Model-free methods & importance weighting

Herke van Hoof

AI THESIS FAIR
3 OCTOBER, 2019

Location
Sporthal 2
USC Gym, Science Park

Registration
11:30 - 12:00
Speeddates
12:30 - 15:00
Borrel
15:30-18:00

4 SEPTEMBER 2019
AI Projects on DataNose become available

10 SEPTEMBER 2019
RSVP due for Pizza Night

13 SEPTEMBER 2019
Pizza Night - presentation on the thesis and voting begins

13 SEPTEMBER 2019
RSVP due for AI Thesis Fair

13 SEPTEMBER - 20 SEPTEMBER
Voting is open

26 SEPTEMBER 2019
Schedules sent to students and organizations

3 OCTOBER 2019
AI Thesis Fair!

Thesis Marketplace and voting is on DataNose. See email from Yasmin Santis for the links

Questions?
thesisfairamsterdam@gmail.com

Today

Finish last lecture on MC techniques for model-free learning

- Monte-Carlo prediction and control

Recap another method for model-free learning

- Temporal-difference prediction and control

Big picture so far

Finding optimal policies

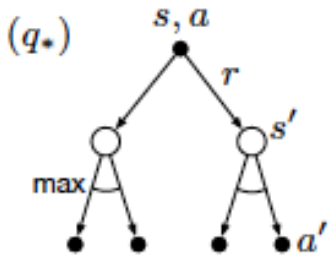
Known MDP dynamics

$$p(s', r|s, a)$$

Dynamic programming

Policy iteration

Value iteration



Only **data** from MDP: Reinforcement learning

$$D = \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}_{i=1\ldots N}$$

Monte-Carlo

Monte Carlo for control

Finding the optimal policy can be ensured if we keep visiting each state action pair

Last week, I mentioned there are two ways to guarantee every state-action pair is visited

- Like in the bandit case, use policy that takes each action with non-zero probability
- 'Exploring starts': Start from random (s,a) pair. Not always possible...

Getting rid of exploring starts

In policy improvement step, greedy update means exploration is lost...

Two possibilities:

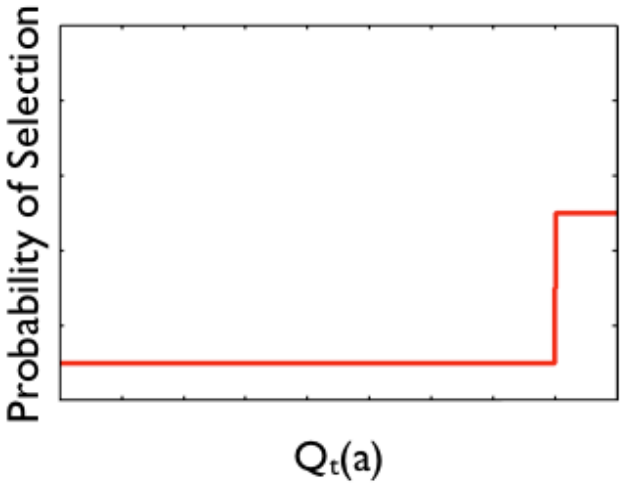
- Change policy update to move “towards” greedy policy, but keep exploring
  We use data from the policy we are updating: **on-policy**
- Use two policies: non-greedy behaviour policy and greedy target policy
  Now, we are using data from a different policy then the one we are updating: **off-policy**

Both possibilities require a different approach. Distinction on-policy and off-policy important for many methods!

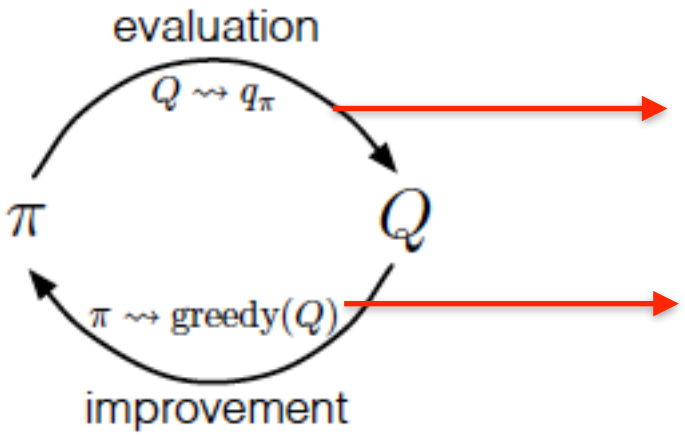
On-policy MC control

Ensure any action is taken with non-0 probability

Example: ε-greedy from last lecture



Again follow the GPI recipe:



Move 'towards' q_π

Move 'towards' greedy(Q) by making policy ε-greedy

Figure: Sutton&Barto, RL:AI

On-policy MC control

Making policy ε-greedy wrt q is indeed an improvement wrt any ε-soft policy that takes any action with probability > ε/|A(s)|

$$q_\pi(s, \pi'(s)) = \sum_a \pi'(a|s)\, q_\pi(s,a)$$

$$= \frac{\epsilon}{|A(s)|} \sum_a q_\pi(s,a) + (1-\epsilon) \max_a q_\pi(s,a)$$

$$\geq \frac{\epsilon}{|A(s)|} \sum_a q_\pi(s,a) + (1-\epsilon) \sum_a \frac{\pi(a|s) - \frac{\epsilon}{|A(s)|}}{1-\epsilon}\, q_\pi(s,a)$$

$$= \frac{\epsilon}{|A(s)|} \sum_a q_\pi(s,a) - \frac{\epsilon}{|A(s)|} \sum_a q_\pi(s,a) + \sum_a \pi(a|s)\, q_\pi(s,a)$$

$$= \sum_a \pi(a|s)\, q_\pi(s,a) = v_\pi(s)$$

then expected q under old policy.

weights.

On-policy MC control

Making policy ε-greedy wrt q is indeed an improvement wrt any ε-soft policy that takes any action with probability > ε/|A(s)|

$$q_\pi\left(s, \pi'(s)\right) = \sum_a \pi'(a|s) q_\pi(s,a)$$

$$= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + (1-\varepsilon) \max_a q_\pi(s,a)$$

$$v_{\pi(s)} = \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) - \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + \sum_a \pi(a|s) q_\pi(s,a)$$

$$= \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + (1-\varepsilon) \sum_a \frac{\pi(a|s) - \frac{\varepsilon}{|\mathcal{A}(s)|}}{1-\varepsilon} q_\pi(s,a)$$

π needs to be ε-soft

convex combination < max

On-policy MC control

We now know:

- if we start with any policy that takes any action with probability at least ε/|A| (ε-soft) *e.g. old ε-greedy policy*
- The new ε-greedy policy will be at least as good wrt the used q function

$$\pi' \geq \pi \,(\text{i.e.}, v_{\pi'}(s) \geq v_\pi(s), \text{ for all } s \in \mathcal{S})$$

Furthermore, it can be proven that the policy does not improve only if it is already optimal among ε-soft policies

So, with ε-greedy policies, the policy improvement step indeed either leads to policy improvement or the policy is already optimal for current q.

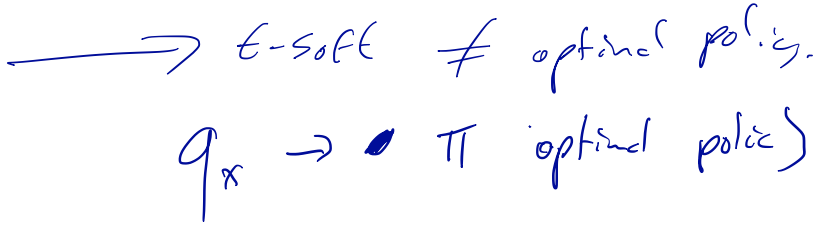
Thus, GPI will converge to the optimal ε-soft policy

Off-policy learning

We ‘only’ obtain the best ε-soft policies. Can we do better?

Use two policies:

- non-greedy behaviour policy b
- greedy target policy π



Data from a different policy than updated one: **off-policy**

Off-policy learning

First consider predicting v_π using data from b

Coverages assumption:

$$\pi(a|s) > 0 \rightarrow b(a|s) > 0$$

(always satisfied if b is ε-soft)

Off-policy learning

Consider trajectories

$$\tau_t = S_t, A_t, \ldots, S_T$$

We know the normal Monte Carlo approximation

$$v_b(s) = \mathbb{E}_b[G(\tau_t)|S_t = s, A_t \sim b] = \sum_{\tau_t} p(\tau_t|S_t = s, A_t \sim b)G(\tau_t)$$

$$\approx \sum_{i=1}^{n} \frac{1}{n} G(\tau^i)$$

where τ^i are sampled using b starting at state s

Off-policy learning

$$v_\pi(s) = \mathbb{E}[G(\tau_t)|S_t = s, A_t \sim \pi] = \sum_{\tau_t} p(\tau_t|S_t = s, A_t \sim \pi) G(\tau_t)$$

How to approximate this expectations using samples from π?

