feedback on DTU learn

Reading material:

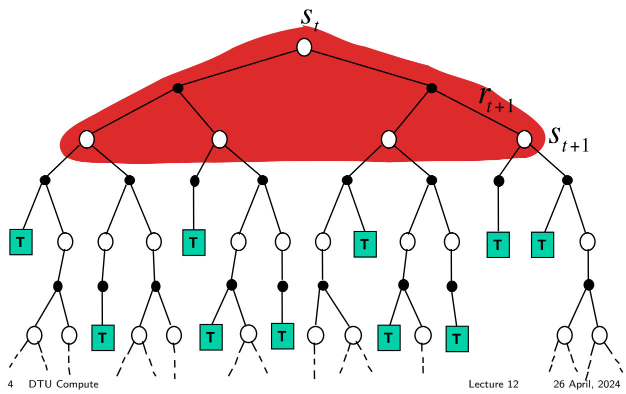
- [SB18, Chapter 10.2; 12-12.7]

Learning Objectives

- Using the TD-lambda return to interpolate between MC and TD(0)
- Eligibility traces as an efficient implementation of TD(lambda) and Sarsa(lambda)
- Function approximators and Sarsa(lambda)
- The online lambda-return, with emphasis on linear function approximators

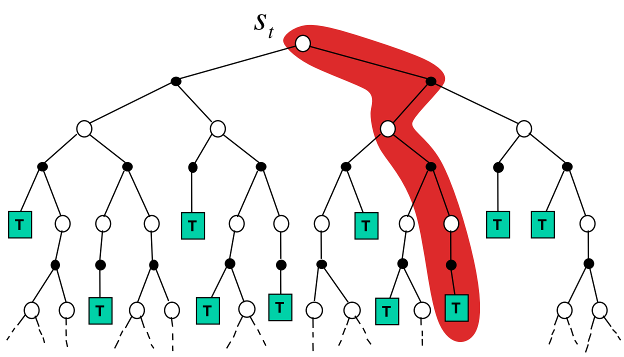
DP backups

$$V(S_t) \leftarrow \mathbb{E}_\pi \left[R_{t+1} + \gamma V(S_{t+1})\right]$$



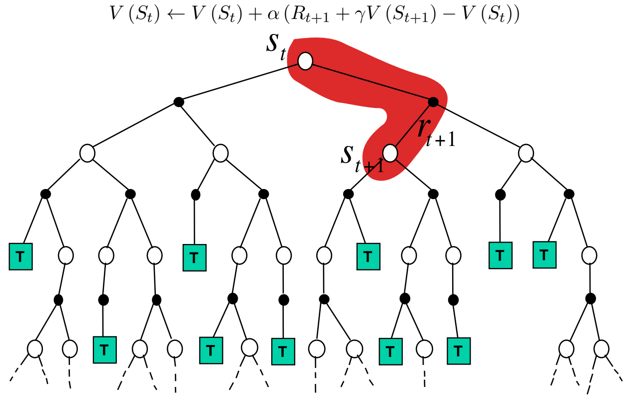
Last week: MC backups

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$

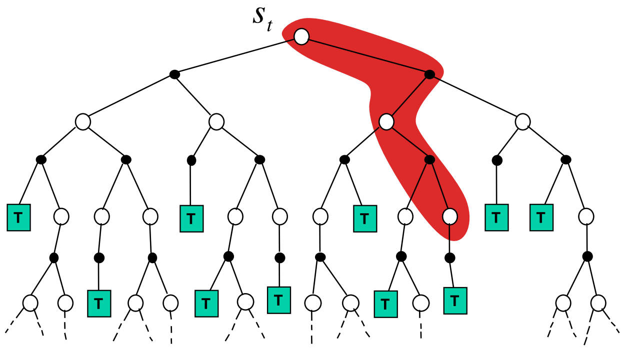


Last week: TD backups

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



Last week: n-step backup



General plan

- The λ-return provides a method to interpolate between TD(0) and Monte-Carlo
- There are **forward** and **backward** variant of λ-return methods
  - **Forward:** Quite easy to understand; annoying to implement
  - **Backward:** Harder to understand; it has the same updates of value-function but applied immediately. Much easier to implement.
- Additionally, [SB18] distinguishes between (i) regular $\mathrm{TD}(\lambda)$ and a more advanced variant (ii) online TD(λ)
  - ...and the online-version also has a forward and backward view...
  - ...and [SB18] presents the methods in context of function approximators...

