n-step Bootstrapping

Reinforcement Learning

Roberto Capobianco

Sapienza Università di Roma

Recap

Data Generation

---

2 steps:

1. Roll-in
2. **Roll-out & compute supervision targets**

Given s, a, how do we estimate $Q^\pi(s,a)$?

$$Q^\pi(s_t,a_t) = \mathbb{E}[\textstyle\sum^{\infty}_{h=0}\gamma^h r_h \,|\, (s_0,a_0)=(s_t,a_t), a_{h+1}=\pi(s_h), s_{h+1}\sim p(.|s_h,a_h)]$$

Monte-Carlo method: estimate this through sampling, execute until termination and then average many roll-outs to compute our estimate. **Note: True MC cannot be applied if infinite horizon, but we can adapt using the 'trick' shown in previous class**

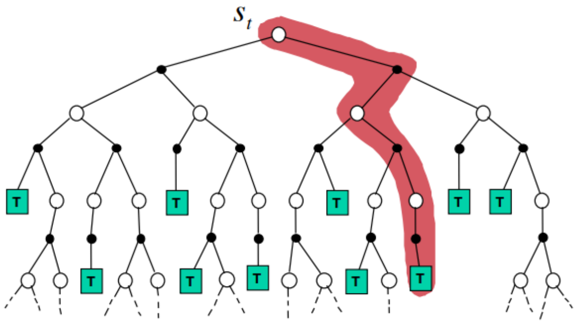
Monte-Carlo Update

$$Q^{\pi}(s_t,a_t) = \mathbb{E}[\sum^{\infty}_{h=0}\gamma^h r_h | (s_0,a_0)=(s_t,a_t), a_{h+1}=\pi(s_h), s_{h+1}\sim p(.|s_h,a_h)]$$

Monte-Carlo method: estimate this through sampling, execute until termination and then average many roll-outs to compute our estimate

Compute target: $y = G$

- G is the return
- The return is the sum of rewards $\sum_{h=0}^{\text{Termination}}\gamma^h r_h$



Credits: David Silver

MC Updates

Tabular:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(G_i - Q(s,a))$$

we do this at every termination

Function Approximator:

$$\mathrm{argmin}_{Q \text{ in } Q} \sum_{i=1}^{N} (Q(s_i, a_i) - G_i)^2$$

we store data and update in batch after a while or do online learning (at every datapoint - less stable)

MC Pros & Cons

Weaknesses:

- Needs some sort of termination
- Depends on many random actions, transitions, rewards
- Needs complete sequences of returns

Strengths:

- Unbiased
- Good convergence properties also with function approx
- Not very sensitive to initialization

Data Generation

---

2 steps:

1. Roll-in
2. **Roll-out & compute supervision targets**

Given s, a, how do we estimate $Q^{\pi}(s,a)$?

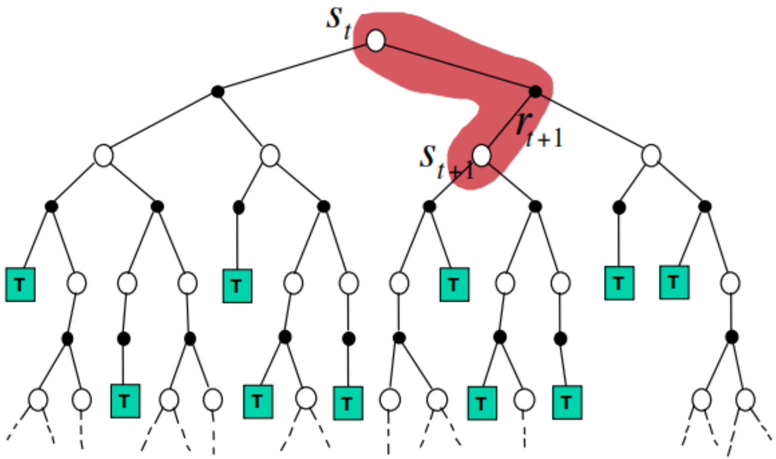
$$\mathbf{Q^{\pi}(s_t,a) = r_t+\gamma\mathbb{E}_{s'\sim p(.|s,a)}[V^{\pi}(s')]}$$

Monte-Carlo uses the actual return. In Temporal Difference we use an estimated return: our current V

Temporal Difference Update

$Q^\pi(s_t,a) = r_t + \gamma \mathbb{E}_{s' \sim p(.|s,a)}[V^\pi(s')]$

Temporal Difference method: estimate this through sampling, update our estimate towards the current reward and the current estimated return (*bootstrapping*) from incomplete episodes



Bootstrapping: an estimate of the next state value is used instead of the true next state value

Credits: David Silver

TD Updates

Tabular:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r_i + \gamma Q(s',.) - Q(s,a))$$

we do this at every timestep

Function Approximator:

$$\text{argmin}_{Q \text{ in } Q} \sum_{i=1}^{N} (Q(s_i, a_i) - r_i + \gamma Q(s',.))^2$$

still we store data and update in batch after a while or do online learning (at every datapoint - less stable), but many more data-points than MC with same experience

TD Pros & Cons

Weaknesses:

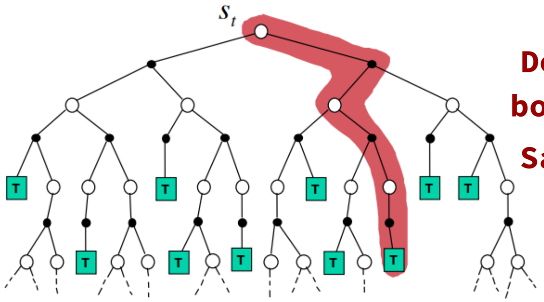
- Sensitive to initial value
- Biased estimate of Q^{π}

Strengths:

- Can learn at every step, from incomplete sequences and in continuing tasks easily
- Depends on just one action instead of a sequence like MC
- Convergences but not always if function approx

MC vs TD vs DP

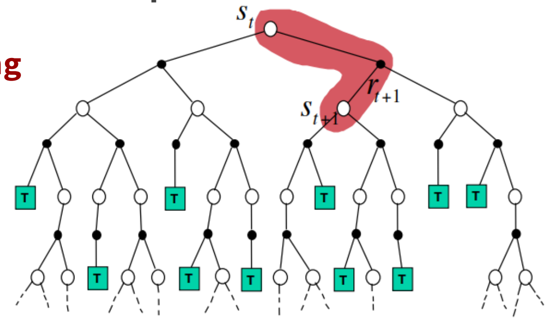
Monte-Carlo



Does not bootstrap

Sampling

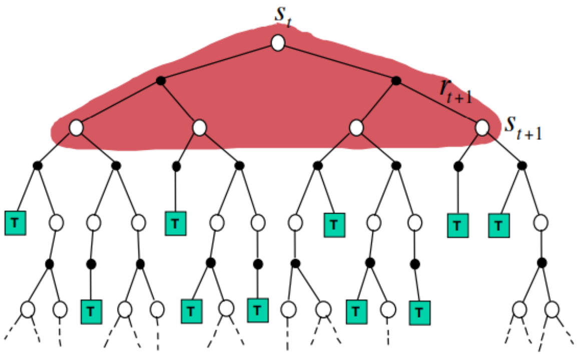
Temporal Difference



Bootstrapping

Sampling

Dynamic Programming



Bootstrapping

Compute expectation

SAPIENZA UNIVERSITÀ DI ROMA

Credits: David Silver

ϵ-Greedy Exploration & Policy Improvement

Simplest idea: instead of only being greedy with respect to Q, try all actions with some probability

- probability 1-ϵ choose the greedy action (do argmax)
- probability ϵ choose a random action

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon & \text{if } a^* = \underset{a \in \mathcal{A}}{\text{argmax}}\ Q(s,a) \\ \epsilon/m & \text{otherwise} \end{cases}$$

This handles the **exploration-exploitation trade-off**

For any ϵ-greedy policy π, the ϵ-greedy policy π' obtained by Q^π is an improvement, such that $V^{\pi'} \geq V^\pi$ holds

If we set ϵ = 1/k, with k going to infinity

- we visit all state-action pairs infinitely many times
- the policy converges to a greedy policy

Greedy in the Limit with Infinite Exploration

ϵ-Greedy and MC

If we apply Greedy in the Limit with Infinite Exploration to MC we converge to the optimal Q*

$$\epsilon \leftarrow 1/k$$
$$\pi \leftarrow \epsilon\text{-greedy}(Q)$$

ϵ-Greedy and TD

Remember $r_i+\gamma Q(s',.)$?

How do we select .?