↳ average again

How about samples from b?

Importance sampling

Let's try importance weights

$$v_\pi(s) = \mathbb{E}[G(\tau_t)|S_t = s, A_t \sim \pi] = \sum_{\tau_t} p(\tau_t|S_t = s, A_t \sim \pi) G(\tau_t)$$

$$= \sum p(\tau_t | s, A_t \sim b) \frac{p(\tau_t | s, A_t \sim \pi)}{p(\tau_t | s, A_t \sim b)} G(\tau_t)$$

importance weights

$$= \sum_{i=1}^{n} \frac{1}{n} \frac{p(\tau_t | s, A_t \sim \pi)}{p(\tau_t | s, A_t \sim b)} G(\tau_t^i)$$

Importance sampling

Let's try importance weights

$$v_\pi(s) = \mathbb{E}[G(\tau_t)|S_t = s, A_t \sim \pi] = \sum_{\tau_t} p(\tau_t|S_t = s, A_t \sim \pi) G(\tau_t)$$

$$= \sum_{\tau_t} p(\tau_t|S_t = s, A_t \sim \pi) \frac{p(\tau_t|S_t = s, A_t \sim b)}{p(\tau_t|S_t = s, A_t \sim b)} G(\tau_t)$$

$$= \sum_{\tau_t} p(\tau_t|S_t = s, A_t \sim b) \frac{p(\tau_t|S_t = s, A_t \sim \pi)}{p(\tau_t|S_t = s, A_t \sim b)} G(\tau_t)$$

$$\approx \sum_{i=1}^{n} \frac{1}{n} \frac{p(\tau_t|S_t = s, A_t \sim \pi)}{p(\tau_t|S_t = s, A_t \sim b)} G(\tau^i)$$

τ^i → trajectory $\sim b$

Approximating an expectation based on b, so sample τ^i from b!

Importance sampling

Let's look at the importance weights

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi\left(A_k|S_k\right) p\left(S_{k+1}|S_k, A_k\right)}{\prod_{k=t}^{T-1} b\left(A_k|S_k\right) p\left(S_{k+1}|S_k, A_k\right)}$$

Importance sampling

Let's look at the importance weights

$$\rho_{t:T-1} \doteq \frac{\prod_{k=t}^{T-1} \pi\left(A_k|S_k\right) p\left(S_{k+1}|S_k, A_k\right)}{\prod_{k=t}^{T-1} b\left(A_k|S_k\right) p\left(S_{k+1}|S_k, A_k\right)} = \prod_{k=t}^{T-1} \frac{\pi\left(A_k|S_k\right)}{b\left(A_k|S_k\right)}$$

Luckily, the weights don't depend on the transition dynamics!

Use this to re-weight trajectories following visits to s

(first- or every-visit)

$$\tilde{V}_\pi(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|}$$

set of time steps with visits to s

Importance sampling

Weighted importance sampling is an alternative

ordinary i.s.

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{|\mathcal{T}(s)|}$$

weighted i.s.

$$V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}}$$

Importance sampling

Weighted importance sampling is an alternative

	Extreme case: one trajectory	ρ=10 or ρ=0.1
ordinary i.s. $V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\lvert\mathcal{T}(s)\rvert}$	ρ/1 can be >1 or <1 ⇒good on average	Estimate very high or very low! High variance!
weighted i.s. $V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}}$		

Importance sampling

Weighted importance sampling is an alternative

	Extreme case: one trajectory	ρ=10 or ρ=0.1
ordinary i.s. $V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\lvert\mathcal{T}(s)\rvert}$	ρ/1 can be >1 or <1 ⇒good on average	Estimate very high or very low! High variance!
weighted i.s. $V(s) \doteq \frac{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}}$	ρ / ρ =1, always ⇒biased	Estimates close together Low variance!