We will focus on the tabular version.

From last week: The n-step return

- Recall return is $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots$

  $n=1$: **(TD)** $G_t^{(1)} = R_{t+1} + \gamma G_{t+1}$

  $n=2$: $G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 G_{t+2}$

  n: $G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n G_{t+n}$

  $n=\infty$ **(MC):** $G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-1} R_T$
- Using the rules of expectations:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n G_{t+n} | s] \\ &= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \mathbb{E}\left[\gamma^n G_{t+n} | S_{t+n}\right] | S_t = s\right] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n v_\pi(S_{t+n}) | S_t = s] \end{aligned}$$

  Therefore, the n-step return is an estimate of $V(S_t)$

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- This gives n-step temporal difference update:

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_{t:t+n} - V(S_t)\right)$$

Averaging n-step returns

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- We can average n-step returns for different n. The estimator

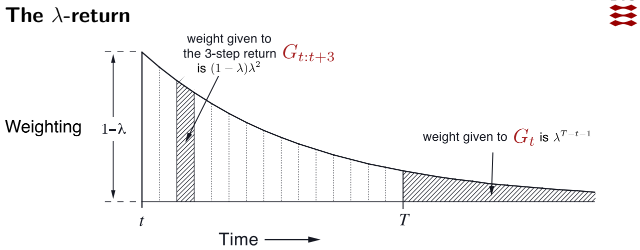
$$\bar{G}_t = \frac{1}{3} G_{t:t+2} + \frac{2}{3} G_{t:t+4}$$

  is still an estimator of the return
- More generally assuming that $\sum_{i=1}^{\infty} w_i = 1$ then

$$\bar{G}_t = \sum_{i=1}^{\infty} w_i G_{t:t+i}$$

  is an estimator of the return

The λ-return

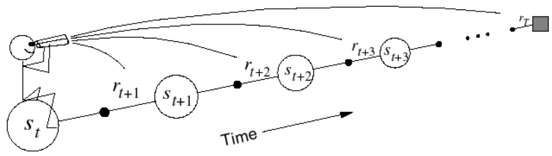


- Combine returns $G_{t:t+n}$ using weights $(1-\lambda)\lambda^{n-1}$ (note $\sum_{n=1}^{\infty}(1-\lambda)\lambda^{n-1} = 1$)

$$G_t^\lambda \doteq (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}$$

- For $t+n > T$ it is the case that $G_{t:t+n} = G_t$:

λ-return: $$G_t^\lambda = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t$$

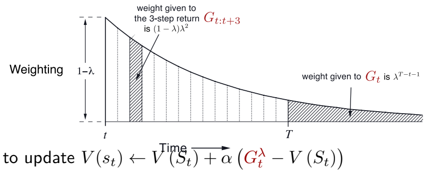


- Forward-view $\mathrm{TD}(\lambda)$ update rule is

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t^\lambda - V(S_t)\right)$$

- Forward-view $\mathrm{TD}(\lambda)$ looks into the future to compute G_t^λ
- Like MC, it can only be computed from complete episodes
- Theoretically simple, but computationally impractical

Backwards $\mathrm{TD}(\lambda)$



- We want to update $V(s_t) \leftarrow V(S_t) + \alpha\left(G_t^\lambda - V(S_t)\right)$

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t$$
$$= (1-\lambda)G_{t:t+1} + (1-\lambda)\lambda G_{t:t+2} + (1-\lambda)\lambda^2 G_{t:t+3} + \cdots + \lambda^{T-t-1} G_t$$