Sarsa: the target action is selected according to π (which can be eps-greedy with respect to Q)

Selects the target action according to the same policy we execute

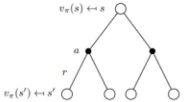
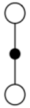
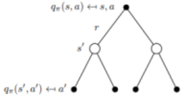
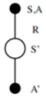
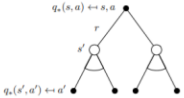
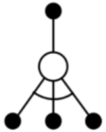
ON-POLICY

Q-learning: the target action is greedy with respect to Q

Selects the target action differently from the policy we execute (which must be eps-greedy, remember?)

OFF-POLICY

TD, Sarsa & Q-Learning vs DP

	Full Backup (DP)	*Sample Backup (TD)*
Bellman Expectation Equation for $v_\pi(s)$	 Iterative Policy Evaluation	 TD Learning
Bellman Expectation Equation for $q_\pi(s, a)$	 Q-Policy Iteration	 Sarsa
Bellman Optimality Equation for $q_*(s, a)$	 Q-Value Iteration	 Q-Learning

Full Backup (DP)	*Sample Backup (TD)*
Iterative Policy Evaluation	TD Learning
$V(s) \leftarrow \mathbb{E}\left[R + \gamma V(S') \mid s\right]$	$V(S) \xleftarrow{\alpha} R + \gamma V(S')$
Q-Policy Iteration	Sarsa
$Q(s, a) \leftarrow \mathbb{E}\left[R + \gamma Q(S', A') \mid s, a\right]$	$Q(S, A) \xleftarrow{\alpha} R + \gamma Q(S', A')$
Q-Value Iteration	Q-Learning
$Q(s, a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S', a') \mid s, a\right]$	$Q(S, A) \xleftarrow{\alpha} R + \gamma \max_{a' \in \mathcal{A}} Q(S', a')$

End Recap

A (statistical) Note on Bootstrapping

Is bootstrapping in RL the same as in statistics?

A (statistical) Note on Bootstrapping

Is bootstrapping in RL the same as in statistics?

Definition from statistics: *Bootstrapping is a method of inferring results for a population from results found on a collection of smaller random samples of that population, using replacement during the sampling process*

Replacement: every time an item is drawn from the pool, the same item remains a part of the sample pool

A (statistical) Note on Bootstrapping

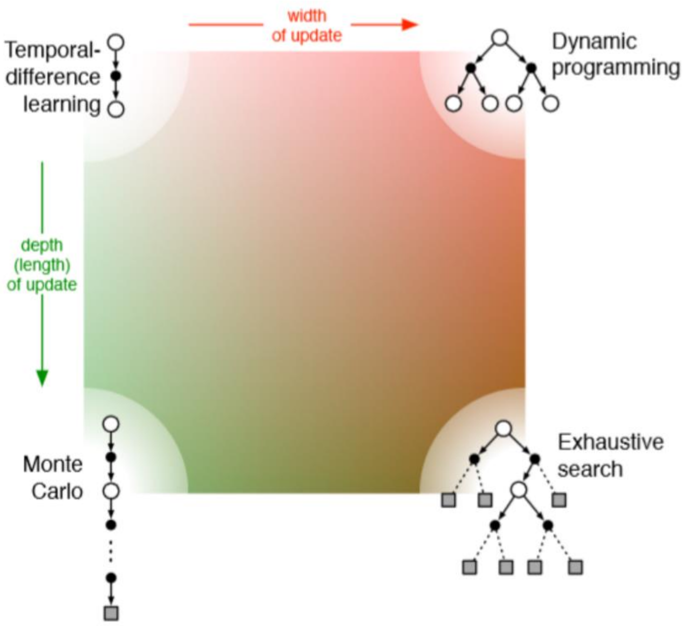
Is bootstrapping in RL the same as in statistics?

Definition from statistics: *Bootstrapping is a method of inferring results for a population from results found on a collection of smaller random samples of that population, using replacement during the sampling process*

Replacement: every time an item is drawn from the pool, the same item remains a part of the sample pool

In RL we are basically doing the same: we infer results (Q) for all the states and actions, inferring them from results found on a collection of smaller random samples (the states that we actually have some information about). All states and actions are always in the 'samplable' pool.

MC vs TD vs DP



width
of update
Temporal-
difference
learning
Dynamic
programming
depth
(length)
of update
Monte
Carlo
Exhaustive
search

MC vs TD vs DP

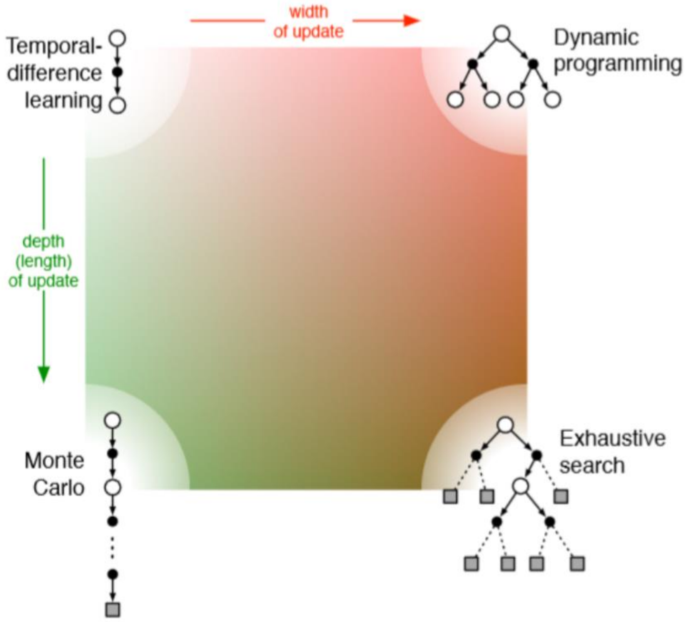
- TD uses a single timestep

    sometimes it’s not enough to acquire any signal or significant state change

- MC uses all timesteps

    leads to high variance

is there any intermediate?

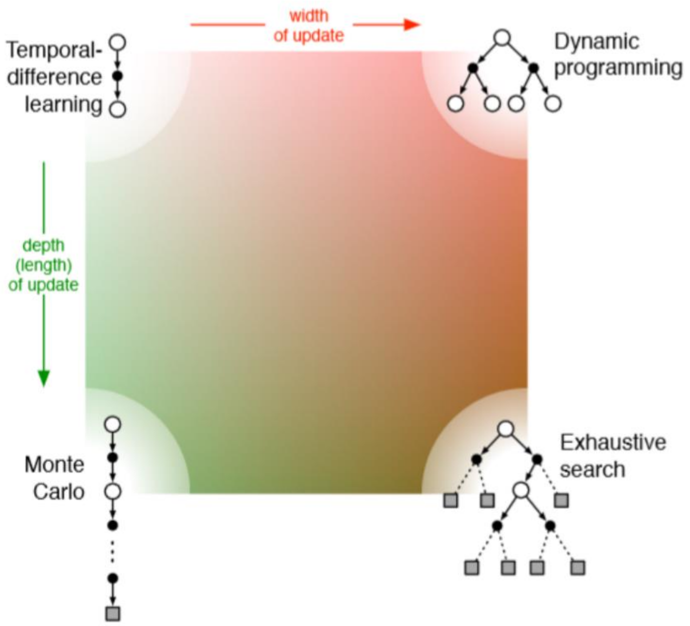


MC vs TD vs DP

- TD uses a single timestep
- MC uses all timesteps

is there any intermediate?

