Temporal difference learning

Herke van Hoof

Last lecture

Monte-Carlo approach to learning value functions

- Store observed returns from states or state-action pairs
- Estimate v or q function by taking average of the returns
- Need exploration:

  Exploring starts

  On-policy learning with a soft policy

  Off-policy learning with sort behaviour policy & greedy target policy

Last lecture: properties

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

Incremental implementation are possible

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

Big picture so far

Finding optimal policies

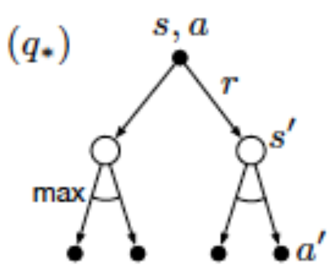
Known MDP dynamics

$$p\left(s', r | s, a\right)$$

Dynamic programming

Policy iteration

Value iteration



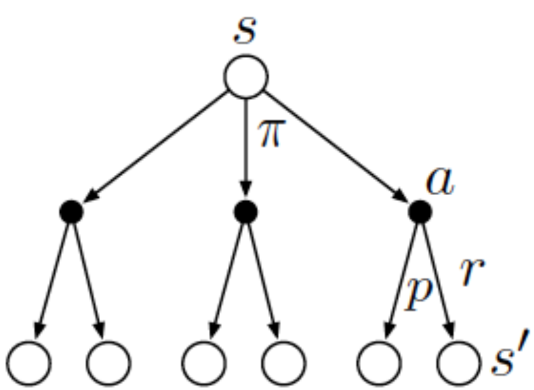
Only **data** from MDP: Reinforcement learning

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

Monte-Carlo

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

$$\begin{array}{ll} A, 0, B, 0 & B, 1 \\ B, 1 & B, 1 \\ B, 1 & B, 1 \\ B, 1 & B, 0 \end{array}$$

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

$$V(B)$$

$$V(A)$$

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!

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

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 (constant-α)

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

New update rule

Consistency suggests following learning rule

$$V(S_t) \leftarrow V(S_t) + \alpha[\underbrace{\underbrace{R_{t+1} + \gamma V(S_{t+1})}_{\text{Target}} - V(S_t)}_{\text{'Error' between two estimates; Temporal difference (TD) error } \delta}]$$

Compare with MC update rule (constant-α)

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

TD(0)

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

$A, 0, B, 0$ $\quad$ $B, 1$

$B, 1$ $\quad$ $B, 1$

$B, 1$ $\quad$ $B, 1$

$B, 1$ $\quad$ $B, 0$

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

$V(B)$

$V(A)$

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

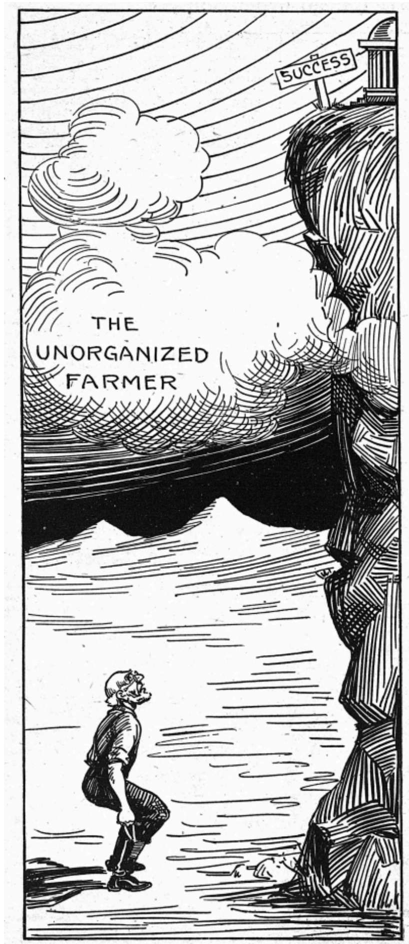
TD(0)

TD(0) improves a value function using a target depending on that same value function

Like ‘pulling yourself up by your bootstraps’?

Methods like DP or TD that use value function in the target referred to as ‘bootstrapping’

It’s more successful than this label implies ;)



Nonpartisan Leader, 1/24/1921.