- The return G_t^λ includes the term $(1-\lambda)\lambda^2 G_{t:t+3}$
- This means $V(s_t)$ is updated towards

$$G_t^\lambda = \cdots + (1-\lambda)\lambda^2 (R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 V(S_{t+3})) + \cdots$$

- **Idea:** Wait until time $t+3$, compute above terms and update $V(s_t)$ **in the past**
- The further **in the future** a term R_{t+n} is, the less it influences **past** term $V(s_t)$

Eligibility trace

- The eligibility trace E_t is just af function of states: $E_t : \mathcal{S} \to \mathbb{R}$
- Measures both how frequent and how recent a state was visited
- Initialized to $E_{t=0}(s) = 0$
- Updated at each time step as

$$E_t(s) = \begin{cases} \gamma\lambda E_{t-1}(s) & \text{if } s \neq s_t \\ \gamma\lambda E_{t-1}(s) + 1 & \text{if } s = s_t \end{cases}$$

  - States decay at a rate of $\gamma\lambda$
  - Each time they are visited they get a bonus of $+1$,

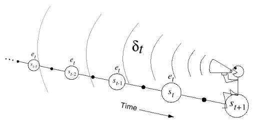
Backward view $\mathrm{TD}(\lambda)$

- Initialize value function for each state.
- At start of each episode, initialize eligibility trace for each state to $E(s) = 0$
- For each transition $S_t = s \to S_{t+1} = s'$, giving reward $R_{t+1} = r$, compute ordinary TD error

$$\delta_t = r + \gamma V(s') - V(s)$$

- Update eligibility trace

$$E_t(s) = E_t(s) + 1$$



- For every state s where $E_t(s) > 0$ update

$$V(s) \leftarrow V(s) + \alpha\delta E(s)$$
$$E(s) \leftarrow \gamma\lambda E(s)$$

- See `http://incompleteideas.net/book/ebook/node75.html`

$\lambda = 0$ is equivalent TD(0)

- When $\lambda = 0$ only the current state is updated:

$$E_t(s) = 1 \text{ if and only if } s = S_t$$
$$V(s) \leftarrow V(s) + \alpha\delta_t E_t(s)$$

- This means $\mathrm{TD}(\lambda)$ is equal to $\mathrm{TD}(0)$ when $\lambda = 0$

Equivalence of forward/Backward $\mathrm{TD}(\lambda)$

Suppose a state $S_t = s$ is visited just once at time step t

Forward-view The change in value-function $V(s)$ in the forward-view update is $\alpha(G_t^\lambda - V(S_t))$

Eligibility traces Implied update is:

- At t we change $E(S_t = s) = 1$
- In subsequent steps we iterate

$$V(s) \leftarrow V(s) + \alpha\delta E(s)$$
$$E(s) \leftarrow \gamma\lambda E(s)$$

- The last update means that at step $t+n$ we have $E(s) = (\gamma\lambda)^n$
- Total change to value function $V(s)$ is therefore

$$\alpha\left(\delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \ldots\right)$$

Are these two updates the same (is the red stuff equal)?

Proof:

Recall $G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$

$$G_t^\lambda - V(S_t) = -V(S_t) + (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}$$
$$= -V(S_t) + \left(\sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}\right) + \left(\sum_{n=1}^{\infty} -\lambda^n G_{t:t+n}\right)$$
$$= -V(S_t) + \left(G_{t:t+1} + \sum_{n=2}^{\infty} \lambda^{n-1} G_{t:t+n}\right) + \left(\sum_{n=2}^{\infty} -\lambda^{n-1} G_{t:t+n-1}\right)$$
$$= G_{t:t+1} - V(S_t) + \sum_{n=2}^{\infty} \lambda^{n-1}\left(G_{t:t+n} - G_{t:t+n-1}\right)$$

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

$$G_{t:t+n} - G_{t:t+n-1} = \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) - \gamma^{n-1} V(S_{t+n-1})$$
$$= \gamma^{n-1}\delta_{t+n-1}$$