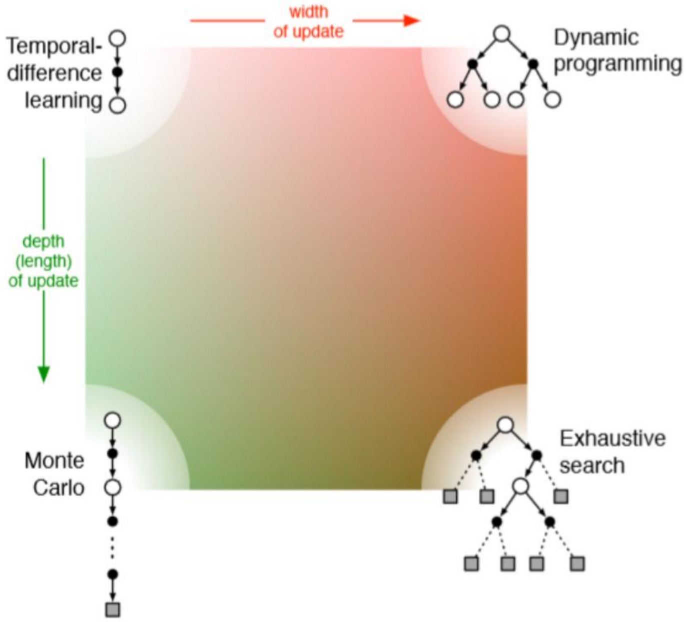
use n-step returns



n-step Returns

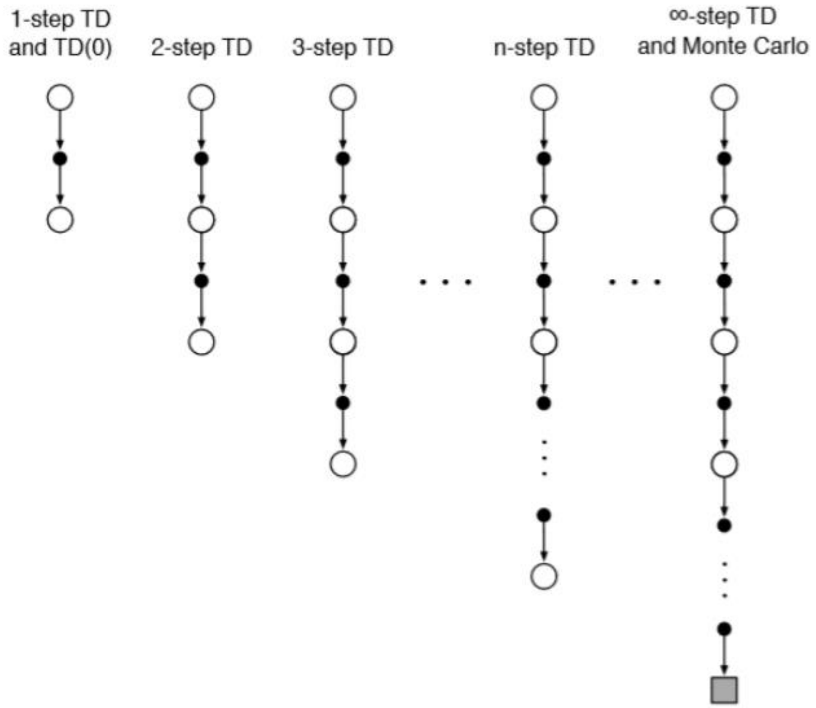
---

n-step returns are still make using of temporal difference, but use more than one reward



n-step Returns

n-step returns are still make using of temporal difference, but use more than one reward



n-step Returns

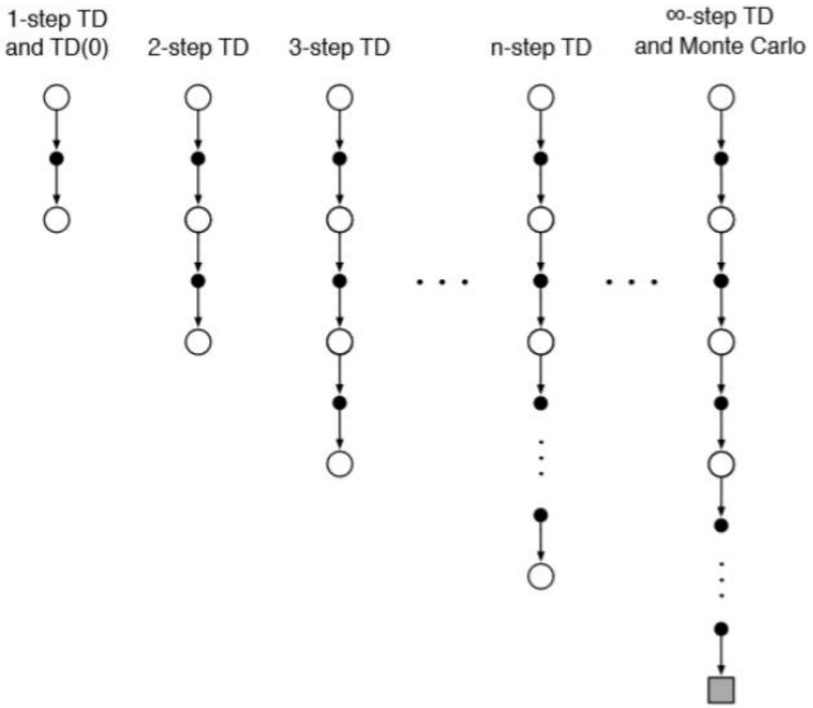
n-step returns are still make using of temporal difference, but use more than one reward

$n = 1 \quad y = G_t^{(1)} = r_t + \gamma V(s_{t+1})$

$n = 2 \quad y = G_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2})$

...

$n = \infty \quad y = G_t = y_{MC}$



n-step Returns

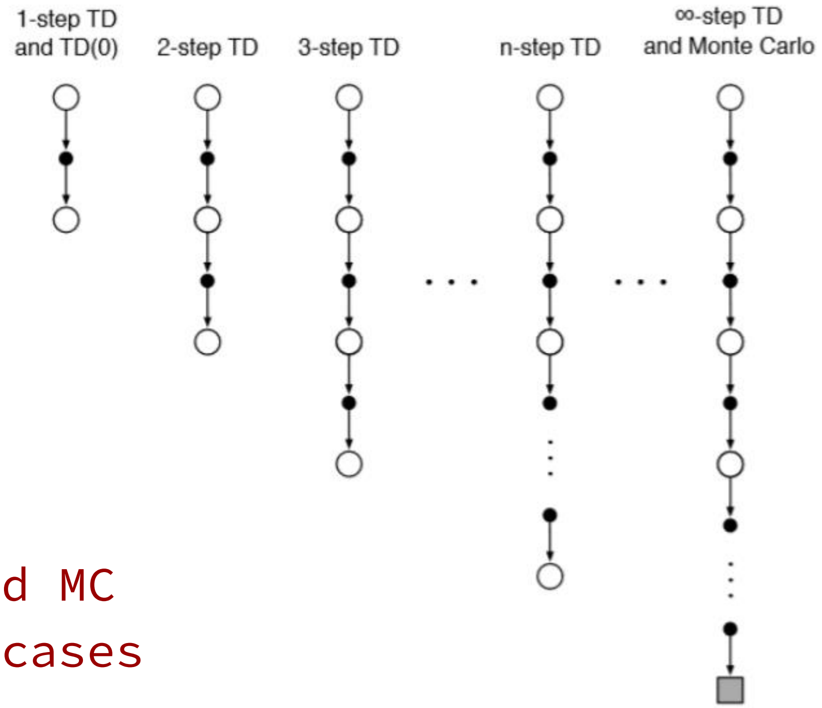
n-step returns are still make using of temporal difference, but use more than one reward

$n = 1 \quad y = G_t^{(1)} = r_t + \gamma V(s_{t+1})$

$n = 2 \quad y = G_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2})$

...

$n = \infty \quad y = G_t = y_{MC}$



1-step TD and MC
just extreme cases

n-step Returns

---

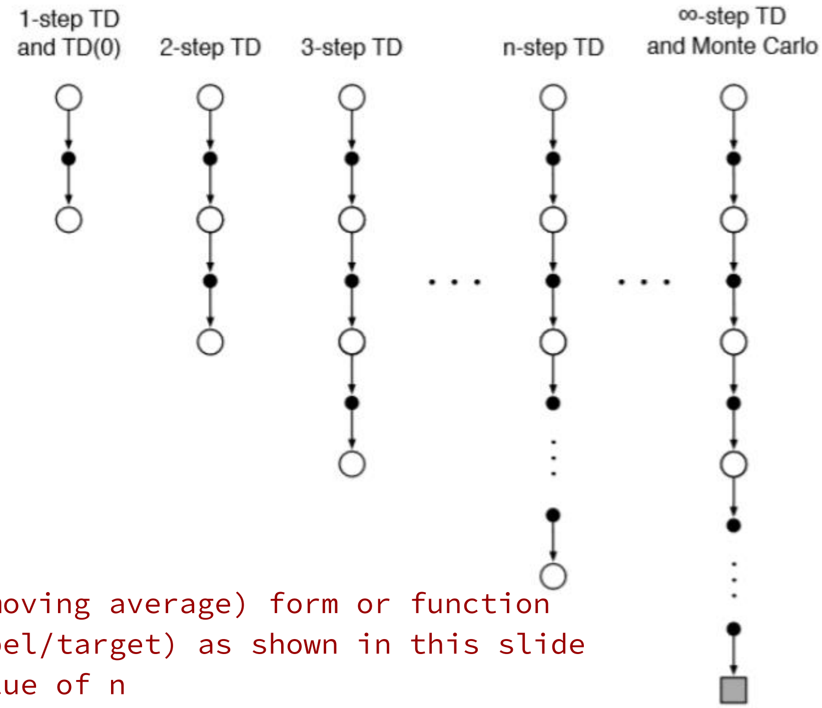
n-step returns are still make using of temporal difference, but use more than one reward

$n = 1 \quad y = G_t^{(1)} = r_t + \gamma V(s_{t+1})$

$n = 2 \quad y = G_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2})$

...

