Lecture 9: Eligibility Traces

Friday, January 14, 2022

Reinforcement Learning, Winter Term 2021/22

Joschka Boedecker, Gabriel Kalweit, Jasper Hoffmann

Neurorobotics Lab
University of Freiburg

Lecture Overview

Lecture Overview

Recap: Off-policy Methods with Function Approximation

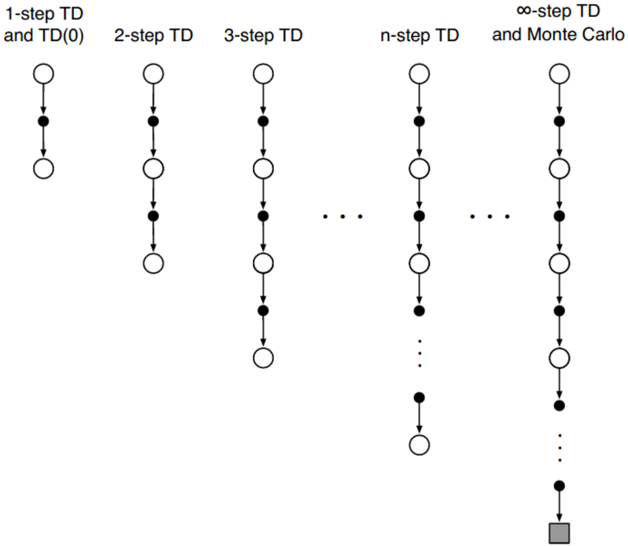
- *Deadly Triad*: The combination of Function Approximation, Bootstrapping and Off-policy Learning
- Gradient TD methods: TD(0) with gradient correction (TDC)
- NFQ (experience replay, full batch)
- DQN (experience replay, minibatches, target networks)

Lecture Overview

n-step Bootstrapping

- Recall:
  - MC-target: $R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_T$
  - TD-target: $R_{t+1} + \gamma V(S_{t+1})$
- n-step Return:
  $G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$
- $n = 0$: TD
- $n = \infty$: MC

n-step TD Prediction



n-step TD Prediction

n-step TD for estimating $V \approx v_\pi$

Input: a policy π
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for S_t and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take an action according to $\pi(\cdot|S_t)$
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then $T \leftarrow t+1$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose state's estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n V(S_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $V(S_\tau) \leftarrow V(S_\tau) + \alpha \left[G - V(S_\tau)\right]$
  Until $\tau = T - 1$

n-step Sarsa

How can n-step methods be used not just for prediction but for control?

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n Q_{t+n-1}(S_{t+n}, A_{t+n})$$

where $n \geq 1$ and $0 \leq t < T - n$ and $G_{t:t+n} = G_t$ if $t + n \geq T$.

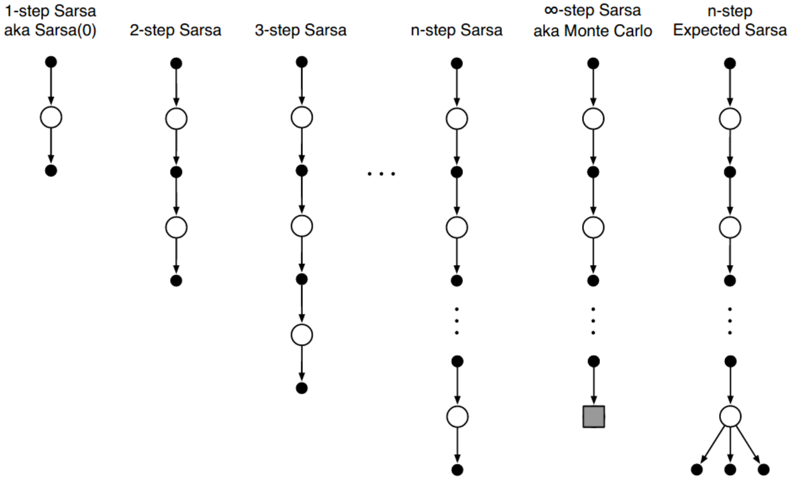
n-step Sarsa

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)],\ 0 \leq t < T$$

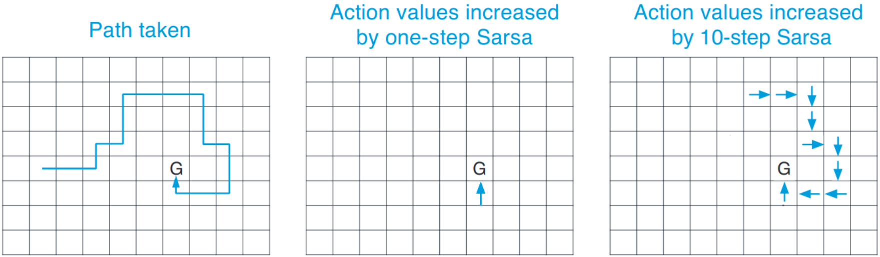
while the values of all other states remain unchanged:

$$Q_{t+n}(s, a) = Q_{t+n-1}(s, a) \quad \forall s, a \text{ s.t. } s \neq S_t \text{ or } a \neq A_t$$

n-step Sarsa


1-step Sarsa
aka Sarsa(0)
2-step Sarsa
3-step Sarsa
n-step Sarsa
∞-step Sarsa
aka Monte Carlo
n-step
Expected Sarsa

Example: Gridworld



n-step Off-policy Learning

Importance sampling ratio (the relative probability under the two policies of taking the n actions from A_t to A_{t+n-1}):

$$\rho_{t:h} = \prod_{k=t}^{\min(h,T-1)} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

Previous n-step Sarsa update can be replaced by an off-policy form:

$$Q_{t+n}(S_t, A_t) = Q_{t+n-1}(S_t, A_t) + \alpha\rho_{t+1:t+n}[G_{t:t+n} - Q_{t+n-1}(S_t, A_t)]$$

with $0 \leq t < T$.

Off-policy n-step Sarsa

Off-policy n-step Sarsa for estimating $Q \approx q_*$ or q_π

Input: an arbitrary behavior policy b such that $b(a|s) > 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be greedy with respect to Q, or as a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim b(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take action A_t
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then:
  | $T \leftarrow t+1$
  | else:
  | Select and store an action $A_{t+1} \sim b(\cdot|S_{t+1})$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | $\rho \leftarrow \prod_{i=\tau+1}^{\min(\tau+n-1,T-1)} \frac{\pi(A_i|S_i)}{b(A_i|S_i)}$ $(\rho_{\tau+1:t+n-1})$
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then: $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\rho\,[G - Q(S_\tau, A_\tau)]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is greedy wrt Q
  Until $\tau = T - 1$

Averaging n-step Returns

- We can average n-step returns over different n
- e.g. average the 2-step and 4-step returns

$$\frac{1}{2}G_{t:t+2} + \frac{1}{2}G_{t:t+4}$$

- Combines information from two different time steps
- Can we efficiently combine information from all time-steps?

Lecture Overview

Eligibility Traces and λ-return

Eligibility traces unify and generalize TD and Monte Carlo methods

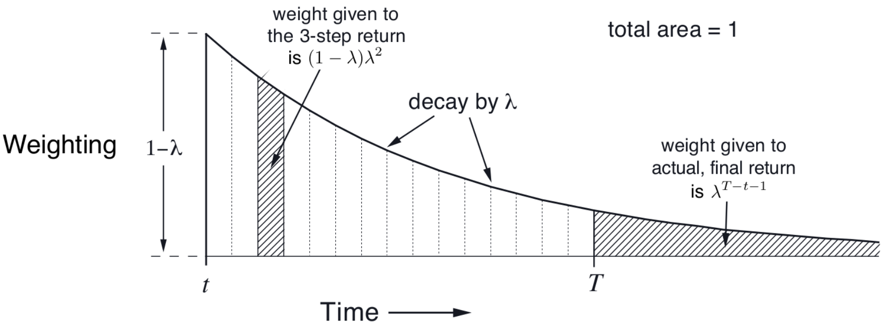
- MC methods at one end ($\lambda = 1$) and one-step TD methods at the other ($\lambda = 0$)
- almost any temporal-difference (TD) method can be combined with eligibility traces to (maybe) learn more efficiently

λ-return

- For infinite control tasks: $G_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}$
- For episodic control tasks:

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

λ-return



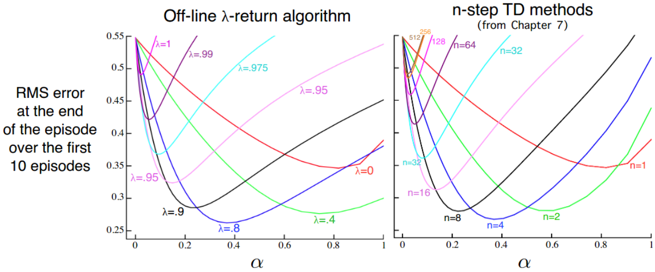
At the end of the episode, updates are made according to:

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

or, in the FA setting, to the semi-gradient rule:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[G_t^\lambda - \hat{v}(S_t, \mathbf{w}_t)]\nabla\hat{v}(S_t, \mathbf{w}_t), t = 0, ..., T-1$$

Alternative way of moving smoothly between MC and one-step TD, can be compared to n-step TD methods.



Offline λ-Return Algorithm

The Offline λ-update can be converted to TD-form:

$$V(S_t) \leftarrow V(S_t) + \alpha \sum_{i=0}^{\infty} \lambda^i \delta_{t+i+1}$$

Offline λ-Return Algorithm

Lemma 9.1

$$(1-\lambda)\sum_{n=m}^{\infty}\lambda^{n-1} = \lambda^{m-1}$$

Proof.

$$\begin{aligned}
(1-\lambda)\sum_{n=m}^{\infty}\lambda^{n-1} &= (1-\lambda)\left(\sum_{n=1}^{\infty}\lambda^{n-1} - \sum_{k=1}^{m-1}\lambda^{k-1}\right)\\
&= (1-\lambda)\left(\sum_{n=0}^{\infty}\lambda^{n} - \sum_{k=0}^{m-2}\lambda^{k}\right)\\
&= (1-\lambda)\left(\frac{1}{1-\lambda} - \frac{1-\lambda^{m-1}}{1-\lambda}\right)\\
&= 1-(1-\lambda^{m-1})\\
&= \lambda^{m-1}
\end{aligned}$$

□

Offline λ-Return Algorithm

Lemma 9.2

$$\sum_{n=1}^{\infty} \lambda^{n-1} \sum_{m=1}^{n} R_{t+m} = \sum_{m=1}^{\infty} R_{t+m} \sum_{n=m}^{\infty} \lambda^{n-1}$$

Proof.

$$\begin{aligned}
\sum_{n=1}^{\infty} \lambda^{n-1} \sum_{m=1}^{n} R_{t+m} &= \lambda^0 R_{t+1} + \lambda^1 (R_{t+1} + R_{t+2}) \\
&\quad + \lambda^2 (R_{t+1} + R_{t+2} + R_{t+3}) + \cdots \\
&= R_{t+1}(\lambda^0 + \lambda^1 + \lambda^2 + \cdots) + R_{t+2}(\lambda^1 + \lambda^2 + \cdots) \\
&\quad + R_{t+3}(\lambda^2 + \cdots) + \cdots \\
&= \sum_{m=1}^{\infty} R_{t+m} \sum_{n=m}^{\infty} \lambda^{n-1}
\end{aligned}$$

□

Offline λ-Return Algorithm

Lemma 9.3

$$(1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\sum_{m=1}^{n}R_{t+m} = \sum_{m=1}^{\infty}\lambda^{m-1}R_{t+m}$$

Proof.

$$\begin{aligned}
(1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\sum_{m=1}^{n}R_{t+m} &\overset{2}{=} (1-\lambda)\sum_{m=1}^{\infty}R_{t+m}\sum_{n=m}^{\infty}\lambda^{n-1} \\
&= \sum_{m=1}^{\infty}R_{t+m}(1-\lambda)\sum_{n=m}^{\infty}\lambda^{n-1} \\
&\overset{1}{=} \sum_{m=1}^{\infty}\lambda^{m-1}R_{t+m}
\end{aligned}$$

$\square$

Offline λ-Return Algorithm

Lemma 9.4

$$(1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}V(S_{t+n}) = V(S_t) + \sum_{m=1}^{\infty}\lambda^{m-1}[V(S_{t+m}) - V(S_{t+m-1})]$$

Proof.