Proof II

$$G_t^\lambda - V(S_t) = G_{t:t+1} - V(S_t) + \sum_{n=2}^{\infty} \lambda^{n-1} \left(G_{t:t+n} - G_{t:t+n-1}\right)$$
$$= \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right) + \sum_{n=2}^{\infty} \lambda^{n-1} \left(\gamma^{n-1}\delta_{t+n-1}\right)$$
$$= (\gamma\lambda)^0 \delta_t + \sum_{n=2}^{\infty} (\gamma\lambda)^{n-1} \delta_{t+n-1}$$
$$= (\gamma\lambda)^0 \delta_t + (\gamma\lambda)^1 \delta_{t+1} + (\gamma\lambda)^2 \delta_{t+2} + \cdots$$

Forward/Backward TD

Suppose a state $S_t = s$ is visited just once at time step t

Forward-view The change in value-function $V(s)$ in the forward-view update is $\alpha(G_t^\lambda - V(S_t))$

Eligibility traces Implied update is:

- At t we change $E(S_t = s) = 1$
- In subsequent steps we iterate

$$V(s) \leftarrow V(s) + \alpha\delta E(s)$$
$$E(s) \leftarrow \gamma\lambda E(s)$$

- The last update means that at step $t+n$ we have $E(s) = (\gamma\lambda)^n$
- Total change to value function $V(s)$ is therefore

$$\alpha\left(\delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \ldots\right)$$

Same updates!

Forward/Backward TD (Summary)

- Forward view is just using G_t^λ is an estimate of return
- Forward/Backwards TD are equivalent
  - Both change the value function the same way
  - Forward-view just changes value-function during an episode
- $\mathrm{TD}(\lambda = 0)$ is equivalent to $\mathrm{TD}(0)$
- $\mathrm{TD}(1)$ corresponds to MC

Control

From last week: n-step Sarsa

Recall the decomposition:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n G_{t+n}$$

- As before:

$$q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n G_{t+n} | S_t = s, A_t = a]$$
$$= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n q_\pi(S_{t+n}, A_{t+n}) | S_t = s, A_t = a]$$

- Therefore, the following n-step action-value return is an unbiased estimate of q_π

$$q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n q_\pi\left(S_{t+n}, A_{t+n}\right)$$

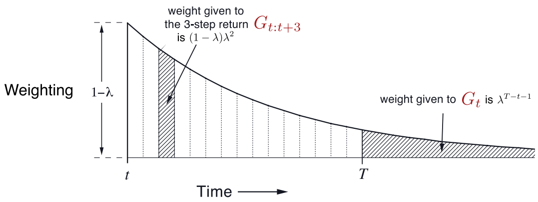
- Suggest the following bootstrap update of the action-value function

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha\left(q_t^{(n)} - Q\left(S_t, A_t\right)\right)$$

`lecture_12_sarsa_nstep_open.py`

Control

Forward-view Sarsa



- Use weights to combine returns $q_{t:t+n}$

$$q_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n Q\left(S_{t+n}, A_{t+n}\right)$$

- For $t + n \geq T$ it is the case $q_{t:t+n} = G_t$:

$$q_t^\lambda = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} q_{t:t+n} + \lambda^{T-t-1} G_t$$

- We therefore obtain the following generalized update rule

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha\left(q_t^\lambda - Q\left(S_t, A_t\right)\right)$$

Control

Backward view Sarsa(λ)

- We once more introduce an eligibility trace E_t, updated as before:

$$E_t(s,a) = \begin{cases} \gamma\lambda E_{t-1}(s,a) + 1 & \text{if } s = s_t \text{ and } a = a_t; \\ \gamma\lambda E_{t-1}(s,a) & \text{otherwise.} \end{cases} \quad \text{for all } s, a$$

- Each each step, given (s, a, r, s'), update

$$\delta_t = R_{t+1} + \gamma Q\left(S_{t+1}, A_{t+1}\right) - Q\left(S_t, A_t\right)$$
$$Q(s,a) \leftarrow Q(s,a) + \alpha\delta_t E_t(s,a)$$