$n = \infty \quad y = G_t = y_{MC}$



we still do our updates using the tabular (moving average) form or function approximation (l2-regression), with y (the label/target) as shown in this slide depending on the value of n

SAPIENZA UNIVERSITÀ DI ROMA

n-step TD in V

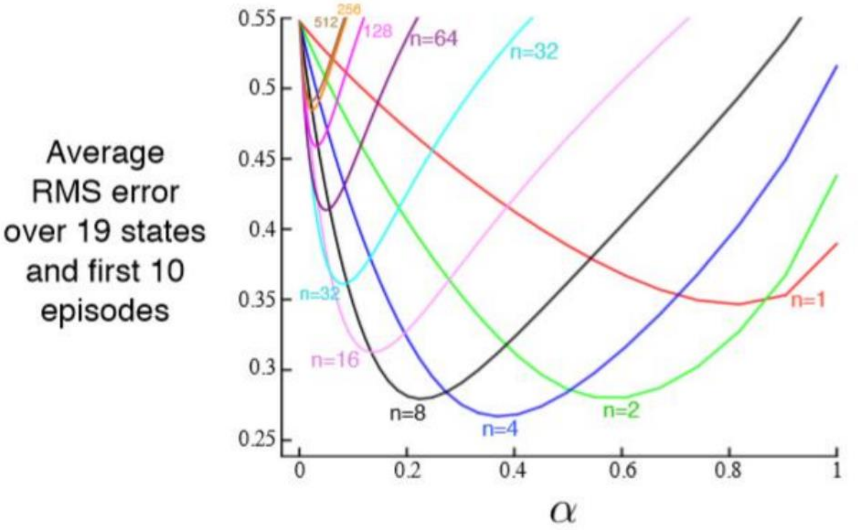
Input: a policy π
Algorithm parameters: step size $\alpha \in (0, 1]$, a positive integer n
Initialize $V(s)$ arbitrarily, for all $s \in \mathcal{S}$
All store and access operations (for S_t and R_t) can take their index mod $n+1$

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 $\tau = T - 1$

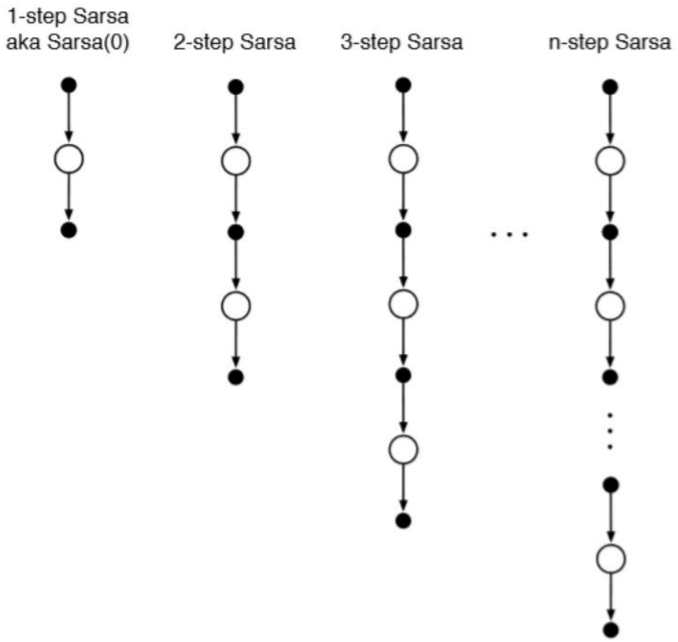
SAPIENZA
UNIVERSITÀ DI ROMA

n-step TD in V

Extremes can potentially perform worse



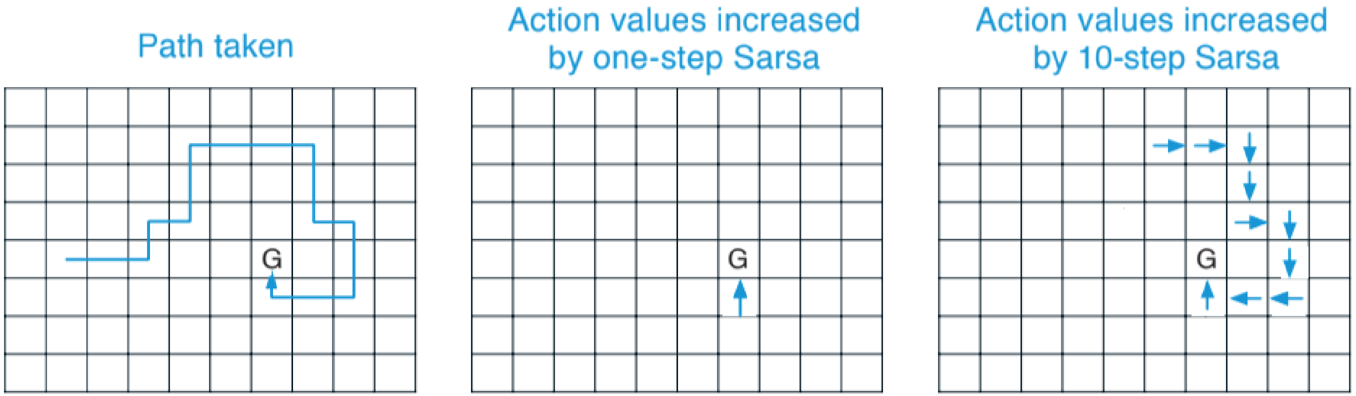
n-step Sarsa



Just compute the target using n-steps

$y = Q_{(n)}{}^{\wedge\pi} = r_t + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n Q^{\wedge\pi}(s_{t+n}, a \sim \pi(s_{t+n}))$

n-step Sarsa: Example



n-steps

But at this point, how do we pick n? Why should we commit to a specific n? Based on what?

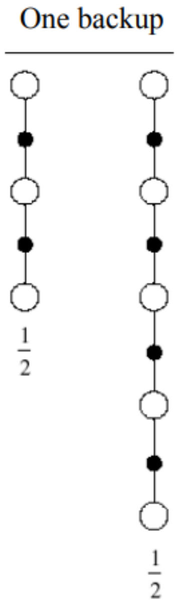
Averaging n-step Returns

We can also solve this: average n-step returns over different lengths, thus combining information from different timesteps

$n = 2 \quad G^{(2)}$

$n = 4 \quad G^{(4)}$

$y = 0.5G^{(2)} + 0.5G^{(4)}$



SAPIENZA
UNIVERSITÀ DI ROMA

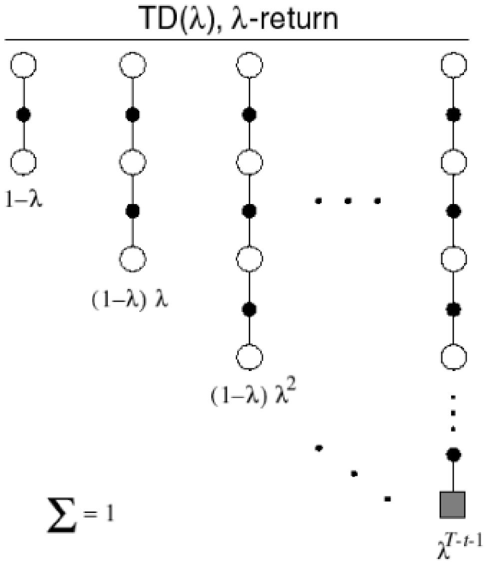
λ-returns

Can we combine information from all timesteps?

λ-returns

Can we combine information from all timesteps?

Yes, λ-returns G_t^λ combine all n-step returns



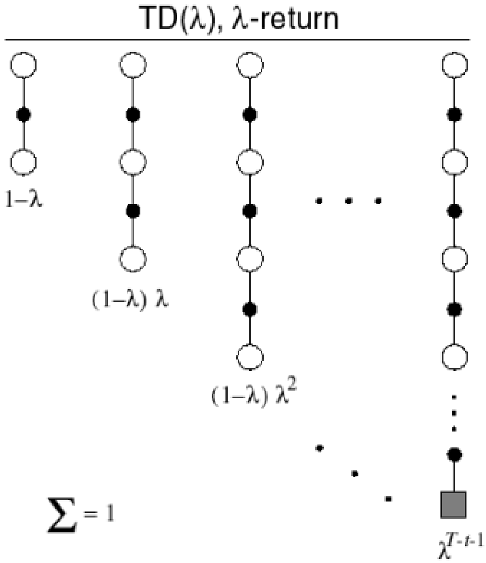
λ-returns

Can we combine information from all timesteps?