$$\begin{aligned}
(1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}V(S_{t+n}) &= \sum_{n=1}^{\infty}V(S_{t+n})(\lambda^{n-1}-\lambda^{n}) \\
&= V(S_{t+1})(\lambda^0-\lambda^1) + V(S_{t+2})(\lambda^1-\lambda^2) + \cdots \\
&= \lambda^0 V(S_{t+1}) + \lambda^1 V(S_{t+2}) - \lambda^1 V(S_{t+1}) + \\
&\qquad \lambda^2 V(S_{t+3}) - \lambda^2 V(S_{t+2}) + \cdots + \\
&\qquad \lambda^0 V(S_t) - \lambda^0 V(S_t) \\
&= V(S_t) + \sum_{m=1}^{\infty}\lambda^{m-1}[V(S_{t+m}) - V(S_{t+m-1})]
\end{aligned}$$

Offline λ-Return Algorithm

The Offline λ-update can be converted to TD-form:

$$V(S_t) \leftarrow V(S_t) + \alpha \sum_{i=0}^{\infty} \lambda^i \delta_{t+i+1}$$

Proof. For simplification, we will ignore the discount.

$$\begin{aligned}
V(S_t) &= (1-\lambda) \cdot \mathbb{E}\left[\sum_{n=1}^{\infty} \lambda^{n-1} G_{t:t+n}\right] \\
&= (1-\lambda) \cdot \mathbb{E}\left[\sum_{n=1}^{\infty} \lambda^{n-1} \left(\sum_{m=1}^{n} R_{t+m} + V(S_{t+n})\right)\right] \\
&\overset{3,4}{=} \mathbb{E}\left[\sum_{m=1}^{\infty} \lambda^{m-1} [R_{t+m} + V(S_{t+m}) - V(S_{t+m-1})] + V(S_t)\right] \\
&= \mathbb{E}\left[\sum_{m=1}^{\infty} \lambda^{m-1} \delta_{t+m} + V(S_t)\right]
\end{aligned}$$

Offline λ-Return Algorithm

The Offline λ-update can be converted to TD-form:

$$V(S_t) \leftarrow V(S_t) + \alpha \sum_{i=0}^{\infty} \lambda^i \delta_{t+i+1}$$

Proof. For simplification, we will ignore the discount.
If we plug this into our value function update, we get:

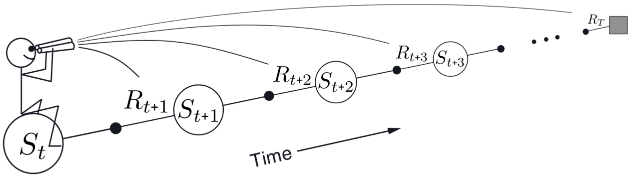
$$V(S_t) \leftarrow V(S_t) + \alpha \left(\sum_{m=1}^{\infty} \lambda^{m-1} \delta_{t+m} + V(S_t) - V(S_t) \right),$$

which leads to:

$$V(S_t) \leftarrow V(S_t) + \alpha \sum_{m=0}^{\infty} \lambda^m \delta_{t+m+1}.$$

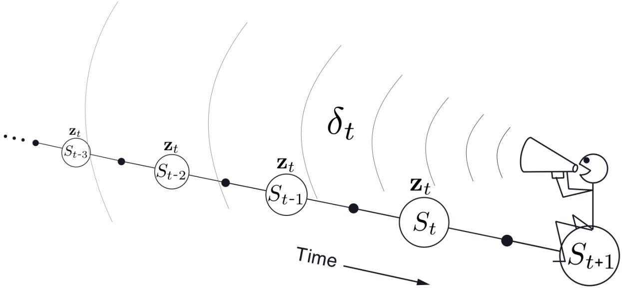
$\square$

Forward View



- Update value function towards the λ-return
- Forward-view looks into the future to compute G_t^λ
- Like MC, can only be computed from complete episodes

Backward View



- look at the current TD error δ_t
- assign it backward to each prior state according to how much that state contributed to the current eligibility trace at that time

Eligibility Traces

- Eligibility Traces assign credit to components of the weight vector according to their contribution to state valuations
- They combine heuristics of *Frequency* and *Recency* (implemented by a λ-decay)
- With function approximation, the eligibility trace is a vector $\mathbf{z}_t \in \mathbb{R}$, initialized by $\mathbf{z}_{-1} = \mathbf{0}$ and incremented on each time step by:

$$\mathbf{z}_t = \gamma\lambda\mathbf{z}_{t-1} + \nabla\hat{v}(S_t, \mathbf{w}_t), \quad 0 \leq t \leq T,$$

where λ is called trace-decay parameter.