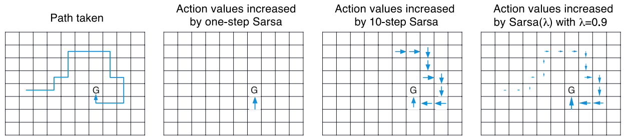
Sarsa(λ) control algorithm (tabular version)

See `http://incompleteideas.net/book/first/ebook/node77.html`

```
Initialize Q(s,a) arbitrarily, for all s ∈ S, a ∈ A(s)
Repeat (for each episode):
    E(s,a) = 0, for all s ∈ S, a ∈ A(s)
    Initialize S, A
    Repeat (for each step of episode):
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        δ ← R + γQ(S',A') − Q(S,A)
        E(S,A) ← E(S,A) + 1
        For all s ∈ S, a ∈ A(s):
            Q(s,a) ← Q(s,a) + αδE(s,a)
            E(s,a) ← γλE(s,a)
        S ← S'; A ← A'
    until S is terminal
```


Implied updates in the Open gridworld example

Recall only terminal state has a reward of $+1$



`lecture_12_sarsa_open.py`, `lecture_12_mc_open.py`, `lecture_12_sarsa_lambda_open.py`

From last time: Feature vectors and linear representations

- Represent value function by a linear combination of features

$$\hat{v}(s, \mathbf{w}) = \mathbf{x}(s)^\top \mathbf{w}, \quad \boldsymbol{w} \in \mathbb{R}^d$$

Where **feature vector** is defined as:

$$\mathbf{x}(s) = \begin{bmatrix} \mathbf{x}_1(s) \\ \vdots \\ \mathbf{x}_d(s) \end{bmatrix}$$

- The gradient is simply:

$$\nabla \hat{v}(s, \mathbf{w}) = \mathbf{x}(s)$$

In this case $\hat{q}(s, a, \boldsymbol{w}) = \boldsymbol{x}(s, a)^\top \boldsymbol{w}$

From last time: implementation details

- TD learning

$$V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s))$$
$$\boldsymbol{w} \leftarrow \boldsymbol{w} \quad + \alpha\left(r + \gamma \hat{v}(s', \boldsymbol{w}) - \hat{v}(s, \boldsymbol{w})\right) \nabla \hat{v}(s, \boldsymbol{w})$$

- Sarsa learning

$$q(s,a) \leftarrow q(s,a) + \alpha\left(r + \gamma q(s', a') - q(s,a)\right)$$
$$\boldsymbol{w} \leftarrow \boldsymbol{w} \quad + \alpha\left(r + \gamma \hat{q}(s', a', \boldsymbol{w}) - \hat{q}(s, a, \boldsymbol{w})\right) \nabla \hat{q}(s, a, \boldsymbol{w})$$

- Using a general estimator:

$$q(s,a) \leftarrow q(s,a) + \alpha\left(G - q(s,a)\right)$$
$$\boldsymbol{w} \leftarrow \boldsymbol{w} \quad + \alpha\left(G - \hat{q}(s, a, \boldsymbol{w})\right) \nabla \hat{q}(s, a, \boldsymbol{w})$$

Forward and backward view

Assuming linear function approximators: $\nabla \hat{q}(s, a, \boldsymbol{w}) = \boldsymbol{x}(s, a)$

- Forward view Sarsa(λ) is exactly as before

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\left(G_t^\lambda - \hat{q}(s, a, \boldsymbol{w})\right) \nabla \hat{q}(s, a, \boldsymbol{w})$$