Off-policy Monte Carlo control

greedy?
→ $\pi(a|s) = 1$ if $a = \arg\max$
$\pi(a|s) = 0$ if $a \neq \arg\max$

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
$Q(s,a) \in \mathbb{R}$ (arbitrarily)
$C(s,a) \leftarrow 0$
$\pi(s) \leftarrow \text{argmax}_a\, Q(s,a)$ (with ties broken consistently)

Loop forever (for each episode):
$b \leftarrow$ any soft policy
Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
$G \leftarrow 0$
$W \leftarrow 1$
Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
$G \leftarrow \gamma G + R_{t+1}$
total weight $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
evaluation $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
improvement $\pi(S_t) \leftarrow \text{argmax}_a\, Q(S_t, a)$ (with ties broken consistently)
If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode) stop! (why?)
weight $W \leftarrow W \frac{1}{b(A_t|S_t)}$

$$\rho = \prod_t \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$$

Figure: Sutton&Barto; RL:AI

Importance sampling

First-visit with ordinary importance sampling is unbiased

(expected value equal to true value function)

Every-visit MC or using weighted importance sampling biased

- But weighted i.s. has much lower variance, so typically lower errors
- Weighed i.s. typically preferred
- Every-visit MC easier to implement
- Bias falls asymptotically to 0 as number of samples increases

Again, incremental implementation are possible

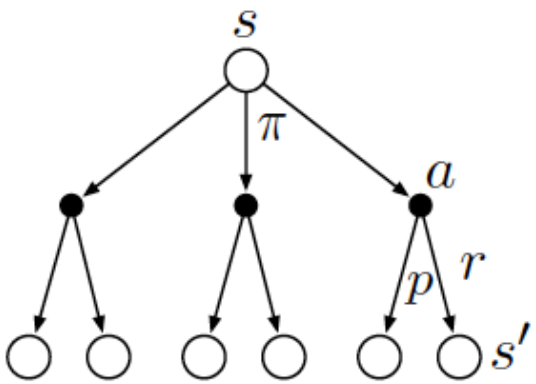
- Bit more complicated for weighted i.s., see book

Off-policy learning

On-policy	Off-policy
Simpler	More complex
Specific case	More general (we can have b=π)
Often converges faster	Often large variance or slow convergence
Only data gathered with current policy	Can reuse data, use data from other source
Generally needs non-greedy policy	Allows greedy target policy

So far

Dynamic programming



Need successor distribution

Uses structure of value function

Monte Carlo

Needs samples only

Unbiased updates possible

Ignores structure

High variance, especially with long updates

Limits of Monte Carlo

Remember the definition of the value function

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

Suggests learning value function by updating in direction of return G (Constant-α MC):

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

Target

but:

- Only know G when episode is over. What if we have a continuing task?
- Not using consistency between V(s) and V(s')

Consistency

Consider these episodes

$A, 0, B, 0$
$B, 1$
$B, 1$
$B, 1$

$B, 1$
$B, 1$
$B, 1$
$B, 0$

What are the following according to batch constant-α MC?
(Batch MC assumes these transitions will occur over and over)

$V(B) = \frac{3}{4}$

$V(A) = 0$

Example 6.4a from Sutton & Barto, RL:AI

Consistency

Previous answer ignores relationship between V(A) and V(B)

Exploit this consistency!

Consistency

Previous answer ignores relationship between V(A) and V(B)

Exploit this consistency!

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\
&= \mathbb{E}_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \middle| S_t = s \right] \\
&= \mathbb{E}_\pi \left[R_{t+1} + \gamma \mathbb{E}_\pi [G_{t+1}] \middle| S_t = s \right] \\
&= \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(s') \right] \middle| S_t = s]
\end{aligned}
$$

Use as target?

New update rule

Consistency suggests following learning rule

$$V(S_t) \leftarrow V(S_t) + \alpha[\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{Target}} - V(S_t)]$$

Compare with MC update rule

$$V(S_t) \leftarrow V(S_t) + \alpha[\underbrace{G_t}_{\text{Target}} - V(S_t)]$$

New update rule

Consistency suggests following learning rule

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

Target

'Error' between two estimates
Temporal difference (TD) error δ

Compare with MC update rule

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

Target

TD(0) learning

Because of central role of TD error, called temporal-difference learning. Specific algorithm: TD(0) (other TD-algorithms exist)

Consider the following state sequences again

$A, 0, B, 0$
$B, 1$
$B, 1$
$B, 1$

$B, 1$
$B, 1$
$B, 1$
$B, 0$

What would be the answer of batch contant-α TD-0?