TD(λ)

- TD(λ) updates the weight vector on every step of an episode rather than only at the end and is not limited to episodic problems
- With the TD error

$$\delta_t = R_{t+1} + \gamma \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t),$$

in TD(λ), the weight vector is updated by:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t.$$

TD(λ)

Semi-gradient TD(λ) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \rightarrow \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameters: step size $\alpha > 0$, trace decay rate $\lambda \in [0, 1]$
Initialize value-function weights $\mathbf{w}$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  Initialize S
  $\mathbf{z} \leftarrow \mathbf{0}$ (a d-dimensional vector)
  Loop for each step of episode:
  | Choose $A \sim \pi(\cdot|S)$
  | Take action A, observe R, S'
  | $\mathbf{z} \leftarrow \gamma\lambda\mathbf{z} + \nabla\hat{v}(S,\mathbf{w})$
  | $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$
  | $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\mathbf{z}$
  | $S \leftarrow S'$
  until S' is terminal

n-step Truncated λ-return

- offline λ-return algorithm is limited because:
  - the λ-return is not known until the end of the episode, or
  - technically never known in the continuing case
- dependence becomes weaker for longer-delayed reward, falling by $\gamma\lambda$ for each step of delay
- natural approximation: truncate the sequence after some number of steps:

truncated λ-return

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

n-step Truncated λ-return

- Now: updates are delayed by n steps and only take into account the first n rewards, but now all the k-step returns are included for $1 \leq k \leq n$.
- State-value case: truncated TD(λ) or TTD(λ)

TTD(λ)

$$\mathbf{w}_{t+n} = \mathbf{w}_{t+n-1} + \alpha[G^{\lambda}_{t:t+n} - \hat{v}(S_t, \mathbf{w}_{t+n-1})]\nabla\hat{v}(S_t, \mathbf{w}_{t+n-1}), 0 \leq t < T.$$

The k-step λ-return can efficiently implemented as:

$$G_{t:t+k} = \hat{v}(S_t, \mathbf{w}_{t-1}) + \sum_{i=t}^{t+k-1} (\gamma\lambda)^{i-t}\delta'_i$$

with $\delta'_i = R_{t+1} + \gamma\hat{v}(S_{t+1}, \mathbf{w}_{t-1}) - \hat{v}(S_t, \mathbf{w}_{t-1})$.

Sarsa(λ)

Extension of eligibility traces to action-value methods.

- Action-value form of the n-step return:

  $$G_{t:t+n} = R_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n \hat{q}(S_{t+n}, A_{t+n}, \mathbf{w}_{t+n-1}), \;\; t+n < T$$

  with $G_{t:t+n} = G_t$ if $t + n \geq T$.
- Action-value form of the truncated λ-return:

  $$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[G_t^\lambda - \hat{q}(S_t, A_t, \mathbf{w}_t)] \nabla \hat{q}(S_t, A_t, \mathbf{w}_t), \quad t = 0, ..., T-1,$$

  where $G_t^\lambda = G_{t:\infty}^\lambda$.

Sarsa(λ)

- With the TD error

$$\delta_t = R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) - \hat{q}(S_t, A_t, \mathbf{w}_t),$$

- And the action-value form of the TD error:

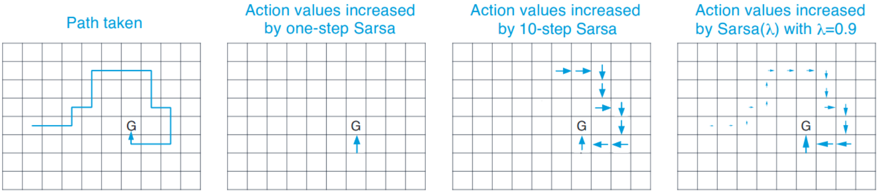
$$\mathbf{z}_{-1} = \mathbf{0}$$

$$\mathbf{z}_t = \gamma\lambda\mathbf{z}_{t-1} + \nabla\hat{q}(S_t, A_t, \mathbf{w}_t), \quad 0 \leq t \leq T,$$

in Sarsa(λ), the weight vector is updated by:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t.$$

Example: Traces in Gridworld



Lecture Overview

Summary by Learning Goals

Having heard this lecture, you can now...

- explain two different ways of shifting and choosing between Monte Carlo and TD methods
- why eligibility trace methods are more general and often faster to learn