TD(0)

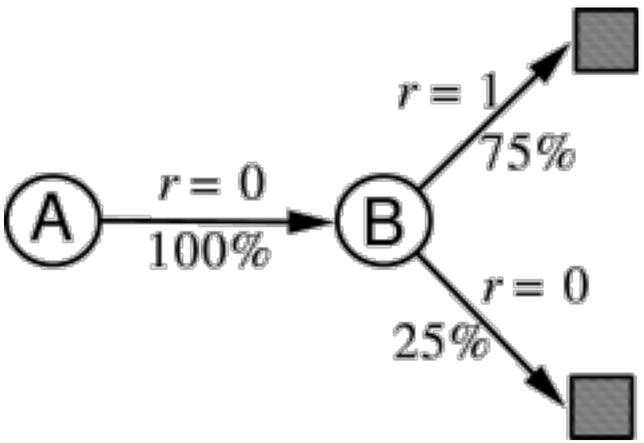
Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\mathit{terminal}) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until S is terminal

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

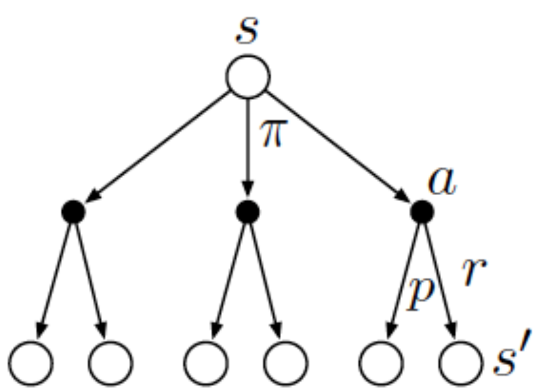
TD(0) properties

TD(0) is not unbiased, as the value estimates depend on the initial value function

For fixed π TD(0) converges to true V_π

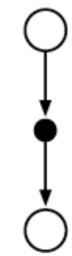
- With probability 1 when learning rate decreases appropriately
- In mean with fixed & sufficiently small learning rate

Comparing back-ups so far



DP policy evaluation:
Need successor distribution
Uses Bellman equation

MC evaluation:
Need samples only
Ignores Bellman equation

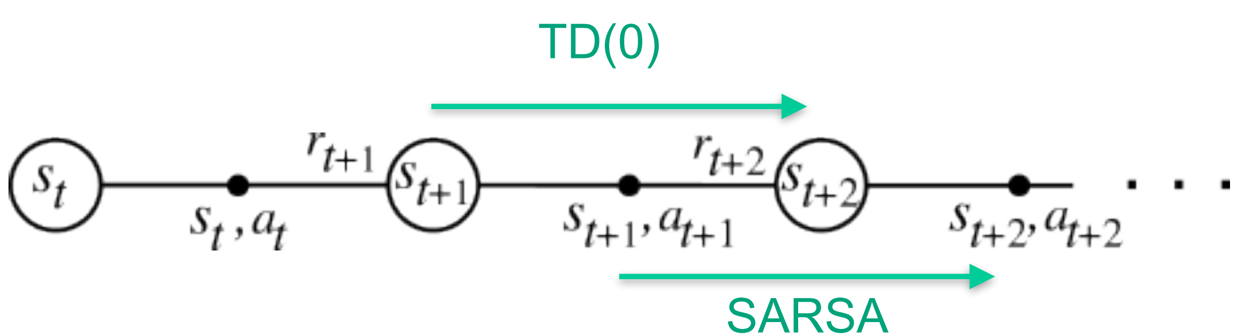


TD evaluation:
Need samples only
Uses Bellman equation
'best of both worlds?'

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)



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

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,
(e.g. ε made increasingly small)
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

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

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha \quad - Q\left(S_t, A_t\right)\right]$$

$$\leftarrow Q\left(S_t, A_t\right) + \alpha \left[\quad - Q\left(S_t, A_t\right)\right]$$

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

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

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

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

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?

We used importance weights in off-policy MC, why not here?

Let's re-visit off-policy MC

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha \rho_{t+1:T-1}\left[G_t - Q\left(S_t, A_t\right)\right]$$

The (S,A) pairs do not change the function being learned, so, it doesn't really matter which policy generated them.

Why no importance weights?

We used importance weights in off-policy MC, why not here?

Let's re-visit off-policy MC

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha\rho_{t+1:T-1}\left[G_t - Q\left(S_t, A_t\right)\right]$$

The (S,A) pairs do not change the function being learned, so, it doesn't really matter which policy generated them.

However, returns G define targets of the function being learned. These returns depend on behaviour policy b.

We can't know what would have been return under π. Only option is to use Gs that we have & correct with importance weights.

Why no importance weights?