Yes, λ-returns G_t^{λ} combine all n-step returns

- using a certain weight $(1-\lambda)\lambda^{n-1}$
- doing a weighted average

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



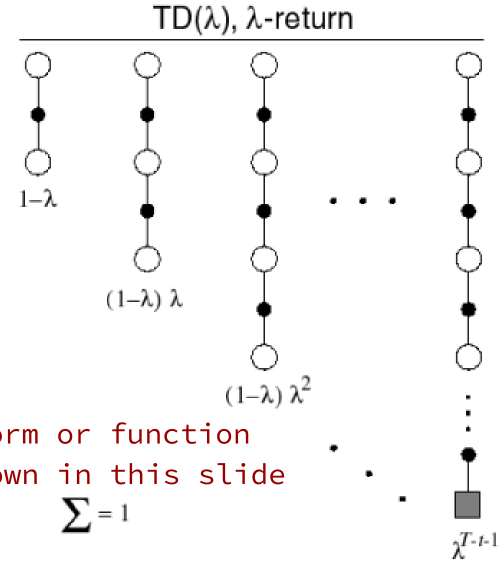
λ-returns & TD(λ)

Can we combine information from all timesteps?

Yes, λ-returns G_t^λ combine all n-step returns

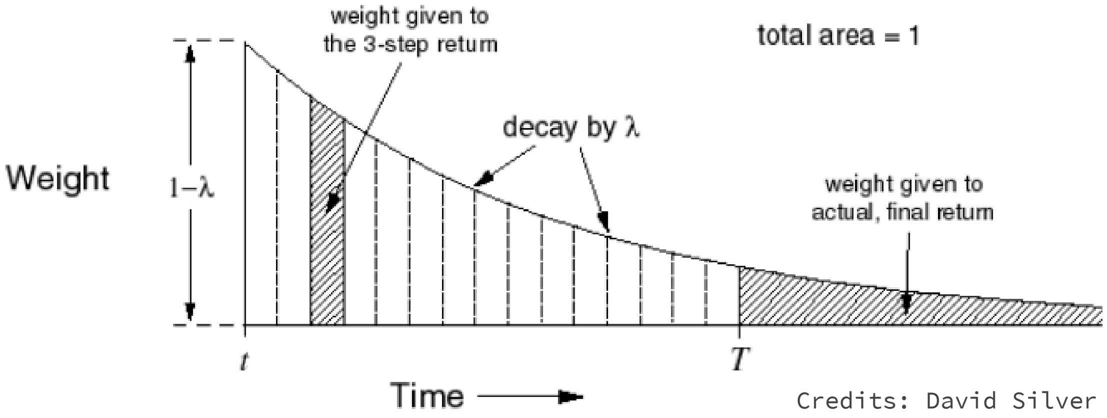
- using a certain weight $(1-\lambda)\lambda^{n-1}$
- doing a weighted average

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



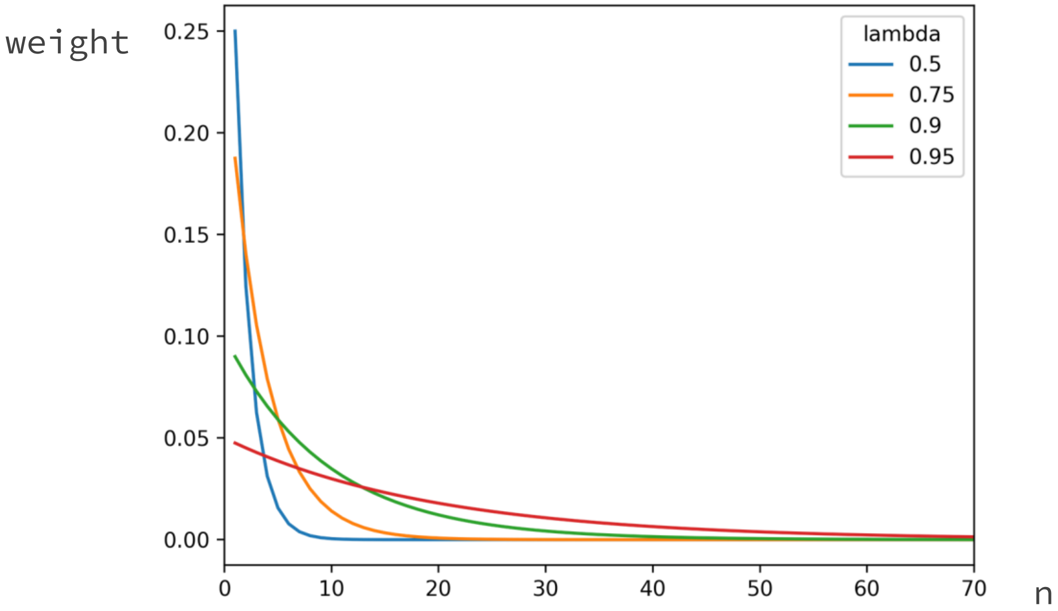
we still do our updates using the tabular (moving average) form or function approximation (l2-regression), with y (the label/target) as shown in this slide depending on the value of n

TD(λ) Weights



Credits: David Silver

TD(λ) Weights



Forward-View TD(λ)

We can combine all n-steps using λ-returns G_t^{λ} but we are back at the problem of MC updates: we need to wait for termination

Forward-View TD(λ)

We can combine all n-steps using λ-returns G_t^λ but we are back at the problem of MC updates: we need to wait for termination

Backward-View TD(λ)

We can actually do better and still preserve the updates at every timestep by maintaining a virtually equivalent intuition under a different perspective: backward-view

Backward-View TD(λ)

We can actually do better and still preserve the updates at every timestep by maintaining a virtually equivalent intuition under a different perspective: backward-view

We will use **eligibility traces**

Eligibility Traces

Instead of waiting for *what is going to happen next*, we will remember *what happened in the past*

Eligibility Traces

Instead of waiting for *what is going to happen next*, we will remember *what happened in the past*

We're specifically looking at a **credit assignment** problem

Credit Assignment

Instead of waiting for *what is going to happen next*, we will remember *what happened in the past*

We're specifically looking at a **credit assignment** problem

At the end of a football match a team either wins or loses. What contribution did each player provide to the win/loss?

This kind of analysis is called credit assignment, and we will assign credit to states (or states and actions)

Credit Assignment

Instead of waiting for *what is going to happen next*, we will remember *what happened in the past*

We're specifically looking at a **credit assignment** problem

Heuristics:

- **frequency:** most frequent states get credit
- **recency:** most recent states get credit

Eligibility Traces

Eligibility traces combine both credit assignment heuristics:

- keep an eligibility trace for all states s in S
- compute the eligibility trace as

$$e_0(s) = 0$$

$$e_t(s) = \gamma\lambda e_{t-1}(s) + \mathbf{I}(s_t=s)$$

Eligibility Traces

Eligibility traces combine both credit assignment heuristics:

- keep an eligibility trace for all states s in S
- compute the eligibility trace as

$$e_0(s) = 0$$

$$e_t(s) = \gamma\lambda e_{t-1}(s) + \mathbf{I}(s_t=s)$$

I is the indicator function, which equals 1 if the condition inside is true, 0 otherwise

Eligibility Traces

Eligibility traces combine both credit assignment heuristics:

- keep an eligibility trace for all states s in S
- compute the eligibility trace as

$$e_0(s) = 0$$

$$e_t(s) = \boxed{\gamma}\lambda e_{t-1}(s) + \mathbf{I}(s_t=s)$$

recency

Eligibility Traces

Eligibility traces combine both credit assignment heuristics:

- keep an eligibility trace for all states s in S
- compute the eligibility trace as

$$e_0(s) = 0$$

$$e_t(s) = \gamma\lambda e_{t-1}(s) + \boxed{\mathbf{I}(s_t=s)}$$

frequency

Eligibility Traces

Eligibility traces combine both credit assignment heuristics:

- keep an eligibility trace for all states s in S
- compute the eligibility trace as

$$e_0(s) = 0$$

$$e_t(s) = \gamma\lambda e_{t-1}(s) + \mathbf{I}(s_t=s)$$

now use eligibility traces as scaling factor for TD error

Eligibility Traces

- keep an eligibility trace for all states s in S
- compute the eligibility trace as

$$e_0(s) = 0$$

$$e_t(s) = \gamma\lambda e_{t-1}(s) + \mathbf{I}(s_t = s)$$

now use eligibility traces as scaling factor for TD error

Eligibility Traces

- keep an eligibility trace for all states s in S
- compute the eligibility trace as

$$e_0(s) = 0$$

$$e_t(s) = \gamma\lambda e_{t-1}(s) + \mathbf{I}(s_t=s)$$

TD error (in V): $\delta_t = r_t + \gamma V^{\wedge\pi}(s_{t+1}) - V^{\wedge\pi}(s_t)$

