Reinforcement Learning
Lecture 8: nstep bootstrapping and eligibility traces

Lecturer: Prof. Dr. Mathias Niepert

Institute for Artificial Intelligence
Machine Learning and Simulation Lab

June 20, 2024

Outline

n-step bootstrapping

- Unifying Monte Carlo and TD
- n-step TD
- n-step Sarsa

Unifying Monte Carlo and TD

Monte Carlo Prediction (Estimation of v_π)

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

$V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow G + R_{t+1}$

Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:

Append G to $Returns(S_t)$

$V(S_t) \leftarrow \text{average}(Returns(S_t))$

Backup diagram

- Entire episode included
- Only single choice considered at each state
- Thus, there will be an explore/exploit dilemma
- Value is estimated by mean return

Policy evaluation (prediction)

- We have some policy π which tells the agent which action a to choose in state s
- Find the value function $v_\pi(s)$ of this policy, i.e. **evaluate** the policy π

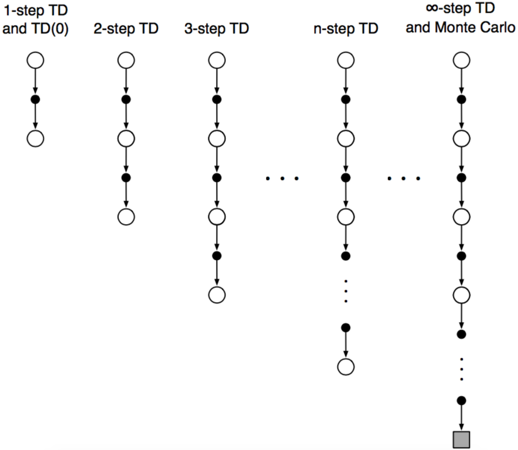
$$V(S_t) = V(S_t) + \alpha\Big[\underbrace{G_t}_{\text{MC target}} - V(S_t)\Big]$$

- Simplest temporal difference update TD(0):

$$V(S_t) = V(S_t) + \alpha\Big[\underbrace{\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{TD target}} - V(S_t)}_{\text{TD error}}\Big]$$

- TD error is error in the estimate made at a particular time step
- Reinforcement:
  - more reward than expected: $R_{t+1} + \gamma V(S_{t+1}) > V(S_t) \Rightarrow V(S_t) \uparrow$
  - less reward than expected: $R_{t+1} + \gamma V(S_{t+1}) < V(S_t) \Rightarrow V(S_t) \downarrow$

n-step TD prediction



n-step returns

- Monte Carlo:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{T-t-1} R_T$$

- TD:

$$G_{t:t+1} = R_{t+1} + \gamma V_t(S_{t+1})$$

- 2-step return:

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

- n-step return:

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

Error-reduction property

- *Error reduction property* of n-step returns

$$\underbrace{\max_s \left|\mathbb{E}_\pi[G_{t:t+n} \mid S_t = s] - v_\pi(s)\right|}_{\text{Maximum error using } n\text{-step return}} \leq \underbrace{\gamma^n \max_s \left|V_{t+n-1}(s) - v_\pi(s)\right|}_{\text{Maximum error using } V}$$

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

- Using above, we can show that n-step methods converge
- Generalization of 1-step:

$$\max_s \left|\mathbb{E}_\pi[R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s] - v_\pi(s)\right| \leq \gamma \max_s \left|V(s) - v_\pi(s)\right|$$

n-step TD

- n-step return

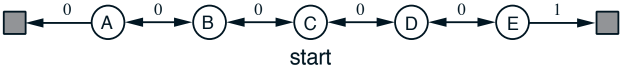
$$\boxed{G_{t:t+n}} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$$