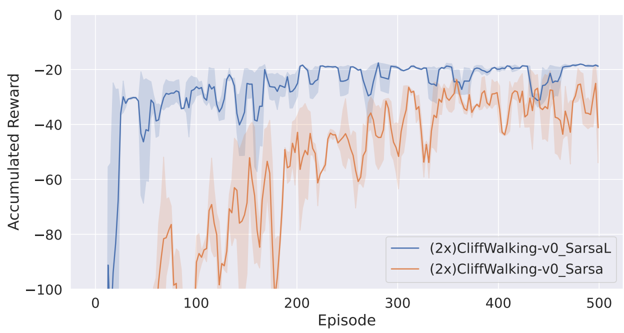
- Keep track of terms that include which gradient to get **backward view** of Sarsa(λ):

$$\delta_t = R_{t+1} + \gamma \hat{q}\left(S_{t+1}, A_{t+1}, \mathbf{w}_t\right) - \hat{q}\left(S_t, A_t, \mathbf{w}_t\right)$$
$$\boldsymbol{z}_t = \gamma \lambda \boldsymbol{z}_{t-1} + \nabla \hat{q}(S_t, A_t, \boldsymbol{w}_t)$$
$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \alpha \delta_t \boldsymbol{z}_t$$

- The gradient plays the role of state-action pairs visited. It is propagated into the future but attenuated by $\gamma\lambda$
- A change in the past (gradient) which lead to a **poor** (or good) result δ_t will be **penalized** (promoted)
- Forward/backward view equivalent in the linear case

Cliffwalk example

Comparison of Sarsa(λ) and Sarsa on the cliffwalk example



(Note that results are somewhat sensitive to the to learning rate)

Quiz: Exam problem spring 2023

Which one of the following questions are correct?

a. $\mathrm{TD}(\lambda)$ cannot be used with function approximators

b. The role of the eligibility trace is to let reward obtained earlier in an episode affect the change in the value function later in the episode

c. The eligibility trace cannot be negative

d. The eligibility trace is a measure of the amount of reward obtained in a given state weighted by an exponential factor

e. Don't know.

Using binary features

Sarsa(λ) with binary features and linear function approximation for estimating $\mathbf{w}^\top \mathbf{x} \approx q_\pi$ or q_*

Input: a function $\mathcal{F}(s, a)$ returning the set of (indices of) active features for s, a
Input: a policy π (if estimating q_π)
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0, 1]$
Initialize: $\mathbf{w} = (w_1, \dots, w_d)^\top \in \mathbb{R}^d$ (e.g., $\mathbf{w} = \mathbf{0}$), $\mathbf{z} = (z_1, \dots, z_d)^\top \in \mathbb{R}^d$

Loop for each episode:
    Initialize S
    Choose $A \sim \pi(\cdot|S)$ or ε-greedy according to $\hat{q}(S, \cdot, \mathbf{w})$
    $\mathbf{z} \leftarrow \mathbf{0}$
    Loop for each step of episode:
        Take action A, observe R, S'
        ~~$\delta \leftarrow R$~~ $\delta \leftarrow R - \boldsymbol{w}^\top \boldsymbol{x}$
        ~~Loop for i in $\mathcal{F}(S, A)$:~~
            ~~$\delta \leftarrow \delta - w_i$~~
            ~~$z_i \leftarrow z_i + 1$~~ $z \leftarrow z + x$ (accumulating traces)
            ~~or $z_i \leftarrow 1$~~ ~~(replacing traces)~~
        If S' is terminal then:
            $\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \mathbf{z}$
            Go to next episode
        Choose $A' \sim \pi(\cdot|S')$ or near greedily $\sim \hat{q}(S', \cdot, \mathbf{w})$
        ~~Loop for i in $\mathcal{F}(S', A')$: $\delta \leftarrow \delta + \gamma w_i$~~ $\delta \leftarrow \delta + \gamma \boldsymbol{w}^\top \boldsymbol{x}'$
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \mathbf{z}$
        $\mathbf{z} \leftarrow \gamma \lambda \mathbf{z}$
        $S \leftarrow S'; A \leftarrow A'$

Truncated, online and true online λ-return algorithms (advanced)

- Recall the λ-return is defined as:

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{T-t-1} G_t$$

- Each G_t is an estimate of the return and the sum of the weights is 1
- More generally the **truncated λ-return** estimator is

$$G_{t:h}^\lambda = (1 - \lambda) \sum_{n=1}^{h-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{h-t-1} G_{t:h}, \quad 0 \le t < h \le T$$

Using the estimator

- Recall the forward-view $\mathrm{TD}(\lambda)$ algorithm:

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^\lambda - V(S_t))$$

- The **truncated λ** return fixes $h = n$ and do:

$$V(S_t) \leftarrow V(S_t) + \alpha(G_{t:t+n}^\lambda - V(S_t))$$

- Or as weight updates

$$\boldsymbol{w}_{t+n} = \boldsymbol{w}_{t+n-1} + \alpha \left(G_{t:t+n}^\lambda - \hat{v}(S_t, \boldsymbol{w}_{t+n-1})\right) \nabla \hat{v}(S_t, \boldsymbol{w}_{t+n-1})$$

- This requires a fixed n and that we store previous results. Can we do better?

Online λ-return

$$G_{t:h}^\lambda = (1 - \lambda) \sum_{n=1}^{h-t-1} \lambda^{n-1} G_{t:t+n} + \lambda^{h-t-1} G_{t:h}, \quad 0 \le t < h \le T$$

- Once we have observed h steps of an episode, we can evaluate

$$G_{0:h}^\lambda, G_{1:h}^\lambda, \dots, G_{h-1:h}^\lambda$$

- **Online λ-return**: After h steps, perform h updates corresponding to all h returns
- Repeat each time h is increased

$$\begin{aligned}
h = 1: \quad & \mathbf{w}_1^1 \doteq \mathbf{w}_0^1 + \alpha \left[G_{0:1}^\lambda - \hat{v}(S_0, \mathbf{w}_0^1)\right] \nabla \hat{v}(S_0, \mathbf{w}_0^1), \\
h = 2: \quad & \mathbf{w}_1^2 \doteq \mathbf{w}_0^2 + \alpha \left[G_{0:2}^\lambda - \hat{v}(S_0, \mathbf{w}_0^2)\right] \nabla \hat{v}(S_0, \mathbf{w}_0^2), \\
& \mathbf{w}_2^2 \doteq \mathbf{w}_1^2 + \alpha \left[G_{1:2}^\lambda - \hat{v}(S_1, \mathbf{w}_1^2)\right] \nabla \hat{v}(S_1, \mathbf{w}_1^2), \\
h = 3: \quad & \mathbf{w}_1^3 \doteq \mathbf{w}_0^3 + \alpha \left[G_{0:3}^\lambda - \hat{v}(S_0, \mathbf{w}_0^3)\right] \nabla \hat{v}(S_0, \mathbf{w}_0^3), \\
& \mathbf{w}_2^3 \doteq \mathbf{w}_1^3 + \alpha \left[G_{1:3}^\lambda - \hat{v}(S_1, \mathbf{w}_1^3)\right] \nabla \hat{v}(S_1, \mathbf{w}_1^3), \\
& \mathbf{w}_3^3 \doteq \mathbf{w}_2^3 + \alpha \left[G_{2:3}^\lambda - \hat{v}(S_2, \mathbf{w}_2^3)\right] \nabla \hat{v}(S_2, \mathbf{w}_2^3).
\end{aligned}$$

- I.e. for each new step $h - 1 \to h$ repeat $t = 0, \dots, h - 1$:

$$\boldsymbol{w}_{t+1}^h = \boldsymbol{w}_t^h + \alpha \left[G_{t:h}^\lambda - \hat{v}(S_t, \boldsymbol{w}_t^h)\right] \nabla \hat{v}(S_t, \boldsymbol{w}_t^h)$$

True online $\mathrm{TD}(\lambda)$

- Online $\mathrm{TD}(\lambda)$ is computationally very wasteful
- For linear function approximators online $\mathrm{TD}(\lambda)$ allows a backwards view known as **True online $\mathrm{TD}(\lambda)$**

$$\begin{aligned}
\boldsymbol{w}_{t+1} &= \boldsymbol{w}_t + \alpha \delta_t \boldsymbol{z}_t + \alpha(\boldsymbol{w}_t^\top \boldsymbol{x}_t - \boldsymbol{w}_{t-1}^\top \boldsymbol{x}_t)(\boldsymbol{z}_t - \boldsymbol{x}_t) \\
\boldsymbol{z}_t &= \gamma \lambda \boldsymbol{z}_{t-1} + (1 - \alpha \gamma \lambda \boldsymbol{z}_{t-1}^\top \boldsymbol{x}_t) \boldsymbol{x}_t
\end{aligned}$$

