Advanced TD methods

Herke van Hoof

Sarsa and off-policy learning

We'll take a closer look at the Sarsa algorithm.

Recap: Sarsa is an on-policy TD method for control (learns Q)

It is in the temporal difference family.

Sarsa is characterised by the shown update diagram and the below update equation

Sarsa

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

Off-policy SARSA

Let's take another look at the SARSA update

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

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

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

Why no importance weights?

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

First, realise there are two state-action pairs involved

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

- Changing blue SA pairs doesn't change learned function

Why no importance weights?

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

First, realise there are two state-action pairs involved

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

Target

- Changing blue SA pairs 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?

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

First, realise there are two state-action pairs involved

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

Target

- Changing blue SA pairs 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?

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

First, realise there are two state-action pairs involved

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

Target

- Changing blue SA pairs 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?

Why did we need importance weights in off-policy MC?

$$Q_\pi(s,a) = \mathbb{E}[G(\tau_t)|S_t = s, A_t = a, A_{t+1:T} \sim \pi]$$

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

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

Why no importance weights?

Why did we need importance weights in off-policy MC?

$$Q_\pi(s,a) = \mathbb{E}[G(\tau_t)|S_t = s, A_t = a, A_{t+1:T} \sim \pi]$$

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

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!

Why no importance weights?

Why did we need importance weights in off-policy MC?

$$Q_\pi(s,a) = \mathbb{E}[G(\tau_t)|S_t = s, A_t = a, A_{t+1:T} \sim \pi]$$

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

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!

To estimate expected G, only have trajectories from behaviour policy b. So use those, correct with importance weights

Why no importance weights?

Why did we need importance weights in off-policy MC?

$$Q_\pi(s,a) = \mathbb{E}[G(\tau_t)|S_t = s, A_t = a, A_{t+1:T} \sim \pi]$$

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

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

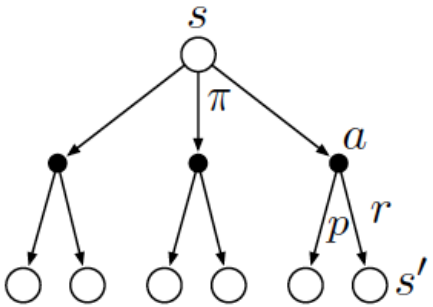
We do not know what these trajectories would look like!

To estimate expected G, only have trajectories from behaviour policy b. So use those, correct with importance weights

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

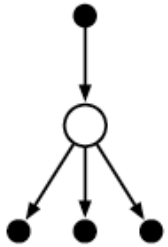
Expected SARSA - special cases

If $\pi=b$ (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

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

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

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 can be non-greedy

Greedy policy 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

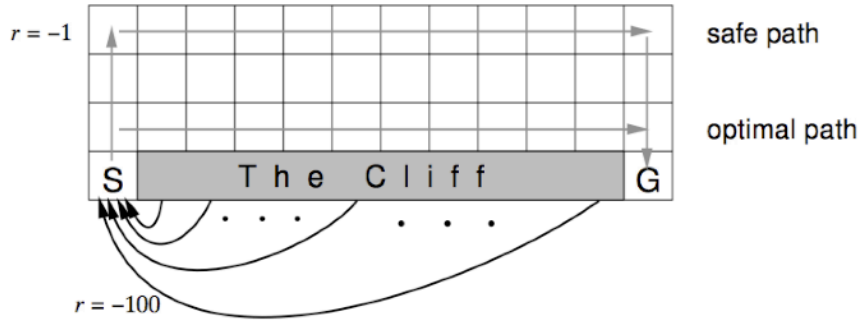
Q-learning always uses greedy policy for π with any b
Again, ϵ-greedy is a popular but not the only choice for b
Q-learning converges to Q^{*}, Q function of best *overall* policy

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^*



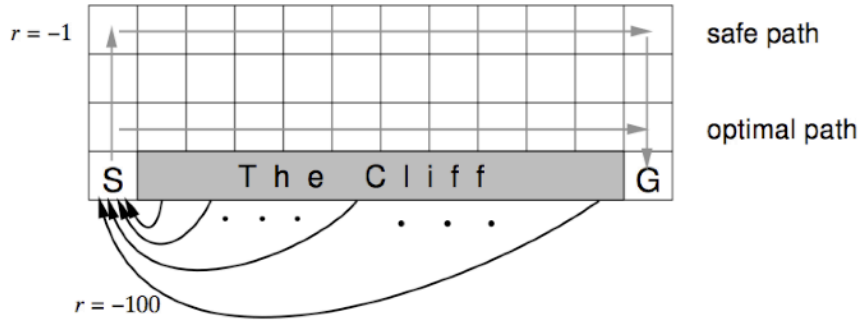
Performance	*Q-learning (optimal)*	*Sarsa (safe path)*
ε-greedy		
greedy		

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^*



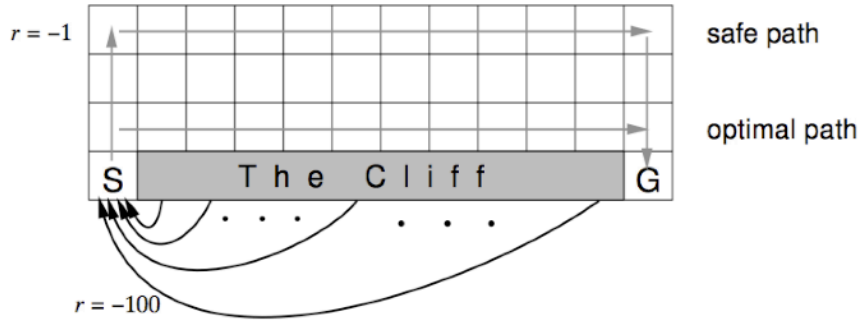
Performance	*Q-learning (optimal)*	*Sarsa (safe path)*
ε-greedy		‘noisy’ safe path
greedy	optimal path	

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^*



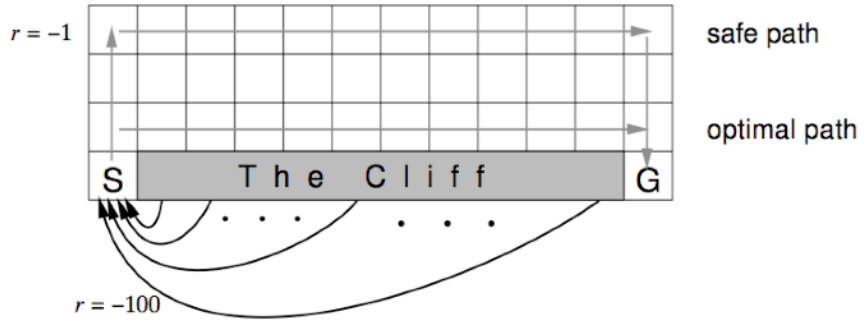
Performance	*Q-learning (optimal)*	*Sarsa (safe path)*
ε-greedy		'noisy' safe path
greedy	optimal path	

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^*



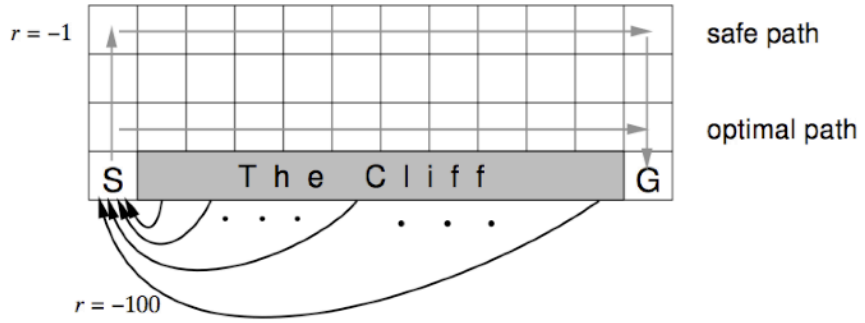
Performance	*Q-learning (optimal)*		*Sarsa (safe path)*
ε-greedy	falling off	<	'noisy' safe path
	∧		∧
greedy	optimal path		