- *Not available* until time $t + n$
- Natural algorithm is to wait until time $t + n$
- n-**step TD** update:

$$V_{\underbrace{t+n}_{\text{next step}}}(S_t) = V_{\underbrace{t+n-1}_{\text{previous step}}}(S_t) + \alpha \Big[G_{t:t+n} - V_{\underbrace{t+n-1}_{\text{previous step}}}(S_t) \Big]$$

```
Initialize V(s) arbitrarily, for all s ∈ S
for all episode do
    Initialize and store S_0 ≠ terminal
    T ← ∞
    repeat for t = 0, 1, 2, ...
        if t < T then
            Take an action according to π(· | S_t)
            Observe and store next reward R_{t+1} and state S_{t+1}
            if S_{t+1} is terminal then T ← t + 1
        end if
        τ ← t − n + 1            ▷ τ is the time whose state's estimate is updated
        if τ ≥ 0 then
            G ← Σ_{i=τ+1}^{min(τ+n,T)} γ^{i−τ−1} R_i
            if τ + n < T then G ← G + γ^n V(S_{τ+n})
            V(S_τ) ← V(S_τ) + α[G − V(S_τ)]
        end if
    until τ = T − 1
end for
```

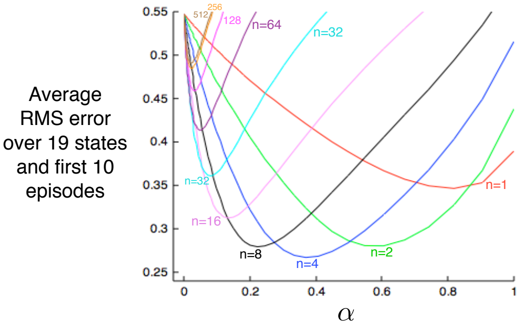
Random walk example



- Start with $V(s) = 0.5$ for all s
- Suppose the first episode progressed directly from C to the right, through D and E
- How does 2-step TD work here?
- How about 3-step TD?
- n-**step TD** update:

$$\underbrace{V_{t+n}}_{\text{next step}}(S_t) = \underbrace{V_{t+n-1}}_{\text{previous step}}(S_t) + \alpha\left[G_{t:t+n} - \underbrace{V_{t+n-1}}_{\text{previous step}}(S_t)\right]$$

19-state random walk



- An intermediate α is best
- An intermediate n is best
- Is there an optimal n? For every task?
- For larger n, smaller α seems best

n-step Sarsa

- Action–value of n-step return:

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

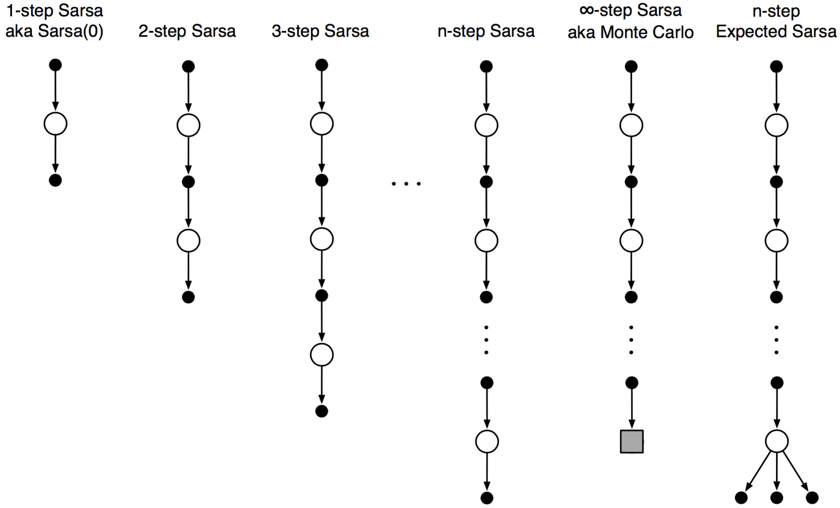
- n-step Sarsa:

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

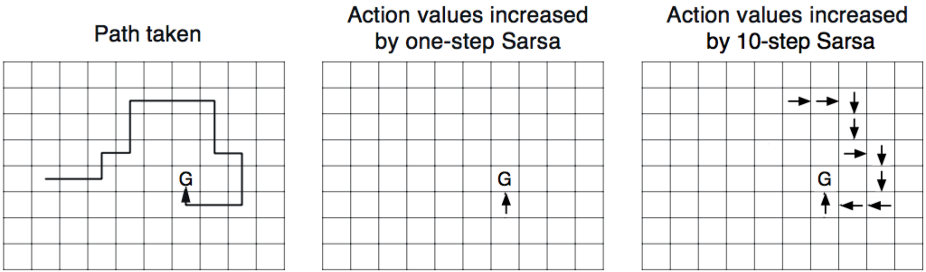
- n-step Expected Sarsa:
  - same update
  - *slightly* different n-step return

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n \sum_a \pi(a \mid S_{t+n}) Q_{t+n-1}(S_{t+n}, a)$$

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

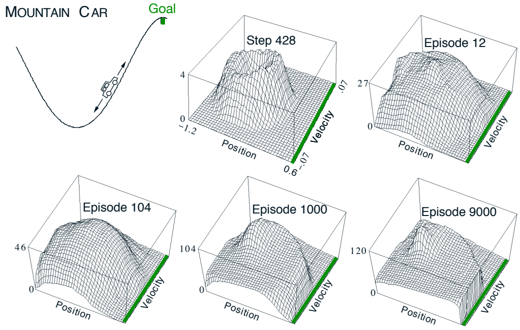
n-step Sarsa example

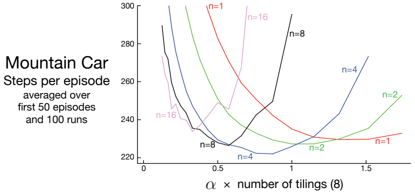


- All values initially 0
- All rewards zero except for a positive reward at G

Initialize action-value function parameterization $\hat{q}$
for all episode **do**
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot \mid S_0)$ or ϵ-greedy wrt $\hat{q}$
  $T \leftarrow \infty$
  repeat for $t = 0, 1, 2, \ldots$
    if $t < T$ **then**
      Take an action A_t
      Observe and store next reward R_{t+1} and state S_{t+1}
      if S_{t+1} is terminal **then** $T \leftarrow t + 1$
      else Select and store an action $A_{t+1} \sim \pi(\cdot \mid S_{t+1})$ or ϵ-greedy wrt $\hat{q}$
    end if
    $\tau \leftarrow t - n + 1$ ▷ τ is the time whose state's estimate is updated
    if $\tau \geq 0$ **then**
      $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
      if $\tau + n < T$ **then** $G \leftarrow G + \gamma^n \hat{q}(S_{\tau+n}, A_{\tau+n})$
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha[G - \hat{q}(S_\tau, A_\tau, \boldsymbol{w})] \boldsymbol{\nabla} \hat{q}(S_\tau, A_\tau, \boldsymbol{w})$
    end if
  until $\tau = T - 1$
end for

n-step Sarsa with function approximation





n-step off–policy learning

- Recall the importance sampling ratio:

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

- Off–policy methods weight updates by this ratio
- Off–policy n-step TD:

$$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha\rho_{t:t+n-1}\Big[G_{t:t+n} - V_{t+n-1}(S_t)\Big]$$

n-step off–policy learning (part 2)

- Recall the importance sampling ratio:

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

- Off–policy n-step Sarsa:

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

- Off–policy n-step Expected Sarsa:

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

$$G_{t:t+n} = R_{t+1} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n \sum_a \pi(a \mid s) Q_{t+n-1}(s, a)$$

Summary

- n-step bootstrapping generalizes TD and MC learning methods
  - $n = 1$ is TD
  - $n = \infty$ is MC
  - intermediate n is often better than either extreme
  - applies to both continuing and episodic domains
- Additional cost in computation
  - we need to remember the last n states
  - learning is delayed by n steps
  - per-step computation is small (like TD)
- Everything generalizes nicely:
  - error-reduction theory
  - Sarsa, off–policy by importance sampling, Expected Sarsa Backup

Eligibility Traces

Scaling-up reinforcement learning

- Sparse rewards
  - e.g. gridworld
- Large *state* spaces:
  - Go: $\log_{10}|S| = \log_{10}(3^{19\times19}) \approx 170 > 82$
  - Camera images , e.g. $\log_{10}|S| = \log_{10}((256^3)^{1280\times720}) \gg 82$ (3 color channels, 8 bits each)
  - Continuous spaces: e.g. inverted pendulum, mobile robot, etc.