- The control algorithm is **true online Sarsa(λ)**

True online Sarsa(λ)

True online Sarsa(λ) for estimating $\mathbf{w}^\top \mathbf{x} \approx q_\pi$ or q_*

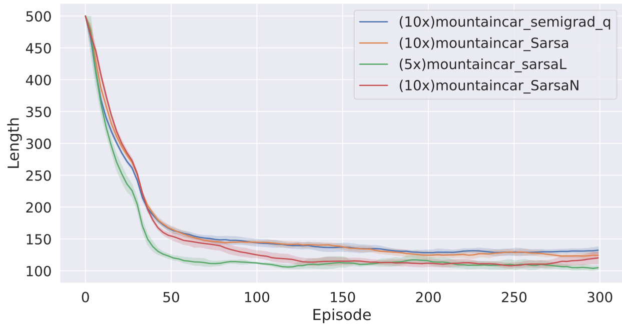
Input: a feature function $\mathbf{x} : \mathcal{S}^+ \times \mathcal{A} \to \mathbb{R}^d$ such that $\mathbf{x}(terminal, \cdot) = \mathbf{0}$
Input: a policy π (if estimating q_π)
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0, 1]$
Initialize: $\mathbf{w} \in \mathbb{R}^d$ (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  Initialize S
  Choose $A \sim \pi(\cdot|S)$ or near greedily from S using $\mathbf{w}$
  $\mathbf{x} \leftarrow \mathbf{x}(S, A)$
  $\mathbf{z} \leftarrow \mathbf{0}$
  $Q_{old} \leftarrow 0$
  Loop for each step of episode:
  | Take action A, observe R, S'
  | Choose $A' \sim \pi(\cdot|S')$ or near greedily from S' using $\mathbf{w}$
  | $\mathbf{x}' \leftarrow \mathbf{x}(S', A')$
  | $Q \leftarrow \mathbf{w}^\top \mathbf{x}$
  | $Q' \leftarrow \mathbf{w}^\top \mathbf{x}'$
  | $\delta \leftarrow R + \gamma Q' - Q$
  | $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \left(1 - \alpha\gamma\lambda\mathbf{z}^\top\mathbf{x}\right)\mathbf{x}$
  | $\mathbf{w} \leftarrow \mathbf{w} + \alpha(\delta + Q - Q_{old})\mathbf{z} - \alpha(Q - Q_{old})\mathbf{x}$
  | $Q_{old} \leftarrow Q'$
  | $\mathbf{x} \leftarrow \mathbf{x}'$
  | $A \leftarrow A'$
  until S' is terminal

(we will implement this during the exercises)

Mountaincar example

Comparison of $\mathrm{Sarsa}(\lambda)$ and Sarsa on the Mountaincar example

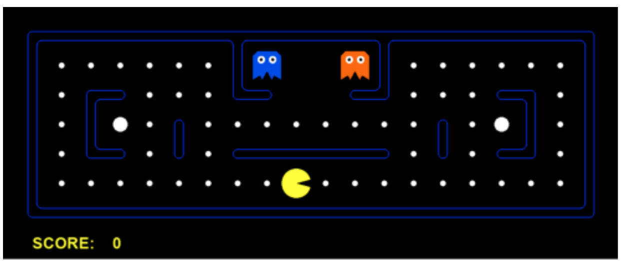


Richard S. Sutton and Andrew G. Barto.
Reinforcement Learning: An Introduction.
The MIT Press, second edition, 2018.
(Freely available online).

Appendix

A more challenging pacman environment

- Use successor representation: $\hat{q}(s, a, \boldsymbol{w}) = \boldsymbol{x}(s')^\top \boldsymbol{w}$, $s' = f(s, a)$



A more challenging pacman environment