We used importance weights in off-policy MC, why not here?

Let's re-visit off-policy MC

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \rho_{t+1:T-1} \left[G_t - Q(S_t, A_t) \right]$$

The (S,A) pairs do not change the function being learned, so, it doesn't really matter which policy generated them.

However, returns G define targets of the function being learned. These returns depend on behaviour policy b.

We can't know what would have been return under π. Only option is to use Gs that we have & correct with importance weights.

Why no importance weights?

Compare this to expected SARSA:

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

- Changing blue SA pairs still doesn't change learned function

Why no importance weights?

Compare this to expected SARSA:

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

- Changing blue SA pairs still doesn't change learned function
- We care about the target being appropriate for the Q function we want to learn. It depends on the red SA pair

Why no importance weights?

Compare this to expected SARSA:

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

- Changing blue SA pairs still doesn't change learned function
- We care about the target being appropriate for the Q function we want to learn. It depends on the red SA pair
- Could take A_{t+1} from behavior policy & correct with importance weights. Easier: take directly from target policy!

Why no importance weights?

Compare this to expected SARSA:

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

- Changing blue SA pairs still doesn't change learned function
- We care about the target being appropriate for the Q function we want to learn. It depends on the red SA pair
- Could take A_{t+1} from behavior policy & correct with importance weights. Easier: take directly from target policy!
- Easy to calculate target for any alternative A_{t+1}

Why no importance weights?

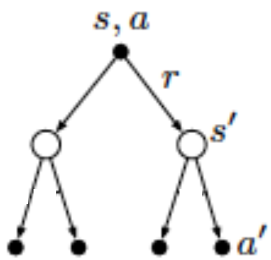
In short:

If we know what the value-function target is under π, use it. No importance weights necessary.

If we don't know what the value function target under π is, use the target under b instead and correct using importance weights.

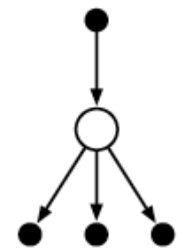
Expected SARSA - special cases

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



DP policy evaluation

multiple next states,
multiple next actions



Expected SARSA

One next state,
multiple next actions

SARSA

one next state,
one next action

Can we use the greedy policy for *π*? Yes, famous special case!

Q-learning, off-policy TD control

Special case of expected SARSA: use the greedy policy

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

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

Q-learning, off-policy TD control

Special case of expected SARSA: use the greedy policy

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

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

Q-learning converges to q* under 'usual conditions' (step-size schedule, guarantee each (s,a) continues to be visited)

Q-learning, off-policy TD control

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

Behavior policy, doesn't need to converge to greedy

Greedy policy in *target* instead of selecting a' from behaviour policy

Some clarifications

'Regular' Sarsa always uses the same policy for b and π
ϵ-greedy is a popular but not the only choice for b and π
Sarsa converges to Q^π of the best *exploring* policy

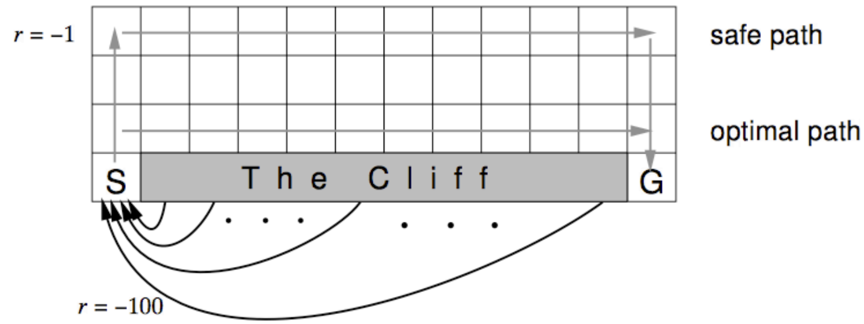
Q-learning always uses greedy policy for π with any b
Again, ϵ-greedy is a popular but not the only choice for b
Q-learning converges to $Q*$, Q function of best *overall* policy

Q-learning and Sarsa can converge to different values. We will see an example

Q-learning and Sarsa

Sarsa learns q_π of behaviour policy (e.g. ε-greedy)