---

nb. of atoms in the universe: $N = 10^{82}$ ($\log_{10} N = 82$)

Recall: n-step return

- n-step returns for $n = 1, 2, \ldots \infty$:

$$
\begin{array}{lll}
n = 1 & TD & R_{t+1} + \gamma V(S_{t+1}) \\
n = 2 & & R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) \\
n = 3 & & R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 V(S_{t+3}) \\
\vdots & & \vdots \\
n = \infty & MC & R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{T-1} R_T
\end{array}
$$

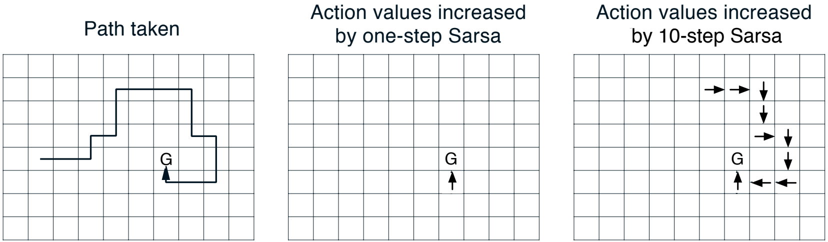
- n-step return:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- n-step temporal difference update:

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

Example: n-step Sarsa



– Reward 0 except for G

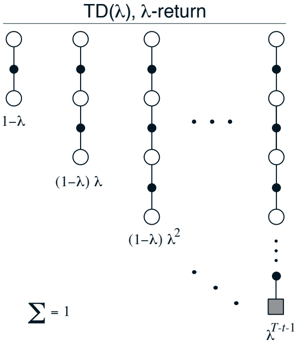
– Which action values would be updated upon reaching the goal?

– How to choose n?

λ-return

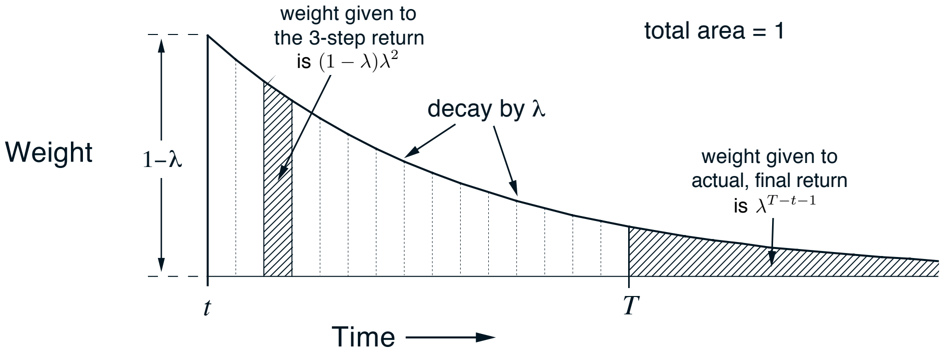
- The λ-return G_t^λ combines all n-step returns (weighted averaging):

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



λ-return weighting function

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



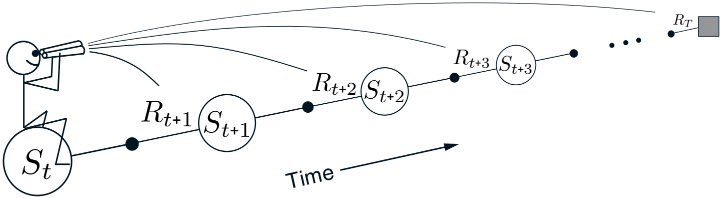
λ-return weighting function (part 2)

- General weighting function:

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

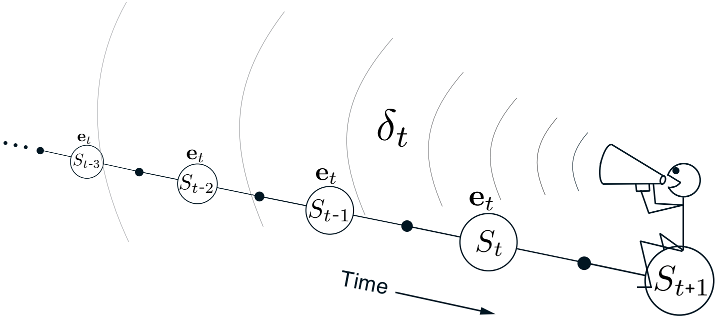
- For $\lambda = 1$: $G_t^\lambda = G_t$ (Monte Carlo)
- For $\lambda = 0$: $G_t^\lambda = G_{t:t+1}$ (1-step TD)

Forward view



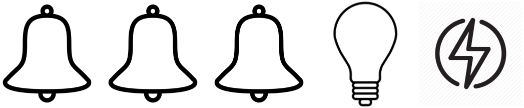
- Update values by looking forward to future rewards and states
- Update values towards λ-return
- Can only be computed for *terminated* sequences

Backward view



- Forward view provides theory
- Backward view provides a mechanism how to perform updates
- Update every step; works for *incomplete* sequences

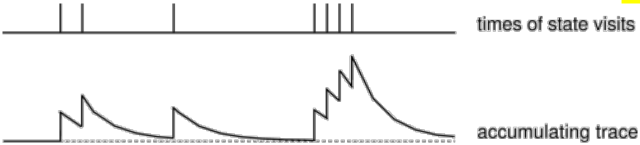
Eligibility traces



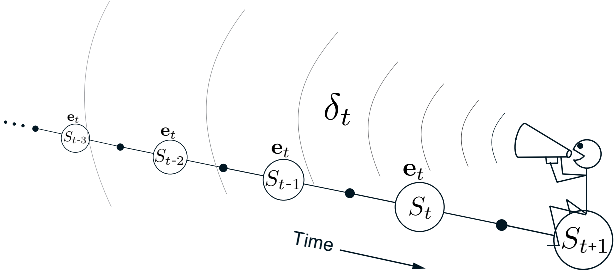
Credit assignment problem:

- **Frequency:** assign credit to most frequent states
- **Recency:** assign credit to recent states

$$\forall s : e(s) \leftarrow \gamma\lambda e(s)$$
$$e(S_t) \leftarrow e(S_t) + 1$$



Backward view



- Keep an eligibility trace for *every* state s
- Update value $V(s)$ for *every* state s:

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

$$\forall s : V(s) \leftarrow V(s) + \alpha \delta_t e_t(s)$$

TD(λ) and TD(0)

- When $\lambda = 0$:

$$e(s) = \begin{cases} 1 & \text{for} \quad s = S_t \\ 0 & \text{else} \end{cases}$$

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

$$\forall s : V(s) \leftarrow V(s) + \alpha \delta_t e_t(s)$$

- Same as TD(0):

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

- What if $\lambda = 1$? Monte-Carlo

TD(λ) with function approximation

- Eligibility trace vector $\boldsymbol{e}$ keeps track which components have contributed to recent state evaluations
- Indicate the *eligibility* of each component for undergoing learning

$$
\begin{aligned}
\delta &= R_{t+1} + \gamma \hat{v}(S_{t+1}, \boldsymbol{w}) - \hat{v}(S_t, \boldsymbol{w}) \\
\boldsymbol{e} &\leftarrow \gamma \lambda \boldsymbol{e} + \nabla \hat{v}(S_t, \boldsymbol{w}) \\
\boldsymbol{w} &\leftarrow \boldsymbol{w} + \alpha \delta \boldsymbol{e}
\end{aligned}
$$

- Update weight vector proportional to scalar TD error and eligibility trace vector

Summary

- We saw a way to **unify TD and MC**
- n-step returns interpolate between TD(0) and MC ($n = \infty$)
- We can get a combination of different n-step returns by using λ returns
- **Eligibility traces** are a computationally efficient way to implement them
- In the high-dimensional function approximation case this becomes tricky to use and is an active area of research