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^*



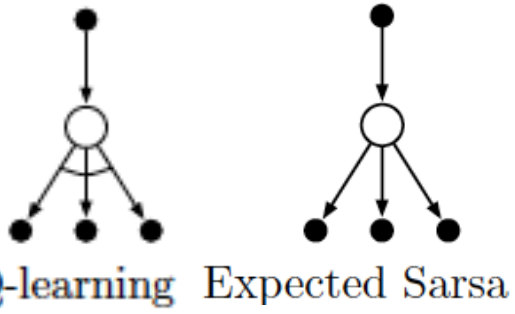
Performance	*Q-learning (optimal)*		*Sarsa (safe path)*
ε-greedy	falling off	<	'noisy' safe path
	∧		∧
greedy	optimal path	>	safe path

Figure from Sutton & Barto, RL:AI

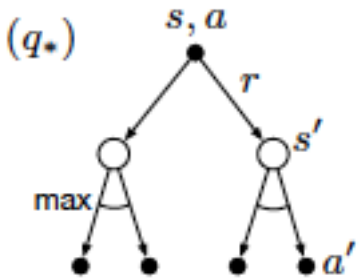
Back-up diagrams

SARSA

one next state,
one next action



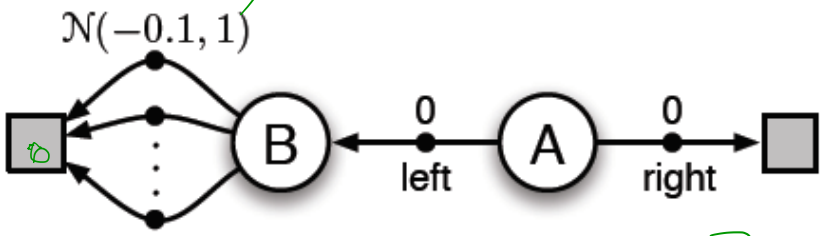
One next state,
multiple next actions



DP q-value iteration

multiple next states,
multiple next actions

Maximization bias



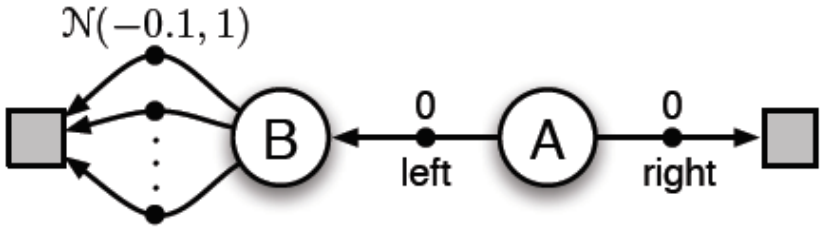
What is the true value of (A,left) and (B, a_1), (B, a_2), etc?

What will batch Q-learning estimate after trying the following:

A / left / 0	B / a_1 / 0.07
A / left / 0	B / a_1 / -1.85
A / left / 0	B / a_2 / 0.16
A / left / 0	B / a_3 / -0.23
A / right / 0	

Figure from Sutton & Barto, RL:AI

Maximization bias



What is the true value of (A,left) and (B, a_1), (B, a_2), etc?

What will batch Q-learning estimate after trying the following:

A / left / 0 B / a_1 / 0.07
A / left / 0 B / a_1 / -1.85
A / left / 0 B / a_2 / 0.16
A / left / 0 B / a_3 / -0.23
A / right / 0