- for **ALL** the states s in S update the value function estimate

$$V^{\wedge\pi}(s) \leftarrow V^{\wedge\pi}(s) + \alpha\delta_t e_t(s)$$

Eligibility Traces

- keep an eligibility trace for all states s in S
- compute the eligibility trace as

$$e_0(s) = 0$$

$$e_t(s) = \gamma\lambda e_{t-1}(s) + \mathbf{I}(s_t=s)$$

TD error (in V): $\delta_t = r_t + \gamma V^{\wedge\pi}(s_{t+1}) - V^{\wedge\pi}(s_t)$

- for **ALL** the states s in S update the value function estimate

$$V^{\wedge\pi}(s) \leftarrow V^{\wedge\pi}(s) + \alpha\delta_t e_t(s)$$

We do our updates for all the states at each timestep: states that are not visited or haven't been visited in a while will have an eligibility value equal or close to zero, which means that our estimate for those states won't change much

Eligibility Traces

- keep an eligibility trace for all states s in S
- compute the eligibility trace as

$$e_0(s) = 0$$

$$e_t(s) = \gamma\lambda e_{t-1}(s) + \mathbf{I}(s_t=s)$$

TD error (in V): $\delta_t = r_t + \gamma V^{\wedge\pi}(s_{t+1}) - V^{\wedge\pi}(s_t)$

- for **ALL** the states s in S update the value function estimate

$$V^{\wedge\pi}(s) \leftarrow V^{\wedge\pi}(s) + \alpha\delta_t e_t(s)$$

We propagate current error information (δ_t) into the states we visited in the past. This allows us to combine the n-step returns in an online fashion.

Eligibility Traces

- keep an eligibility trace for all states s in S
- compute the eligibility trace as

$$e_0(s) = 0$$

$$e_t(s) = \gamma\lambda e_{t-1}(s) + \mathbf{I}(s_t=s)$$

TD error (in V): $\delta_t = r_t + \gamma V^{\wedge\pi}(s_{t+1}) - V^{\wedge\pi}(s_t)$

- for **ALL** the states s in S update the value function estimate

$$V^{\wedge\pi}(s) \leftarrow V^{\wedge\pi}(s) + \alpha\delta_t e_t(s)$$

This is still a TD method: still biased because of the bootstrapping. It should intuitively have less variance than MC because of the averaging of low-variance information, but this is not proved

Eligibility Traces and λ Values (0)

If $\lambda = 0$

$$e_0(s) = 0$$

$$e_t(s) = \mathbf{I}(s_t = s)$$

As a result we basically update only s_t as we do in standard TD(0)

$$V^{\wedge\pi}(s_t) \leftarrow V^{\wedge\pi}(s_t) + \alpha\delta_t$$

Eligibility Traces and λ Values (1)

If $\lambda = 1$ credit is maintained until the end of the episode

As a result, over the course of an episode, the total update to each state is the same as the total update for MC if the value function is updated just at the end of the episode (e.g., through a batch update)

Eligibility Traces and λ Values (1)

Consider a state s *visited only once at time k*:

- $e_t(s) = 0$ if $t < k$
- $e_t(s) = \gamma^{t-k}$ if $t \geq k$

The total updates that it cumulates are:

$$\sum_{t=0}^{T-1} \alpha \delta_t e_t(s) = \alpha \sum_{t=k}^{T-1} \gamma^{t-k} \delta_t = \alpha(G_k - V^{\wedge\pi}(s_t))$$

Eligibility Traces and λ Values (1)

Consider a state s *visited only once at time k*:

- $e_t(s) = 0$ if $t < k$
- $e_t(s) = \gamma^{t-k}$ if $t \geq k$

The total updates that it cumulates are:

$$\sum_{t=0}^{T-1} \alpha \delta_t e_t(s) = \alpha \sum_{t=k}^{T-1} \gamma^{t-k} \delta_t = \alpha(G_k - V^{\wedge\pi}(s_t))$$

$$\delta_k + \gamma\delta_{k+1} + \gamma^2\delta_{k+2} + \ldots + \gamma^{T-1-k}\delta_{T-1}$$

Eligibility Traces and λ Values (1)

$$\delta_k+\gamma\delta_{k+1}+\gamma^2\delta_{k+2}+...+\gamma^{T-1-k}\delta_{T-1} =$$

$$r_k+\gamma V^{\wedge\pi}(s_{k+1})-V^{\wedge\pi}(s_k) \ +$$

$$\gamma r_{k+1}+\gamma^2 V^{\wedge\pi}(s_{k+2})-\gamma V^{\wedge\pi}(s_{k+1}) \ +$$

$$\gamma^2 r_{k+2}+\gamma^3 V^{\wedge\pi}(s_{k+3})-\gamma^2 V^{\wedge\pi}(s_{k+2}) \ +$$

$$...$$

$$\gamma^{T-1-k} r_{T-1}$$

Eligibility Traces and λ Values (1)

$$\delta_k+\gamma\delta_{k+1}+\gamma^2\delta_{k+2}+...+\gamma^{T-1-k}\delta_{T-1} =$$

$$r_k+\cancel{\gamma V^{\pi}(s_{k+1})}-V^{\pi}(s_k) \quad +$$

$$\gamma r_{k+1}+\cancel{\gamma^2 V^{\pi}(s_{k+2})}\cancel{-\gamma V^{\pi}(s_{k+1})} \quad +$$

$$\gamma^2 r_{k+2}+\cancel{\gamma^3 V^{\pi}(s_{k+3})}\cancel{-\gamma^2 V^{\pi}(s_{k+2})} \quad +$$

$$...$$

$$\gamma^{T-1-k} r_{T-1}$$

Eligibility Traces and λ Values (1)

$$\delta_k+\gamma\delta_{k+1}+\gamma^2\delta_{k+2}+...+\gamma^{T-1-k}\delta_{T-1} =$$

$$r_k+\cancel{\gamma V^{\wedge\pi}(s_{k+1})}-V^{\wedge\pi}(s_k) \;+$$

$$\gamma r_{k+1}+\cancel{\gamma^2 V^{\wedge\pi}(s_{k+2})}-\cancel{\gamma V^{\wedge\pi}(s_{k+1})} \;+$$

$$\gamma^2 r_{k+2}+\cancel{\gamma^3 V^{\wedge\pi}(s_{k+3})}-\cancel{\gamma^2 V^{\wedge\pi}(s_{k+2})} \;+$$

$$...$$

$$\gamma^{T-1-k}r_{T-1} =$$

$$r_k+\gamma r_{k+1}+\gamma^2 r_{k+2}+\gamma^{T-1-k}r_{T-1}-V^{\wedge\pi}(s_k) = G_t-V^{\wedge\pi}(s_k)$$

Eligibility Traces and λ Values (General)

Consider a state s *visited only once at time k*:

- $e_t(s) = 0$ if $t < k$
- $e_t(s) = (\gamma\lambda)^{t-k}$ if $t \geq k$

The total updates that it cumulates are:

$$\sum_{t=0}^{T-1} \alpha \delta_t e_t(s) = \alpha \sum_{t=k}^{T-1} (\gamma\lambda)^{t-k} \delta_t = \alpha(G_k^{\lambda} - V^{\wedge\pi}(s_t))$$

You can expand the definition of G_k^{λ} and prove it

Forward & Backward View Equivalence

Theorem: The sum of offline updates is identical for forward-view and backward-view

$$\sum_{t=0}^{T-1} \alpha \delta_t e_t(s) = \sum_{t=0}^{T-1} \alpha (G_t^{\lambda} - V^{\wedge\pi}(s_t)) \mathbf{I}(s_t = s)$$

Forward & Backward View Equivalence

Theorem: The sum of offline updates is identical for forward-view and backward-view

$$\sum_{t=0}^{T-1}\alpha\delta_t e_t(s) = \sum_{t=0}^{T-1}\alpha(G_t^{\lambda} - V^{\wedge\pi}(s_t))\mathbf{I}(s_t=s)$$

- **offline:** apply update in batch at the end of the episode
- **online:** apply update at every timestep