$V(B) = 3/4$

$V(A) = 3/4 \cdot \gamma$

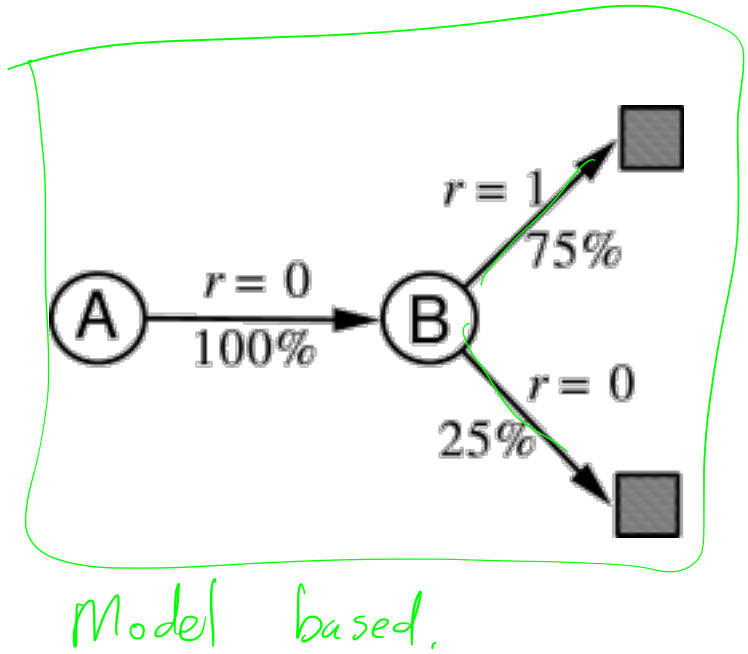
Hint: at end of episode, treat V(s') as 0

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

Example 6.4a from Sutton & Barto, RL:AI

Comparison

The answers from both TD-learning and MC make sense, but it seems reasonable that if we truly always go from A to B, the TD-answer will generalise better

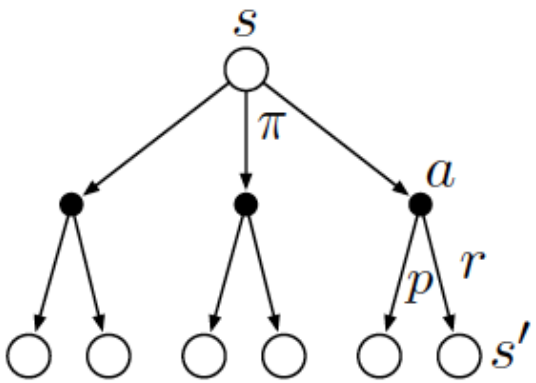


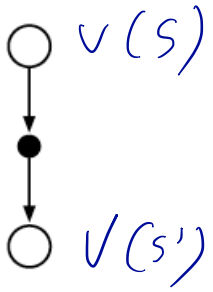
Example 6.4a from Sutton & Barto, RL:AI

Comparing back-ups so far

prediction → $v_\pi(s)$

control → $\pi(s) \rightarrow q_\pi(a, s)$





DP policy evaluation:
Need successor distribution
Uses Bellman equation

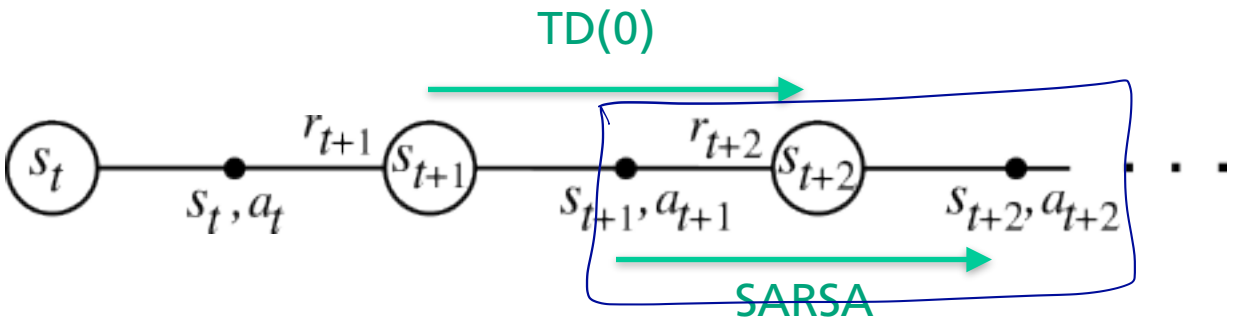
MC evaluation:
Need samples only
Ignores Bellman equation

TD evaluation:
Need samples only
Uses Bellman equation
'best of both worlds?'