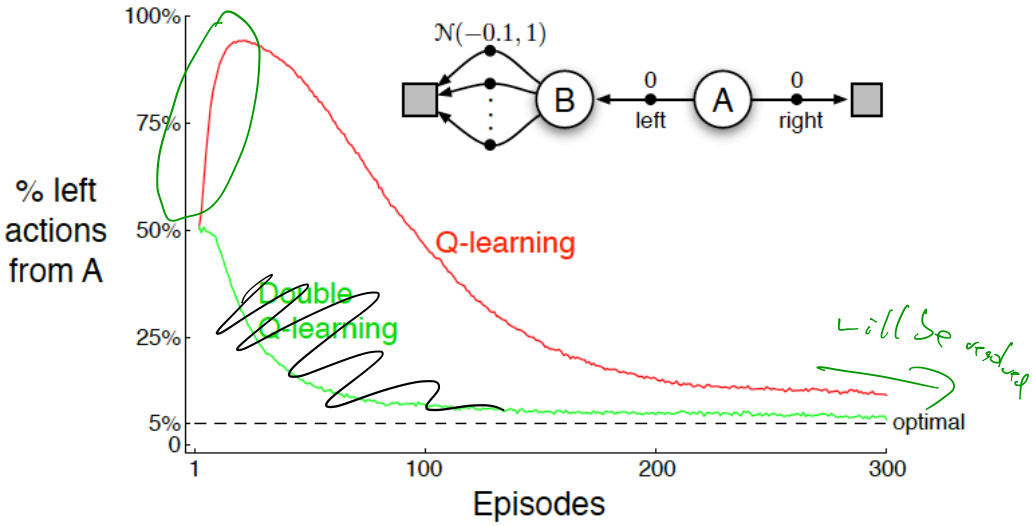


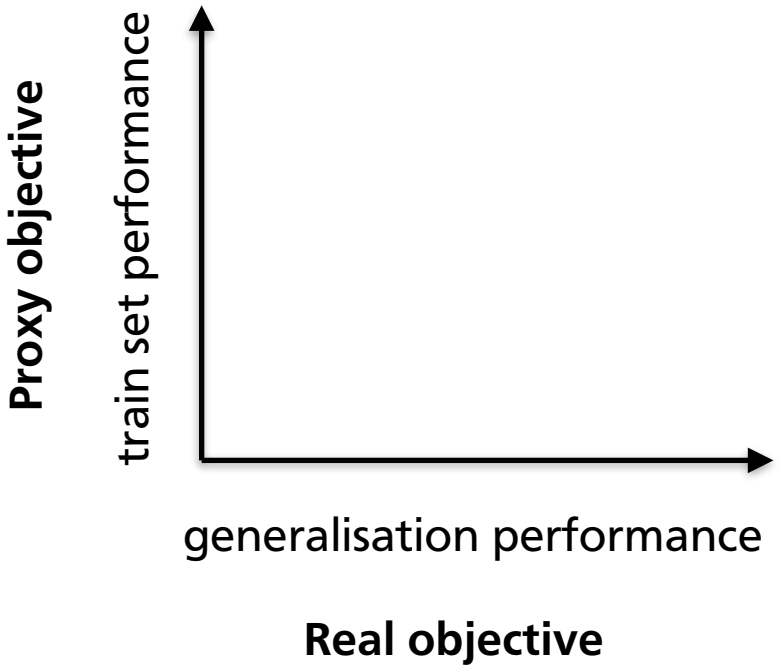
Figure from Sutton & Barto, RL:AI

Goodhart's Law

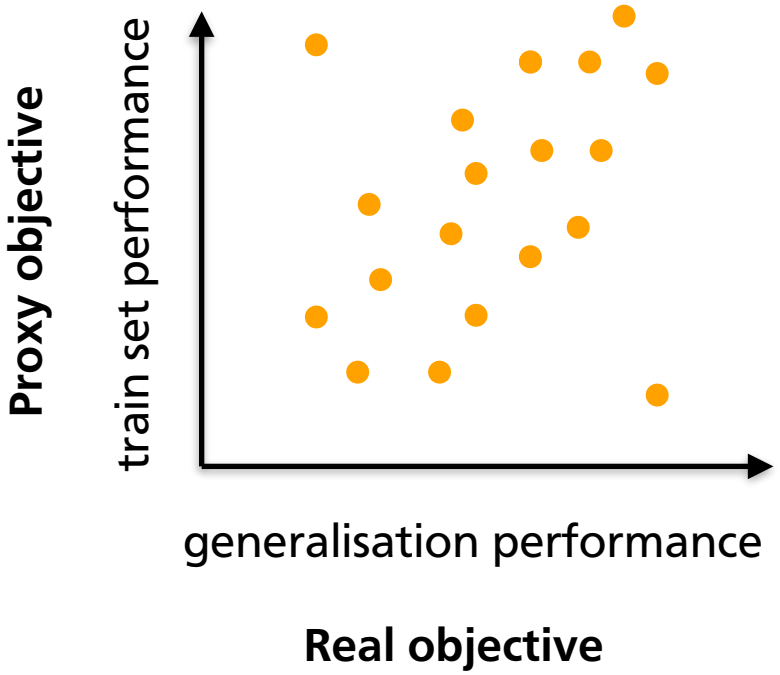
"When a measure becomes a target, it ceases to be a good measure"