Q-learning learns q^*



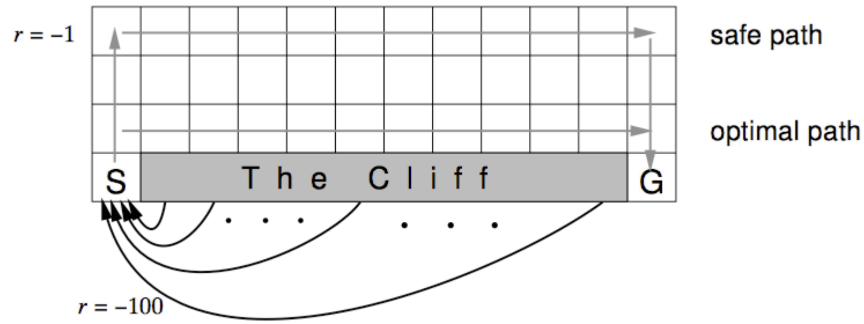
Performance	*Short path*	*Safe path*
With noise		
Deterministic		

Figure from Sutton & Barto, RL:AI

Q-learning and Sarsa

Sarsa learns q_π of behaviour policy (e.g. ε-greedy)

Q-learning learns q^*



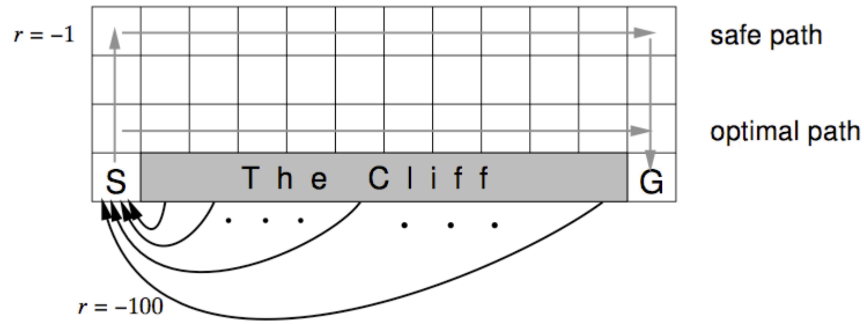
Performance	***Short path***	***Safe path***
With noise	falling off	'noisy' safe path
Deterministic		

Figure from Sutton & Barto, RL:AI

Q-learning and Sarsa

Sarsa learns q_π of behaviour policy (e.g. ε-greedy)

Q-learning learns q^*



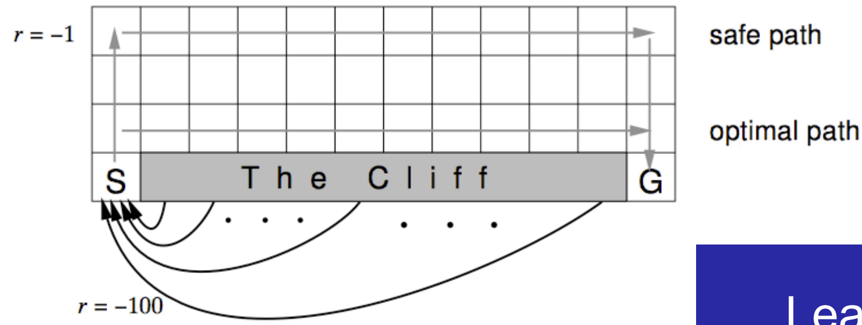
Performance	*Short path*	*Safe path*
With noise	falling off <	'noisy' safe path
Deterministic		

Figure from Sutton & Barto, RL:AI

Q-learning and Sarsa

Sarsa learns q_π of behaviour policy (e.g. ε-greedy)

Q-learning learns q^*



Learned by SARSA

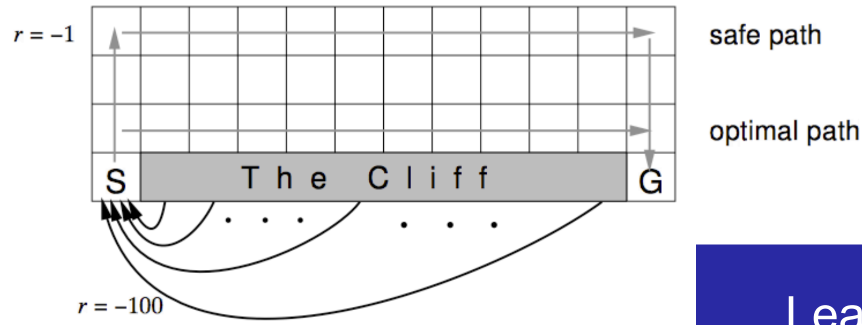
Performance	*Short path*	*Safe path*
With noise	falling off <	‘noisy’ safe path
Deterministic		