Sarsa, on-policy TD control

Like before, learn a Q-function to select actions

Instead of going from state to a state, go from (state, action) to next (state, action)



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

Target

Temporal difference

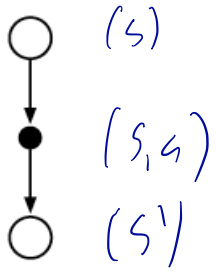
Sarsa, on-policy TD control

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ε-greedy)
  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)
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until s is terminal

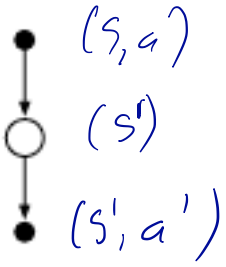
Policy needs to converge to greedy policy, for Sarsa to converge to q*

From each transition, the algorithm uses S,A,R,S',A', hence the name SARSA
Note: at termination Q(s',a') is defined to be 0

Back-up diagram for SARSA



TD(0)



Sarsa

Off-policy SARSA

Let's take another look at the SARSA update

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

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \mathbb{E}_{A_{t+1} \sim \pi} \left[Q(S_{t+1}, A_{t+1}) \mid S_{t+1} \right] - Q(S_t, A_t) \right]$$

$$\leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

Off-policy SARSA

Let's take another look at the SARSA update

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

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \mathbb{E}_\pi \left[Q(S_{t+1}, A_{t+1}) | S_{t+1} \right] - Q(S_t, A_t) \right]$$

$$\leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

Off-policy SARSA

Let's take another look at the SARSA update

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

$$Q_\pi(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \mathbb{E}_\pi \left[Q(S_{t+1}, A_{t+1}) | S_{t+1} \right] - Q(S_t, A_t) \right]$$

$$\leftarrow Q_\pi(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q_\pi(S_{t+1}, a) - Q_\pi(S_t, A_t) \right]$$

target

We can replace the Q value of the next state-action pair by its expectation *(Expected SARSA)*

This can be calculated for any policy π, also if different from behaviour policy b!

Behavior policy b used to sample *where* to update (S_t, A_t), but *not to calculate the target value*

Why no importance weights?

Why did we need importance weights in off-policy MC

$$v_\pi(s) = \mathbb{E}[G(\tau_t)|S_t = s, A_t \sim \pi]$$

$$\approx \frac{1}{N}\sum_{i=1}^{N} G(\tau^i) \quad \tau^i \sim p(\tau_t|S_t = s, A_t \sim \pi)$$

If we had trajectories starting with A_t drawn from the policy π, we would not need importance weights

We do not know what these trajectories would look like. So we need to use trajectories from a behaviour policy b, and correct with importance weights

In expected SARSA we only need $Q(S_{t+1}, a)$ for actions we haven't tried (from π), which is not a problem.

Why no importance weights?

Why did we need importance weights in off-policy MC

$$v_\pi(s) = \mathbb{E}[G(\tau_t)|S_t = s, A_t \sim \pi]$$

$$\approx \frac{1}{N}\sum_{i=1}^{N} G(\tau^i) \quad \underbrace{\tau^i \sim p(\tau_t|S_t = s, A_t \sim \pi)}_{???}$$

If we had trajectories starting with A_t drawn from the policy π, we would not need importance weights

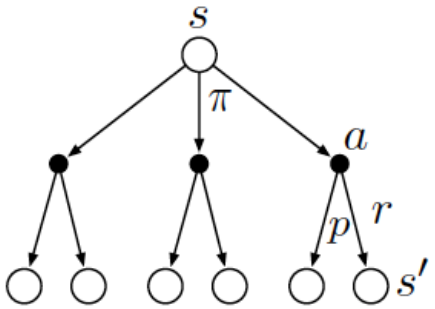
We do not know what these trajectories would look like. So we need to use trajectories from a behaviour policy b, and correct with importance weights

In expected SARSA we only need $Q(S_{t+1}, a)$ for actions we haven't tried (from π), which is not a problem.