Charles Goodhart

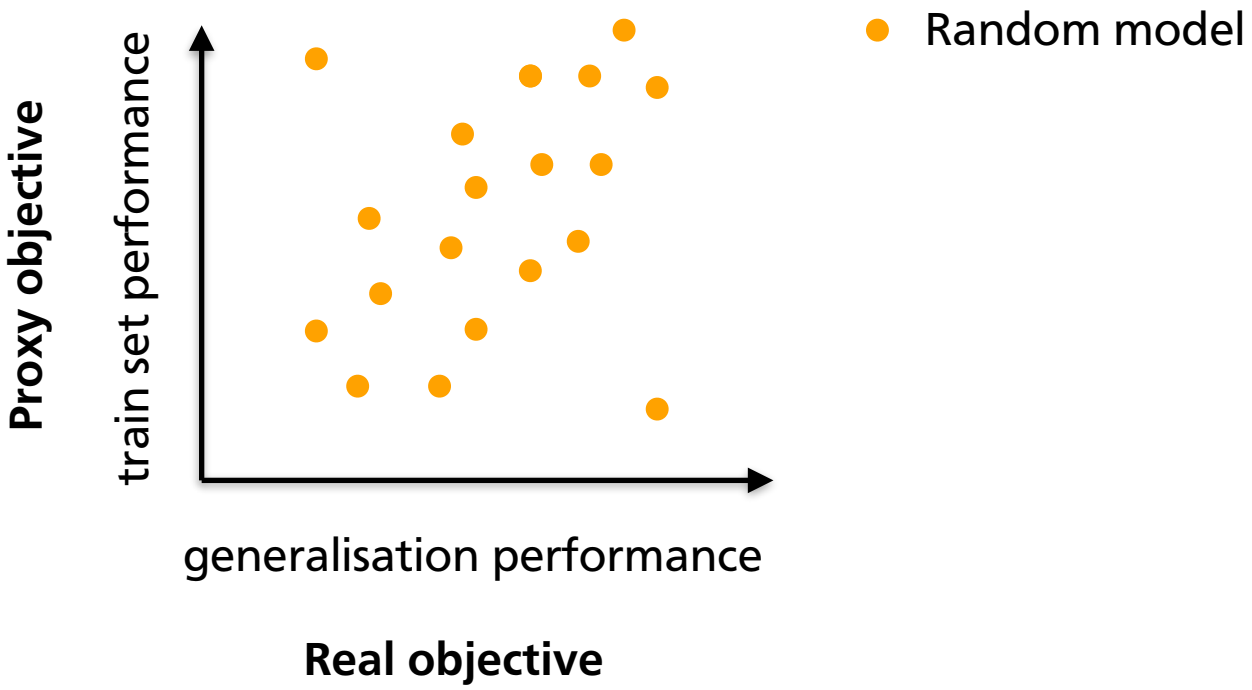
Goodhart's Law



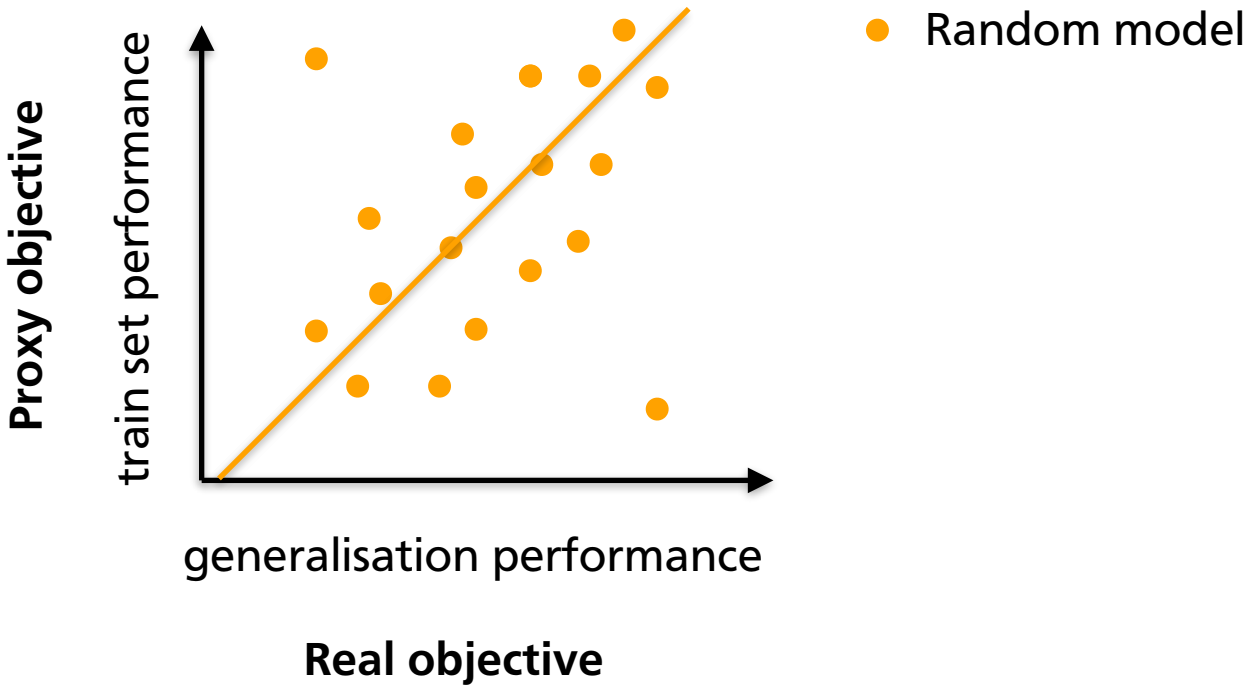
Goodhart's Law



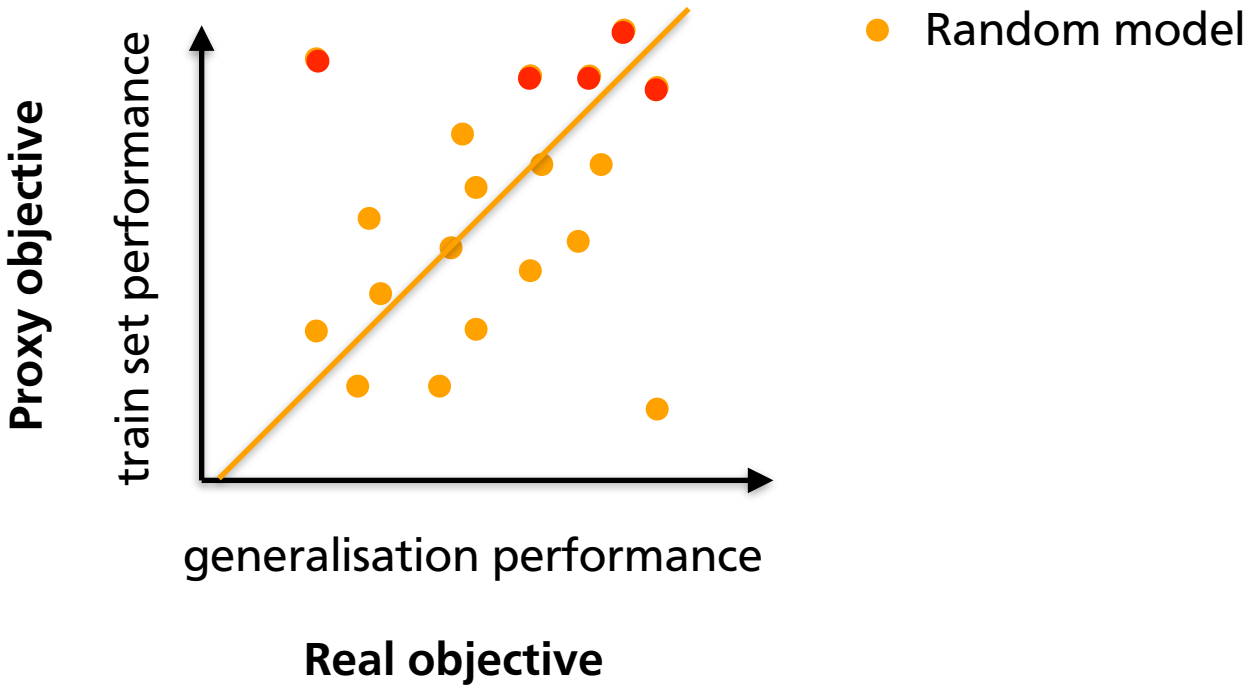
Goodhart's Law



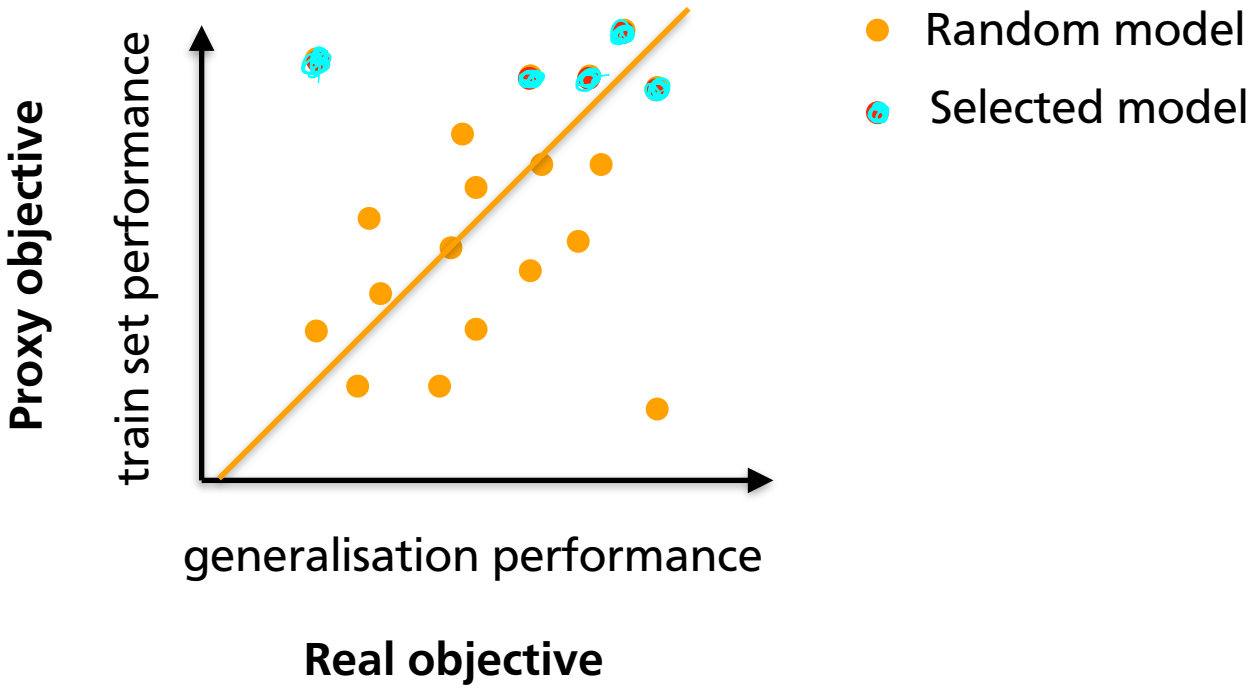
Goodhart's Law



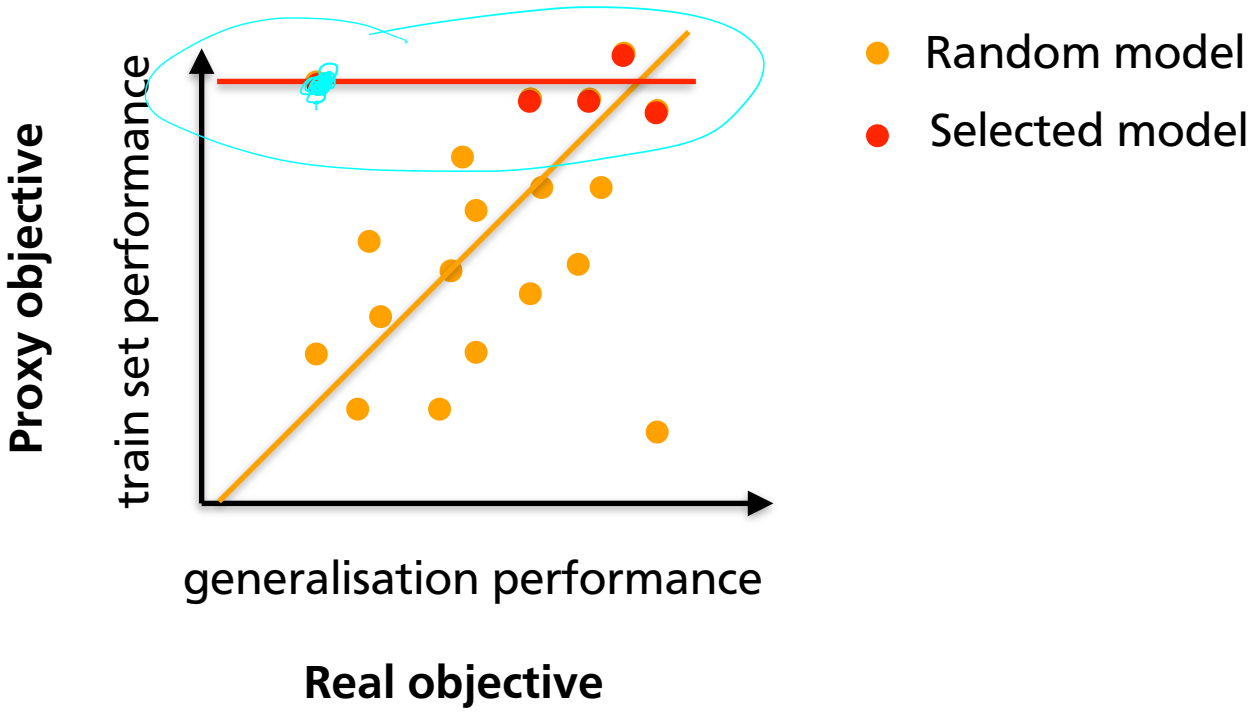
Goodhart's Law



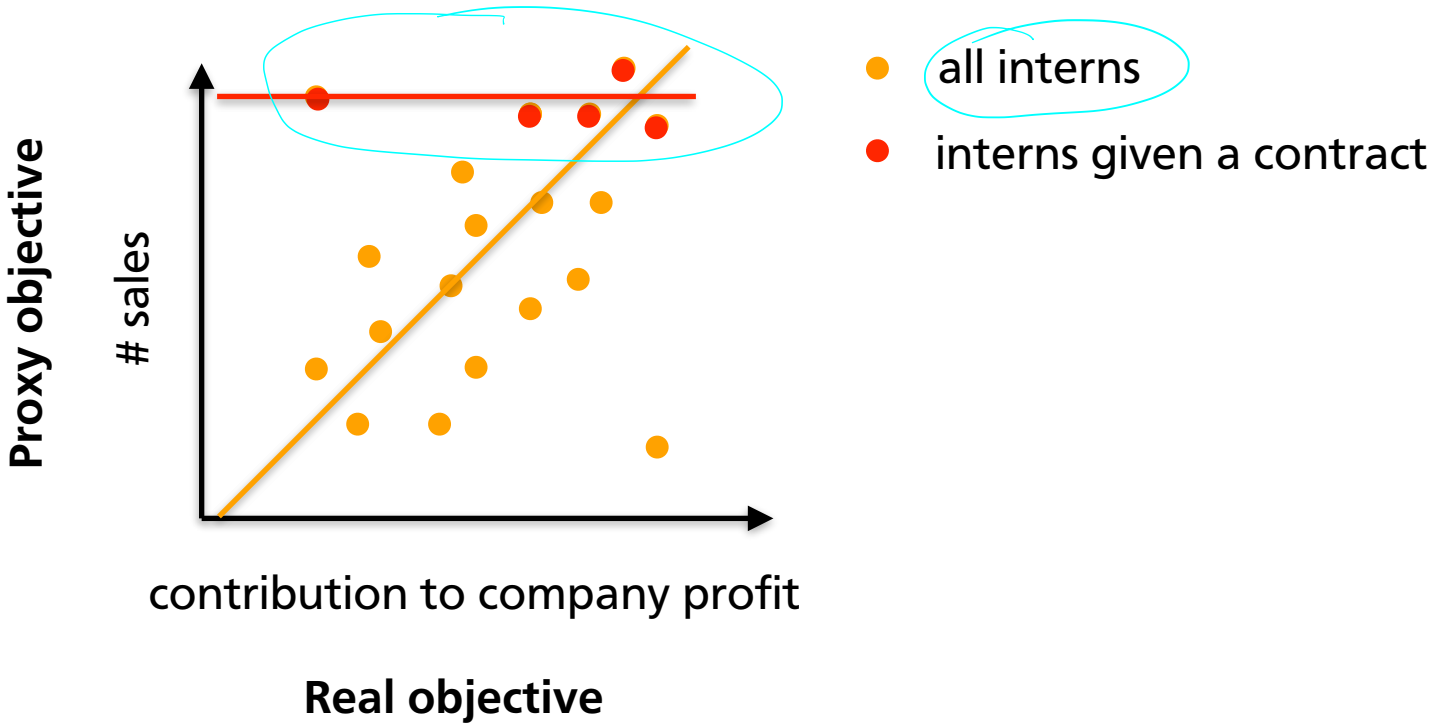
Goodhart's Law



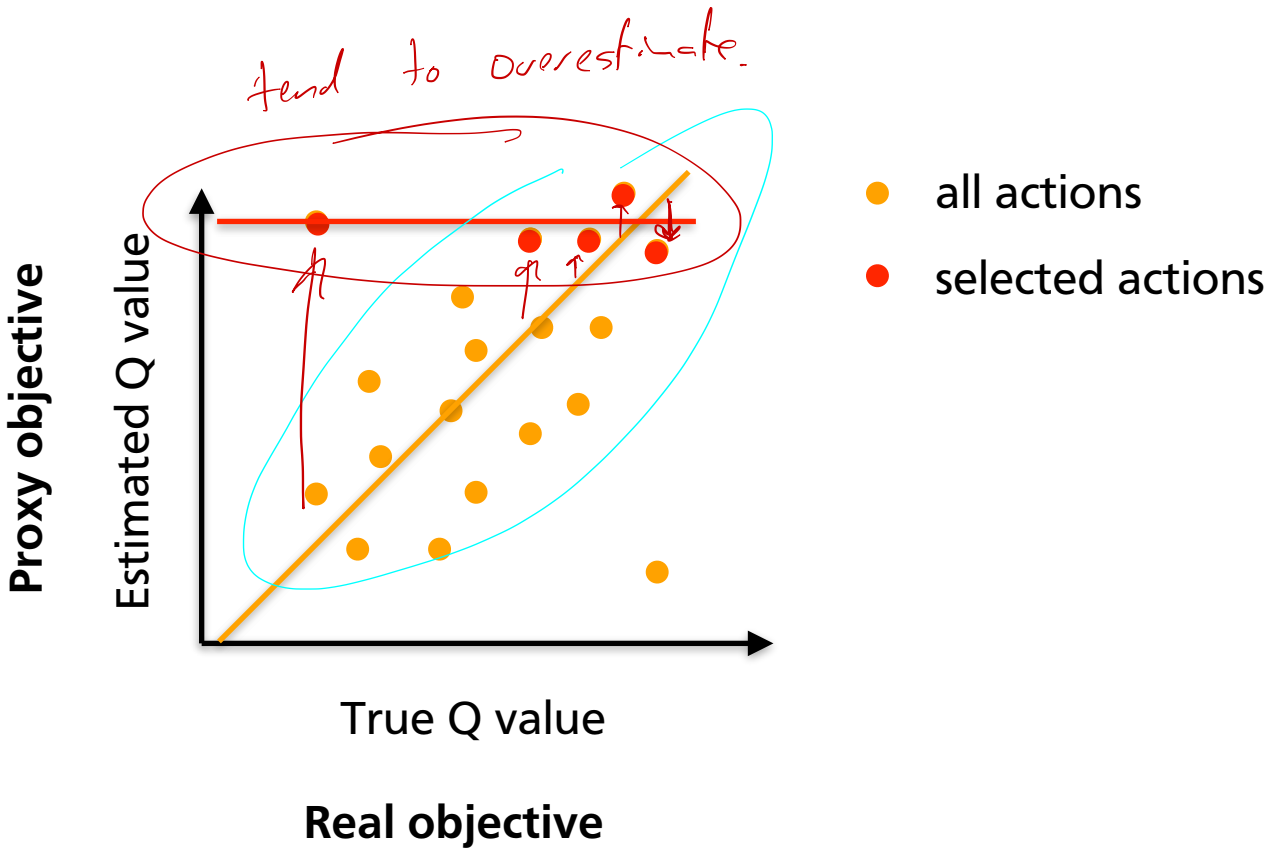
Goodhart's Law



Goodhart's Law



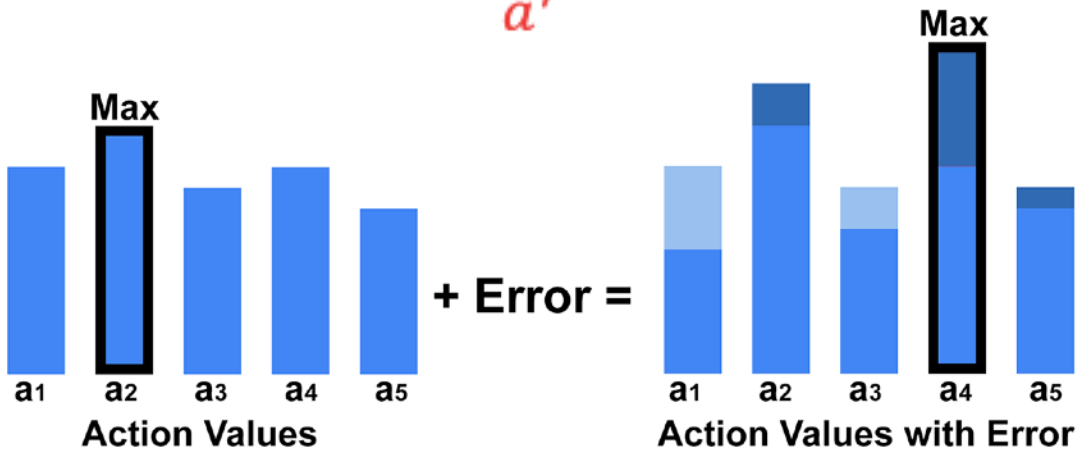
Goodhart's Law



Optimisation bias

The update rule of Q-learning with discrete actions:

$$Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$$



With errors ϵ overestimation is likely to occur (Thrun & Schwartz, 1993):

$$\mathrm{E}\left[\max_{a'}(Q(s',a') + \epsilon)\right] > \mathrm{E}\left[\max_{a'} Q(s',a')\right]$$