Figure from Sutton & Barto, RL:AI

Q-learning and Sarsa

Sarsa learns q_π of behaviour policy (e.g. ε-greedy)

Q-learning learns q^*



Learned by SARSA

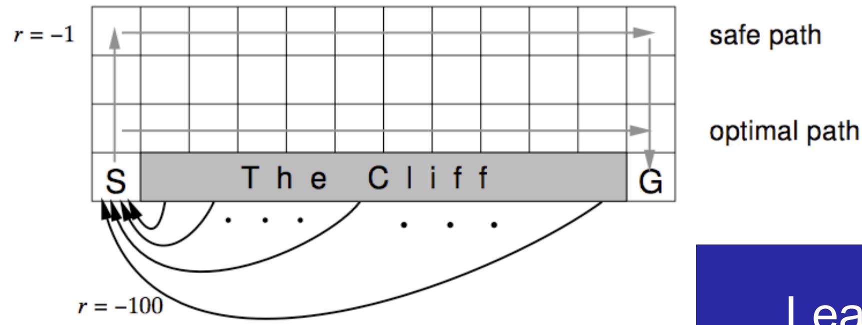
Performance	Short path	Safe path
With noise	falling off <	'noisy' safe path
Deterministic	optimal path	safe path

Figure from Sutton & Barto, RL:AI

Q-learning and Sarsa

Sarsa learns q_π of behaviour policy (e.g. ε-greedy)

Q-learning learns q^*



Learned by SARSA

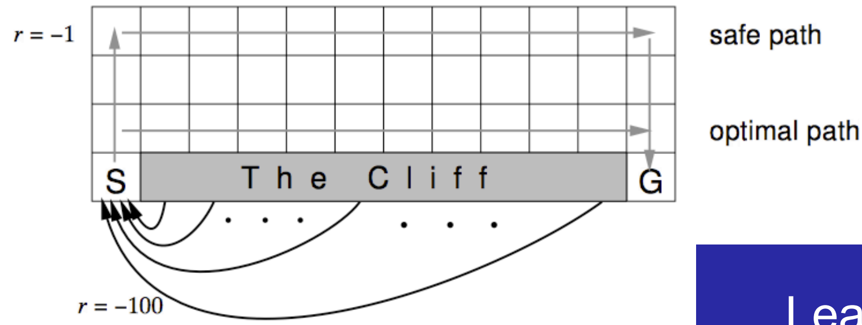
Performance	***Short path***		***Safe path***
With noise	falling off	<	'noisy' safe path
Deterministic	optimal path	>	safe path

Figure from Sutton & Barto, RL:AI

Q-learning and Sarsa

Sarsa learns q_π of behaviour policy (e.g. ε-greedy)

Q-learning learns q^*



Learned by SARSA

Performance	***Short path***		***Safe path***
With noise	falling off	<	'noisy' safe path
Deterministic	optimal path	>	safe path

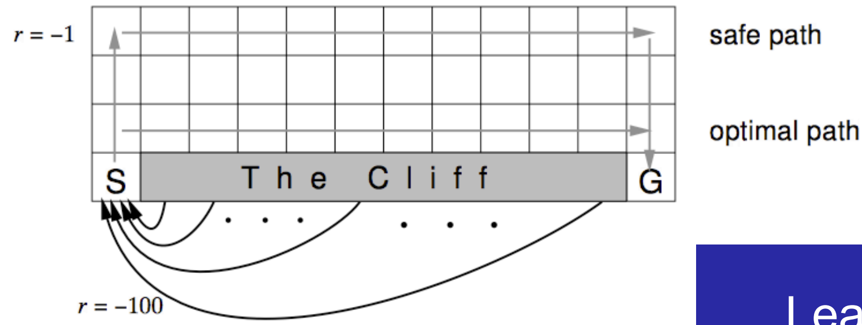
Greedy policy under q*
(Target policy Q-learning)

Figure from Sutton & Barto, RL:AI

Q-learning and Sarsa

Sarsa learns q_π of behaviour policy (e.g. ε-greedy)

Q-learning learns q^*



Learned by SARSA

Performance	***Short path***		***Safe path***
With noise	falling off	<	‘noisy’ safe path
	∧		∧
Deterministic	optimal path	>	safe path