Expected SARSA - special cases

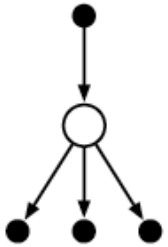
If $\pi=b$ (on-policy) it usually outperforms SARSA with same number of samples at some computational cost

π can be the greedy policy, in which case we obtain the famous Q-learning algorithm (more next week)



DP policy evaluation

multiple next states,
multiple next actions



Expected SARSA

One next state,
multiple next actions

SARSA

one next state,
one next action

Big picture so far

Finding optimal policies

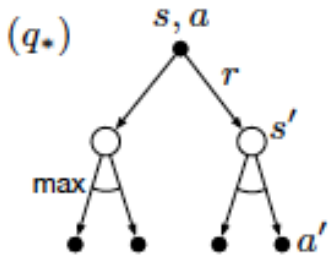
Known MDP dynamics

$$p(s', r|s, a)$$

Dynamic programming

Policy iteration

Value iteration



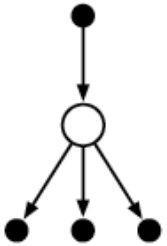
Only **data** from MDP: Reinforcement learning

$$D = \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}_{i=1\ldots N}$$

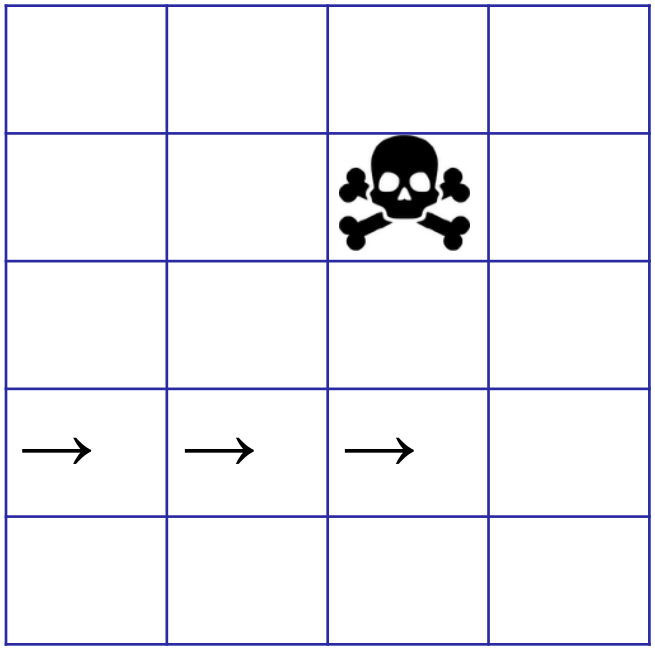
Monte-Carlo

Temporal Difference

(Expected) SARSA



Bigger picture: MC and TD

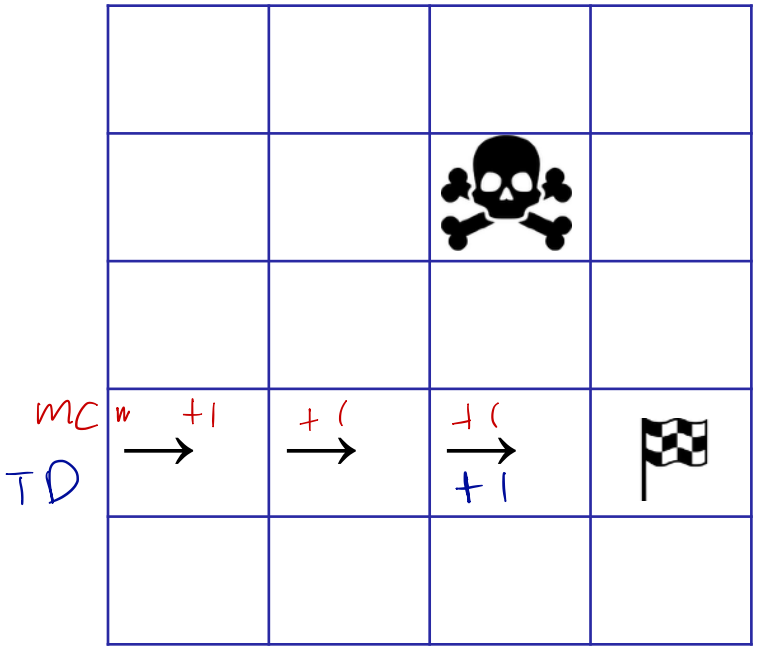


r = 1 to reach goal,
skull ends episode with r=0

Which states get updated by MC?