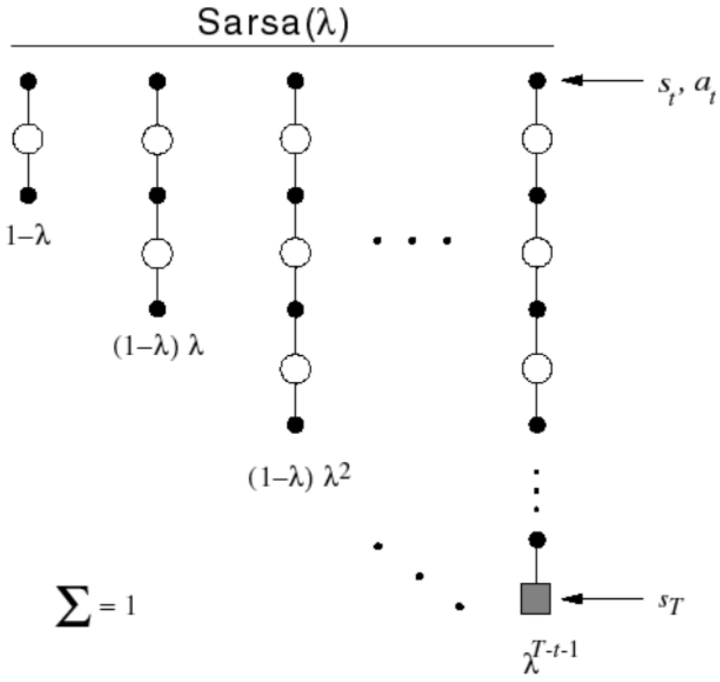
Sapienza Università di Roma

Forward & Backward View Equivalence

Offline updates	$\lambda = 0$	$\lambda \in (0, 1)$	$\lambda = 1$
Backward view Forward view	TD(0) $\parallel$ TD(0)	TD(λ) $\parallel$ Forward TD(λ)	TD(1) $\parallel$ MC
Online updates	$\lambda = 0$	$\lambda \in (0, 1)$	$\lambda = 1$
Backward view Forward view	TD(0) $\parallel$ TD(0)	TD(λ) $\nparallel$ Forward TD(λ)	TD(1) $\nparallel$ MC

Credits: David Silver

Sarsa-λ: Forward View



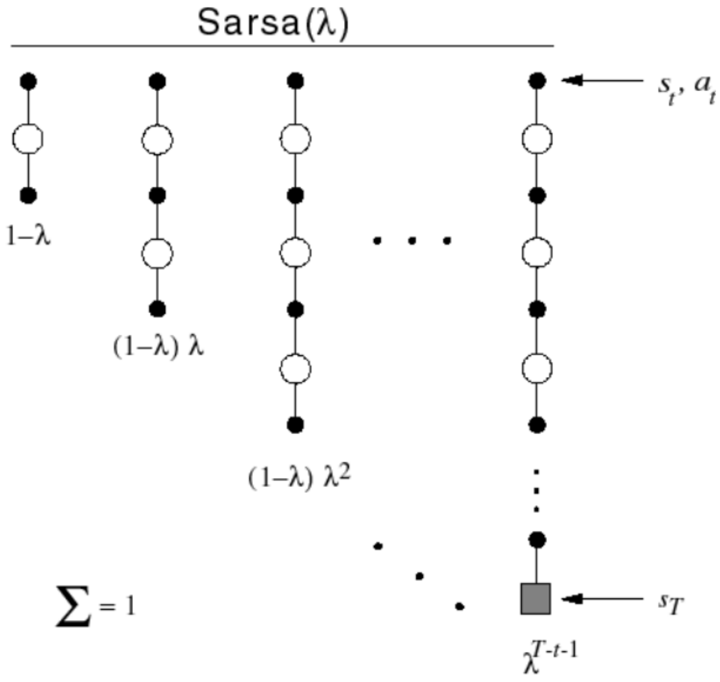
Forward view:

$y = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}Q_{(n)}^{\wedge\pi}$

$Q_{(n)}^{\wedge\pi} = r_t + \ldots + \gamma^{n-1}r_{t+n-1} + \gamma^n Q^{\wedge\pi}(s_{t+n}, a\sim\pi(s_{t+n}))$

$Q(s,a) \leftarrow Q(s,a) + \alpha(y - Q(s,a))$

Sarsa-λ: Backward View



Backward view:

$e_0(s,a) = 0$ and $e_t(s,a)=\gamma e_{t-1}(s,a)+\mathbf{I}(s_t=s,a_t=a)$ for all s,a

Update for all s,a every time:

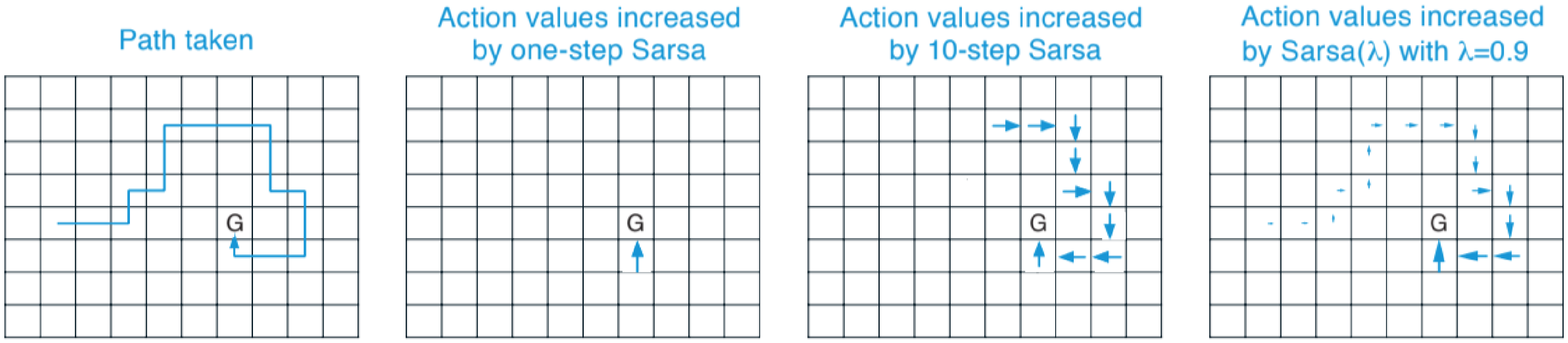
$\delta_t = r_t + \gamma Q^{\wedge\pi}(s_{t+1},a\sim\pi(s_{t+1})) - Q^{\wedge\pi}(s_t,a_t)$

$Q(s,a) \leftarrow Q(s,a) + \alpha\delta_t e_t(s,a)$

Sarsa-λ: Backward View

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
Repeat (for each episode):
    $E(s,a) = 0$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$
    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)
        $\delta \leftarrow R + \gamma Q(S', A') - Q(S, A)$
        $E(S,A) \leftarrow E(S,A) + 1$
        For all $s \in \mathcal{S}, a \in \mathcal{A}(s)$:
            $Q(s,a) \leftarrow Q(s,a) + \alpha \delta E(s,a)$
            $E(s,a) \leftarrow \gamma \lambda E(s,a)$
        $S \leftarrow S'$; $A \leftarrow A'$
    until S is terminal

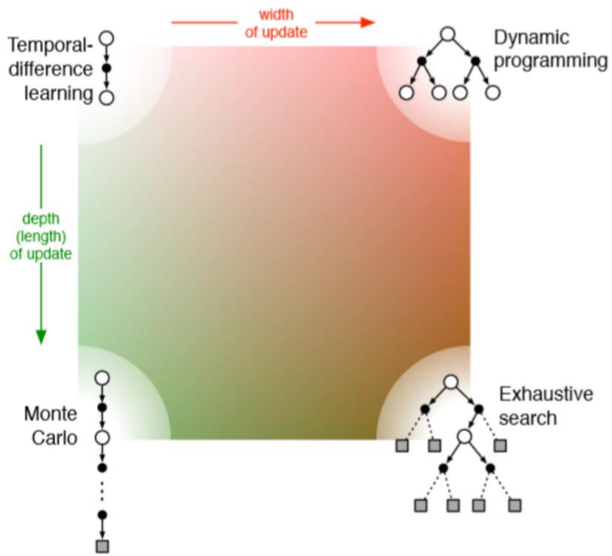
Sarsa-λ: Example



Going Wide vs Going Deep

So far we analyzed the depth of our updates, by considering the amount of sampling, which is:

- cheap computationally
- affected by sampling error
- easy to collect directly from the environment

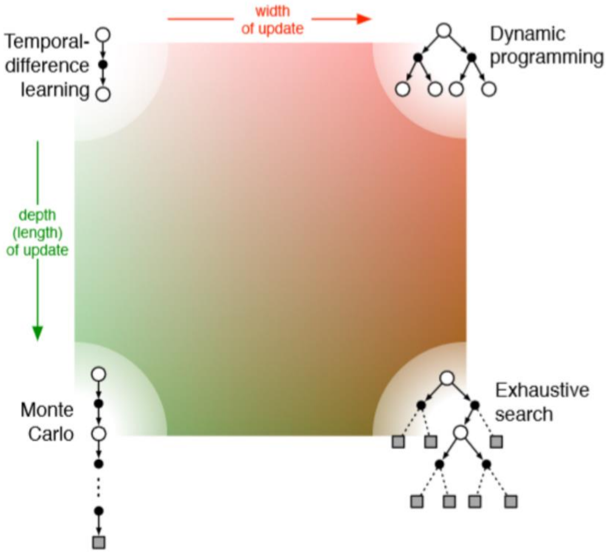


Going Wide vs Going Deep

So far we analyzed the depth of our updates, by considering the amount of sampling, which is:

- cheap computationally
- affected by sampling error
- easy to collect directly from the environment

Can we also go wider with our updates?