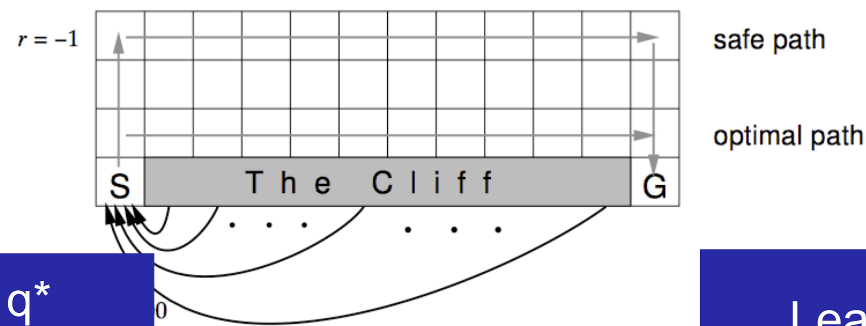
Greedy policy under q*
(Target policy Q-learning)

Figure from Sutton & Barto, RL:AI

Q-learning and Sarsa

Sarsa learns q_π of behaviour policy (e.g. ε-greedy)

Q-learning learns q^*



ϵ-greedy policy under q*
(e.g. behaviour policy q-learing)

Learned by SARSA

Performance	*Short path*		*Safe path*
With noise	falling off	<	'noisy' safe path
	∧		∧
Deterministic	optimal path	>	safe path

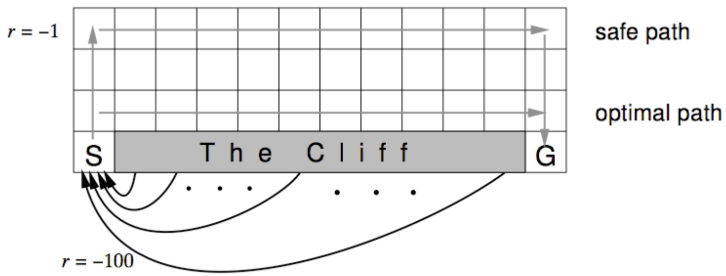
Greedy policy under q*
(Target policy Q-learning)

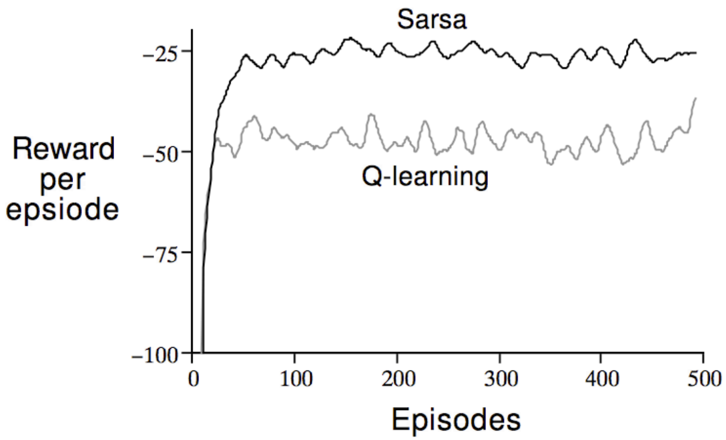
Figure from Sutton & Barto, RL:AI

Q-learning and Sarsa

Sarsa learns q_π of behaviour policy (e.g. ε-greedy)

Q-learning learns q^*





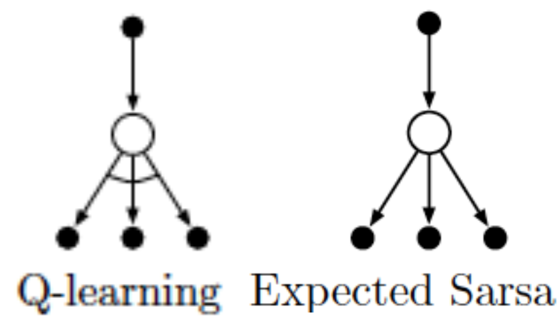
performance of ε-greedy behaviour policy with constant ε