Maximization bias

In effect, the Q learning target is

$$R + \gamma \max_a Q(s, a)$$

$$R + \gamma Q(s', \arg\max_{a'} Q(s', a'))$$

Same function used to **select** actions and **evaluate** (s,a) pair

Result: overoptimistic values more likely to be used in updates

Maximization bias

In effect, the Q learning target is

$$R + \gamma Q(s', \arg\max_{a'} Q(s', a'))$$

Same function used to **select** actions and **evaluate** (s,a) pair

Result: overoptimistic values more likely to be used in updates

Remedy: learn two independent Q-functions Q_1, Q_2

$$Q_2(s,a) \leftarrow Q_2(s,a) + \alpha \left(R + \gamma Q_1(s', \arg\max_{a'} Q_2(s', a')) - Q_2(s,a) \right)$$

(Vice-versa to update Q_1.)

Actions where Q_2 is overoptimistic more likely to be used

Q_1 might be lower or higher than actual value, but **no inherent bias**

(every datapoint used to update Q_1 or Q_2 with 50% prob.)

Maximization bias

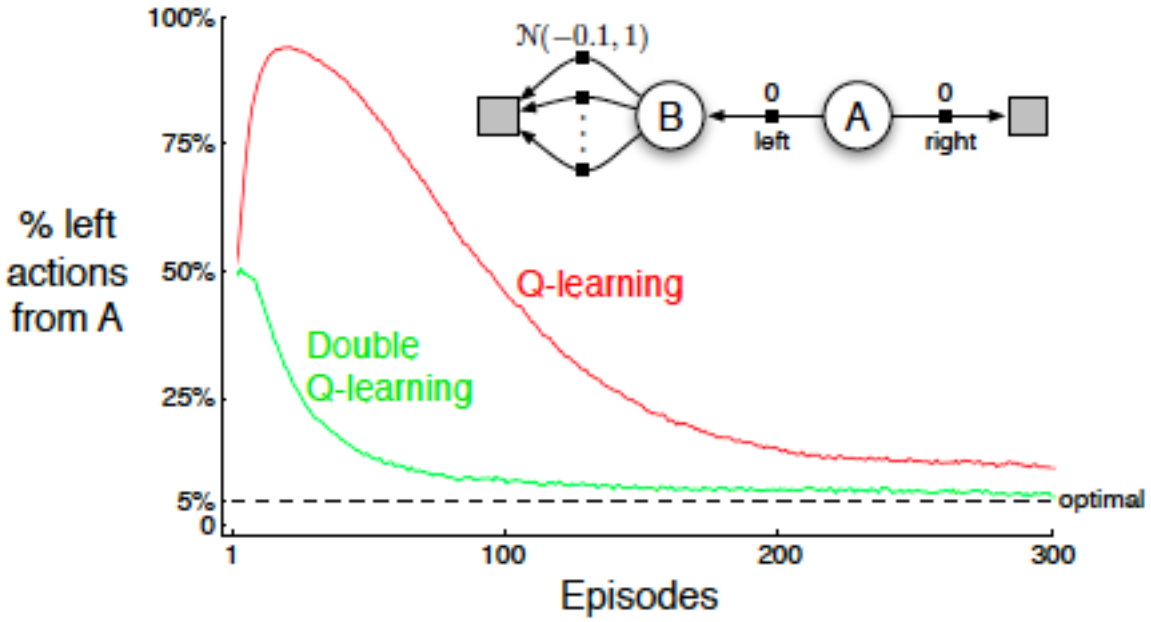


Figure from Sutton & Barto, RL:AI

Big picture so far

Finding optimal policies

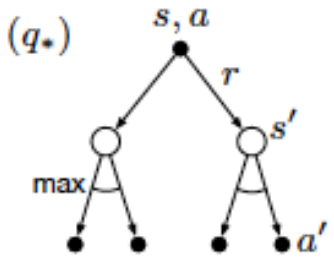
Known MDP dynamics

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

Dynamic programming

Policy iteration

Value iteration



Only **data** from MDP: Reinforcement learning

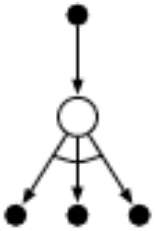
$$D = \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}_{i=1\ldots N}$$

Monte-Carlo

Temporal Difference

SARSA (on policy)

Q-learning (off policy)



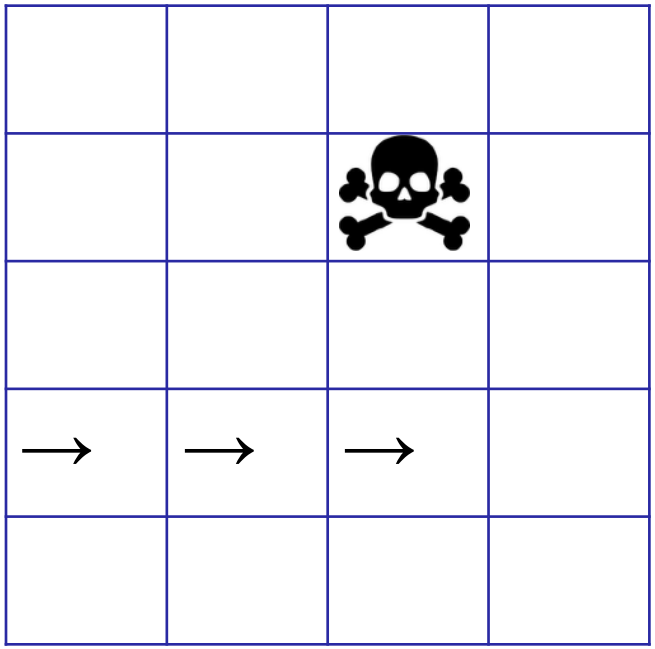
Which TD method to choose?