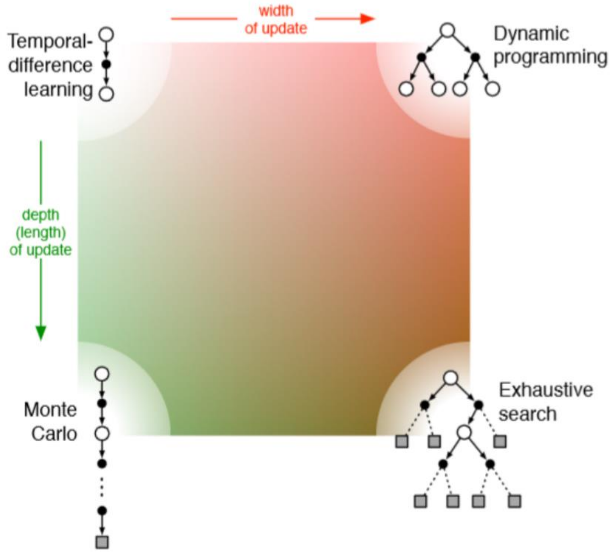


Going Wide vs Going Deep

So far we analyzed the depth of our updates, by considering the amount of sampling, which is:

- cheap computationally
- affected by sampling error
- easy to collect directly from the environment

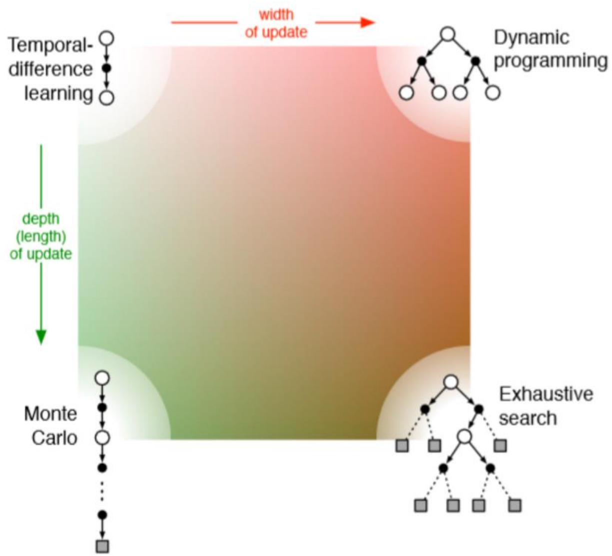
Yes, we can use expected updates



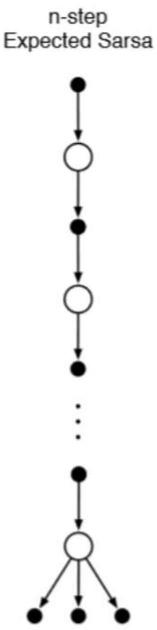
Going Wide vs Going Deep

Expected updates are

- computationally heavier
- exact (no sampling error)
- impossible to obtain without some distribution model



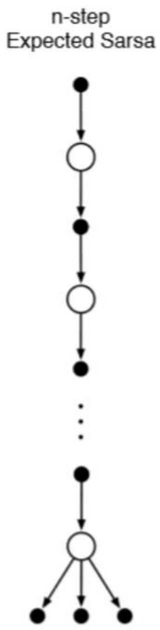
n-step Expected Sarsa



Just compute the target using n-steps and an expectation over the policy at the end

$$y = Q_{(n)}{}^{\wedge\pi} = r_t + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n \sum_a \pi(a|s_{t+n}) Q^{\wedge\pi}(s_{t+n}, a)$$

n-step Expected Sarsa

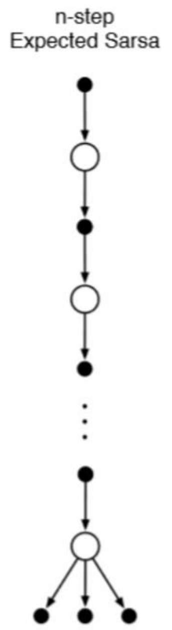


Just compute the target using n-steps and an expectation over the policy at the end

$$y = Q_{(n)}{}^{\wedge\pi} = r_t + \dots + \gamma^{n-1} r_{t+n-1} + \gamma^n \textstyle\sum_a \pi(a|s_{t+n}) Q^{\wedge\pi}(s_{t+n}, a)$$

All sample transitions except the last one where we use a full distribution

n-step Expected Sarsa

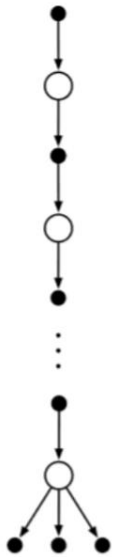


Just compute the target using n-steps and an expectation over the policy at the end

$$y = Q_{(n)}{}^{\wedge\pi} = r_t + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n \sum_a \pi(a|s_{t+n}) Q^{\wedge\pi}(s_{t+n}, a)$$

A tree-backup algorithm would have no sampling and all expectations

n-step Expected Sarsa



Just compute the target using n-steps and an expectation over the policy at the end

$$y = Q_{(n)}^{\wedge\pi} = r_t + \dots + \gamma^{n-1} r_{t+n-1} + \gamma^n \sum_a \pi(a|s_{t+n}) Q^{\wedge\pi}(s_{t+n}, a)$$

Can we handle the degree of sampling/expectation at each step?

Q(σ)

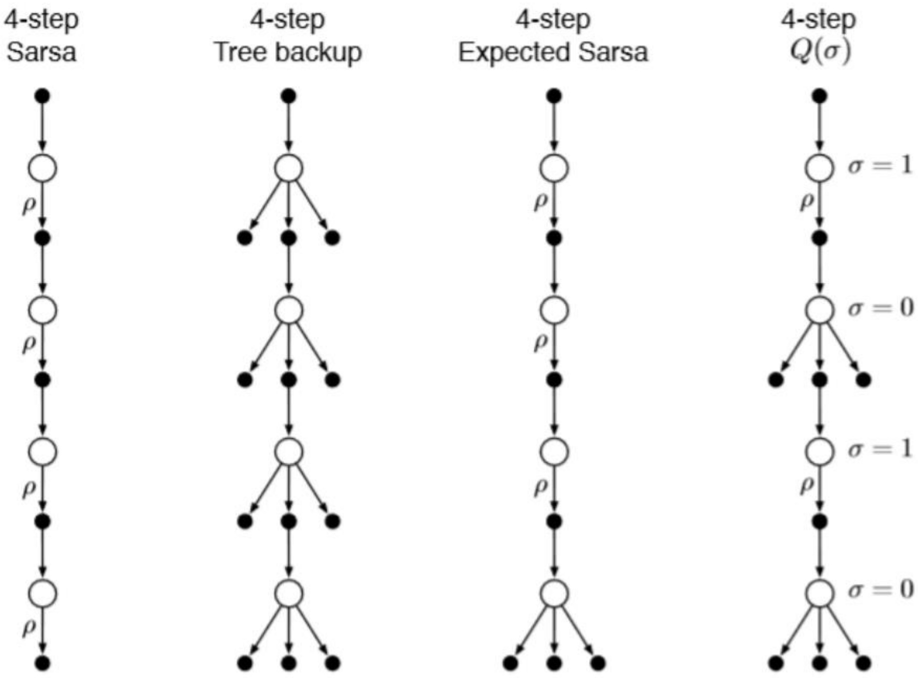
Use $\boldsymbol{\sigma}_t$ in [0,1] to denote the degree of sampling at each timestep:

- 1 means full sampling
- 0 means full expectation
- can be a function of the state

At each timestep the TD error is

$$\delta_t = r_t + \gamma(\boldsymbol{\sigma}_t Q^{\wedge\pi}(s_{t+1}, a \sim \pi(s_{t+1})) + (1-\boldsymbol{\sigma}_t)\sum_a \pi(a|s_{t+1}) Q^{\wedge\pi}(s_{t+1}, a)) - Q^{\wedge\pi}(s_t, a_t)$$

Q(σ)


4-step
Sarsa
4-step
Tree backup
4-step
Expected Sarsa
4-step
$Q(\sigma)$
ρ
$\sigma = 1$
$\sigma = 0$
$\sigma = 1$
$\sigma = 0$