Figure from Sutton & Barto, RL:AI

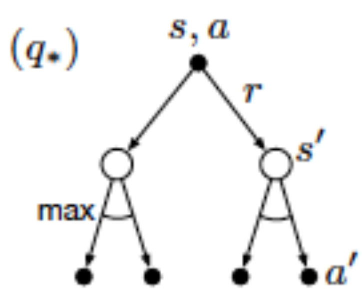
Back-up diagrams

SARSA

one next state,
one next action



One next state,
multiple next actions



DP q-value iteration

multiple next states,
multiple next actions

Big picture so far

Finding optimal policies

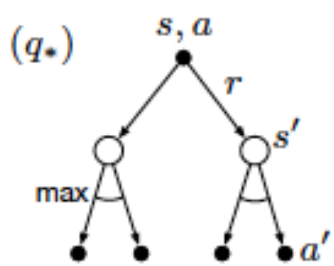
Known MDP dynamics

$$p\left(s', r \middle| s, a\right)$$

Dynamic programming

Policy iteration

Value iteration



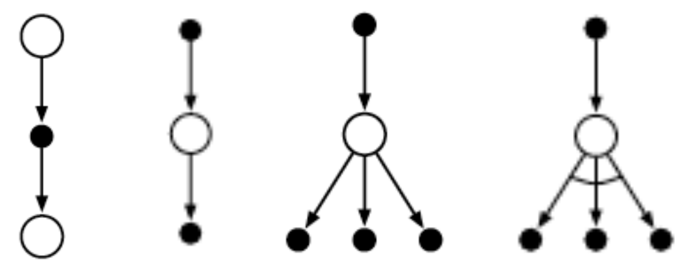
Only **data** from MDP: Reinforcement learning