Which states get updated by TD?

Bigger picture: MC and TD

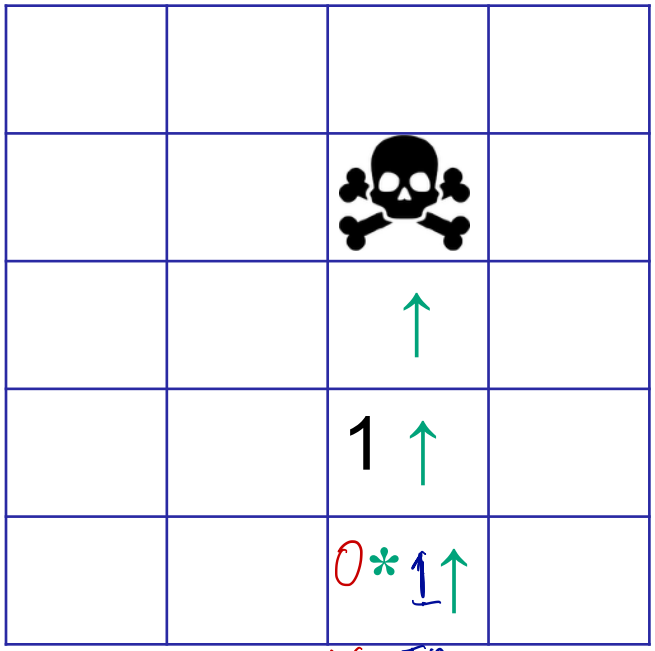


r = 1 to reach goal,
skull ends episode with r=0

Which states get updated by MC?

Which states get updated by TD?

Bigger picture: MC and TD



r = 1 to reach goal,
skull ends episode with r=0

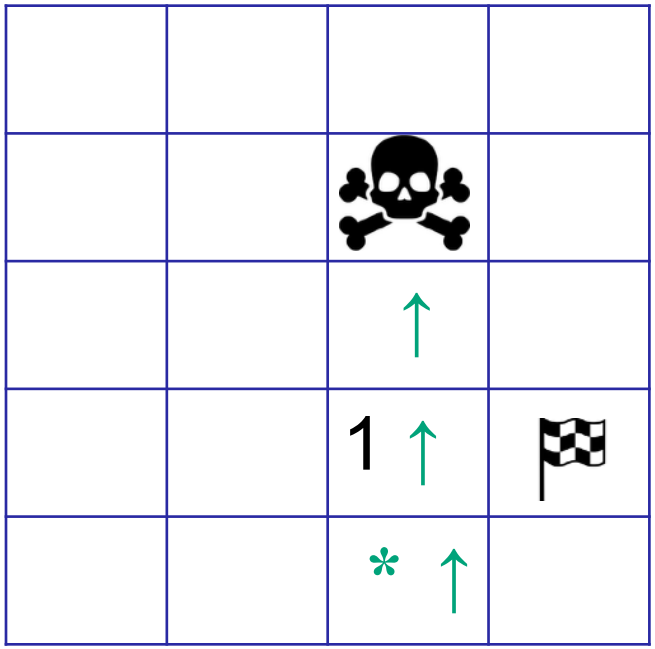
In first episode, we learned value -1

How is state * updated in second episode?

By TD(0)?

By MC?

Bigger picture: MC and TD



r = 1 to reach goal,
skull ends episode with r=0

In first episode, we learned value -1

How is state * updated in second episode?

By TD(0)?

By MC?

Bigger picture: MC and TD

TD(0) and MC both have advantages

- MC can quickly back-up from a single episode
- TD(0) can exploit learned value at intermediate states

Feedback?

h.c.vanhoof@uva.nl

Or anonymous at:
https://forms.gle/a8tYbeKNCnF8gftz6