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

Monte-Carlo

Temporal Difference

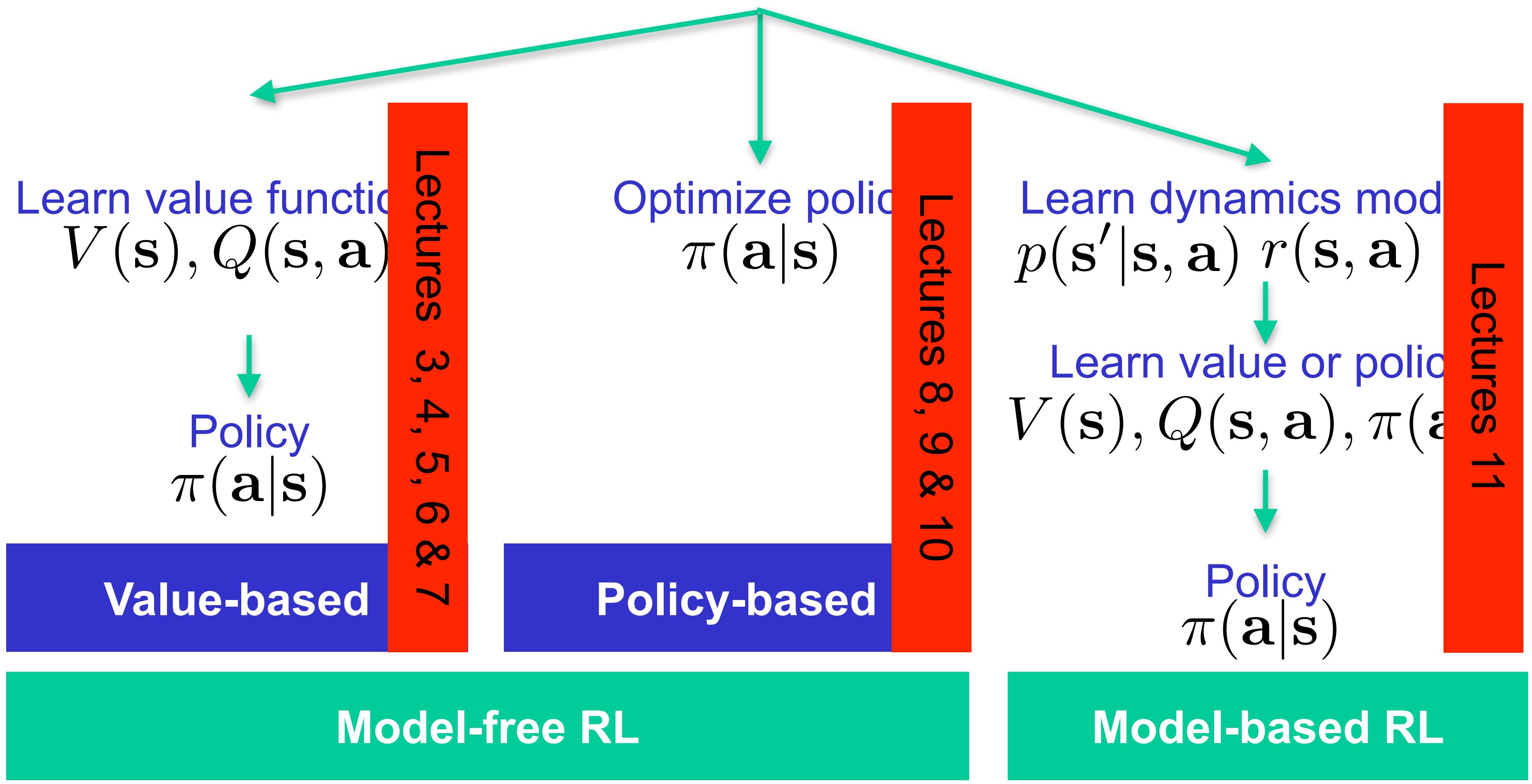
TD(0), (Expected) SARSA & Q-learning



All: directly learning value function without learning transition model

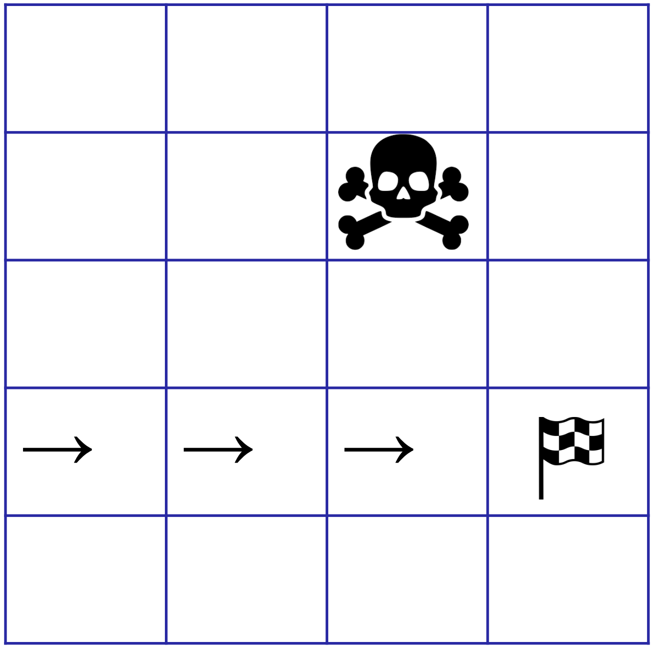
Big picture: How to learn policies

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



Thanks to Jan Peters

Bigger picture: MC and TD

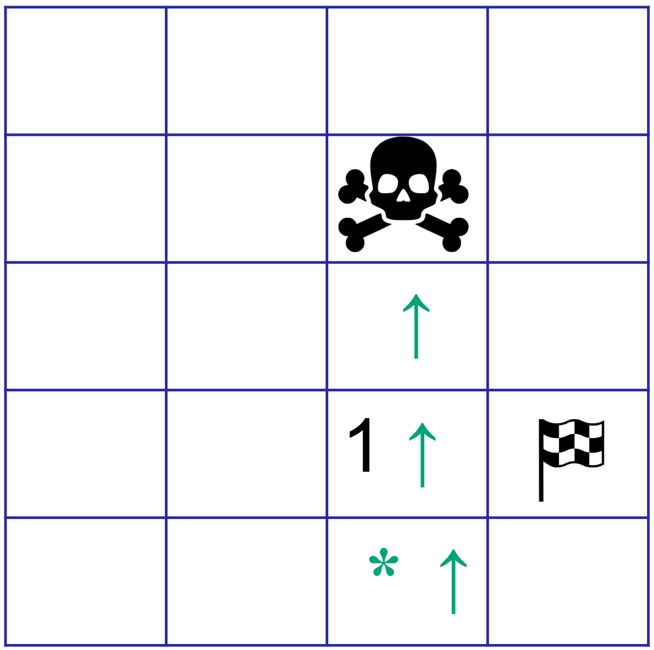


r = 1 to reach goal,
skull ends episode with r=0

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?

Which TD method to choose?

Classical on-policy vs off-policy - like discussed for MC method

On-policy (SARSA)	**Off-policy (Expected SARSA)**
Only need sampled action in update	Consider all actions per update (comp. cost)
Specific case	More general (we can have b=π)
Only data gathered with current policy	Can reuse data, use data from other source
Generally needs non-greedy policy	Allows greedy target policy (Q-learning)

Q-learning frequent special case as we're often interested in greedy target policies

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

What you should know

How do temporal difference (TD) methods compare to dynamic programming (DP) and Monte Carlo (MC) methods?

What are SARSA, expected SARSA and Q-learning and what are their properties?

Thanks for your attention!

Feedback?

h.c.vanhoof@uva.nl