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

On-policy (SARSA)	**Off-policy (Expected SARSA)**
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 (Q-learning)

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

Bigger picture: MC and TD

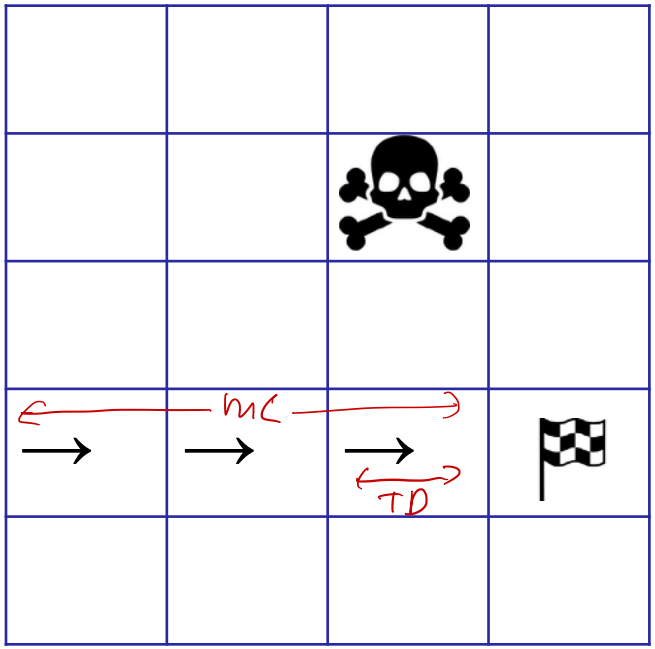


r = 1 to reach goal,
skull ends episode with r=0

Which states get updated by MC?

Which states get updated by TD?

Bigger picture: MC and TD

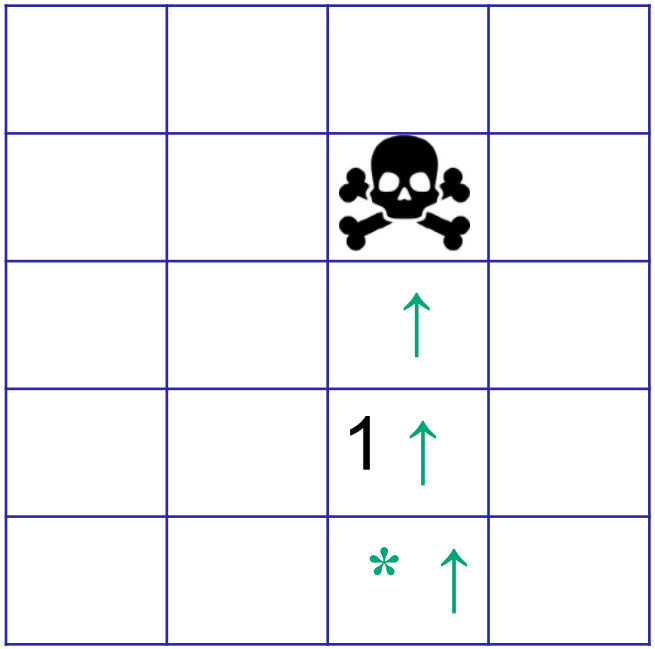


r = 1 to reach goal,
skull ends episode with r=0

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

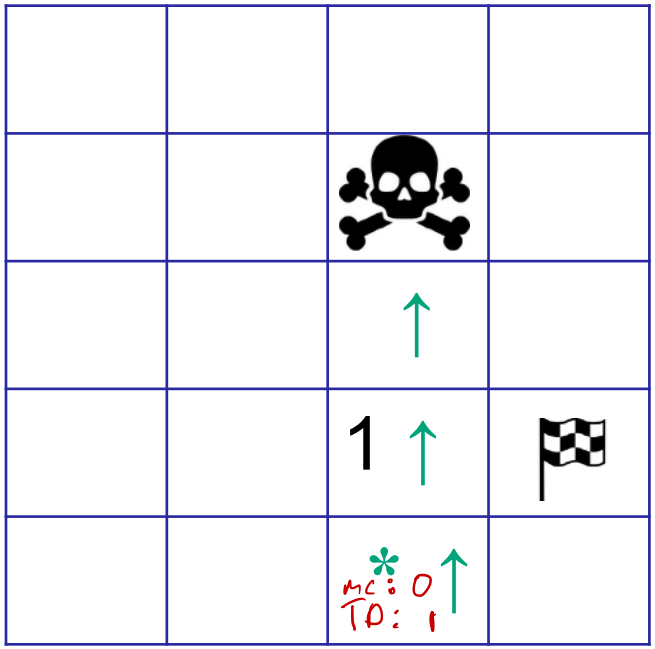
In first episode, we learned value -1

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

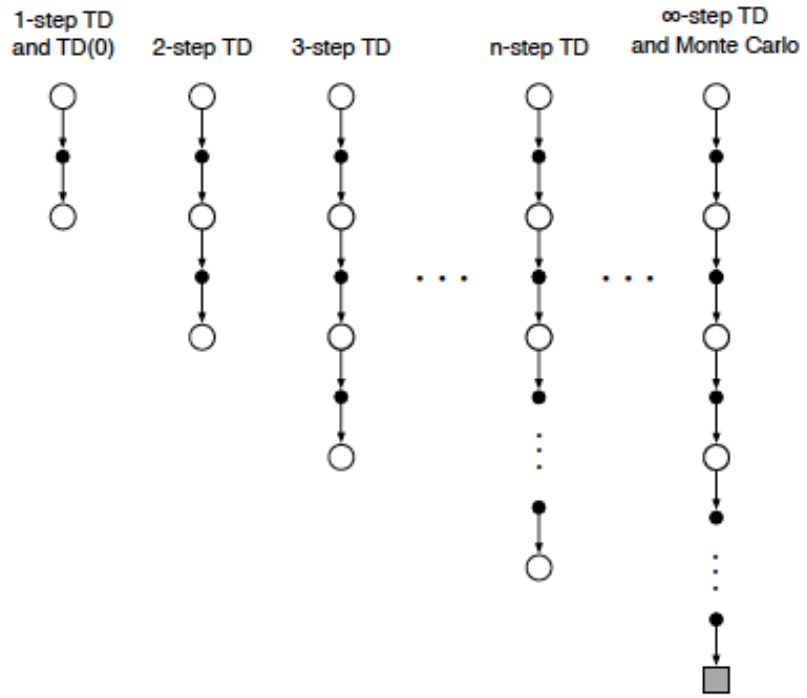
- MC can quickly back-up from a single episode
- TD(0) can exploit learned value at intermediate states

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

Let's try intermediate approach



N-step prediction

Let's generalise the notion of return:

- Complete return
  $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$

- One-step return
  $G_{t:t+1} \doteq R_{t+1} + \gamma V_t \left(S_{t+1}\right)$

N-step prediction

Let's generalise the notion of return:

- Complete return
  $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$

- One-step return
  $G_{t:t+1} \doteq R_{t+1} + \gamma V_t \left(S_{t+1}\right)$

- N-step return
  $G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1} \left(S_{t+n}\right)$

N-step prediction

$V_{new} \leftarrow V_{old} + \alpha(target - V_{old})$

$\leftarrow (1-\alpha) V_{old} + \alpha \cdot target$

Let's generalise the notion of return:

- Complete return
  $G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T$

- One-step return
  $G_{t:t+1} \doteq R_{t+1} + \gamma V_t(S_{t+1})$

- N-step return
  $G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n})$

We'll have to wait n steps before we know n-step return

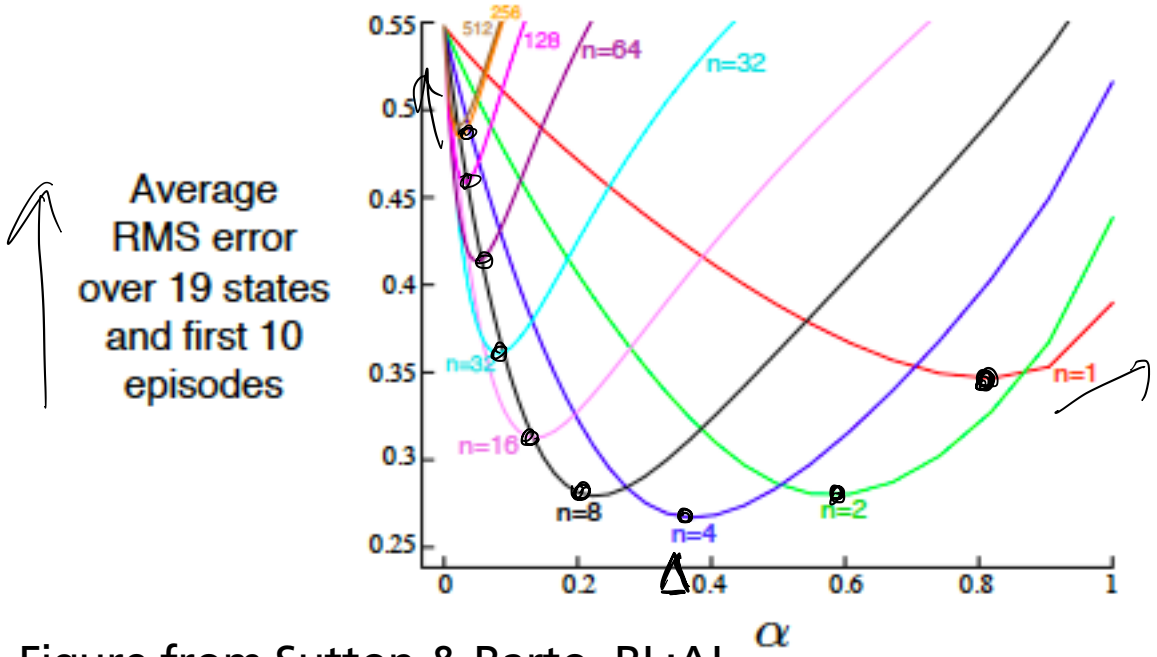
Then update:

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \left[G_{t:t+n} - V_{t+n-1}(S_t) \right], \quad 0 \leq t < T$$

N-step prediction

Which N works best depends on the problem

Example: 19-state random-walk



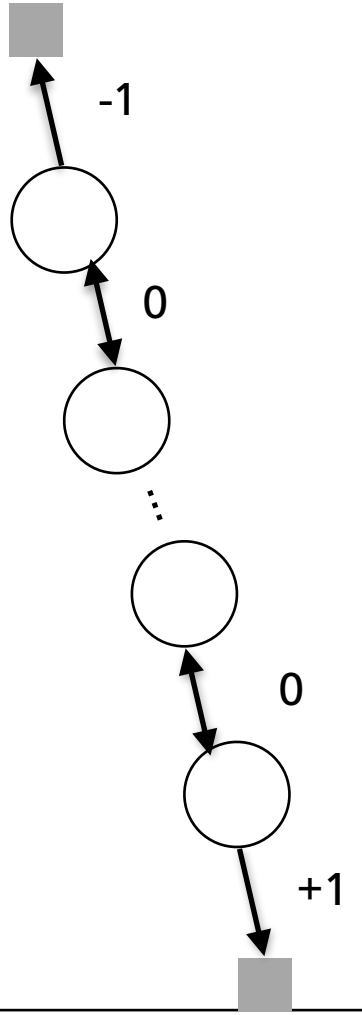


Figure from Sutton & Barto, RL:AI

N-step control

Like Sarsa, we can again take (s,a) to (s,a) transitions rather than state to state transitions

N-step Sarsa:

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

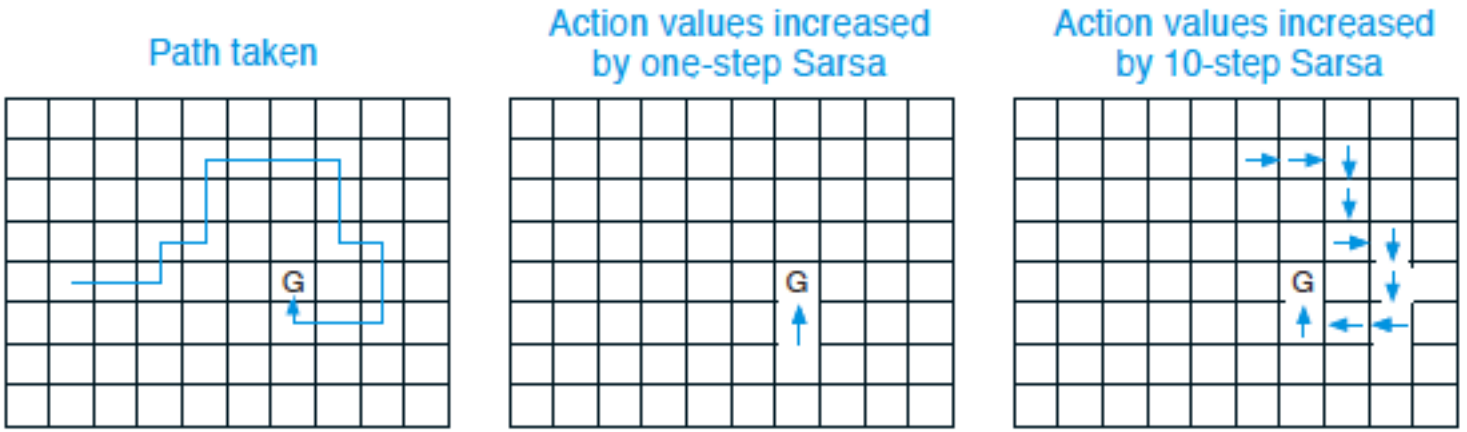


Figure from Sutton & Barto, RL:AI

Off-policy N-step TD

Say, we want to learn policy π but have data from policy b

Where $\pi(a|s) < b(a|s)$ the (s,a) pair will occur 'too often'

Where $\pi(a|s) > b(a|s)$ the (s,a) pair will not occur often enough

Off-policy N-step TD

$$\rho = \frac{p_\pi(\tau)}{p_b(\tau)} = \frac{\prod_t \pi(a_t|s_t) \cdot p(s'_t|s_t, a_t)}{\prod_t b(a_t|s_t) \cdot p(s'_t|s_t, a_t)}$$

Say, we want to learn policy π but have data from policy b

Where π(a|s) < b(a|s) the (s,a) pair will occur 'too often'

Where π(a|s) > b(a|s) the (s,a) pair will not occur often enough

Can correct with an importance sampling ratio:

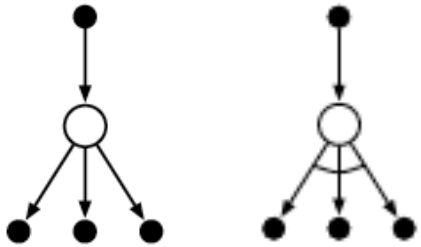
$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha \rho_{t:t+n-1} \left[G_{t:t+n} - V_{t+n-1}(S_t) \right]$$

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

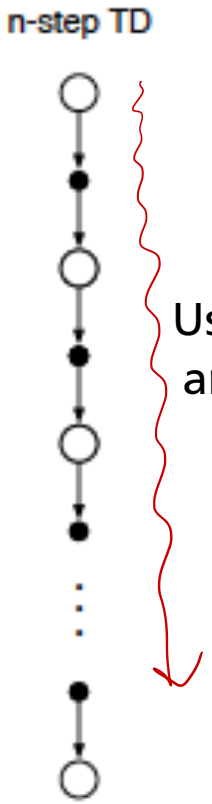
π(a|s) << b(a|s): Close to 0

π(a|s) > b(a|s): ρ > 1

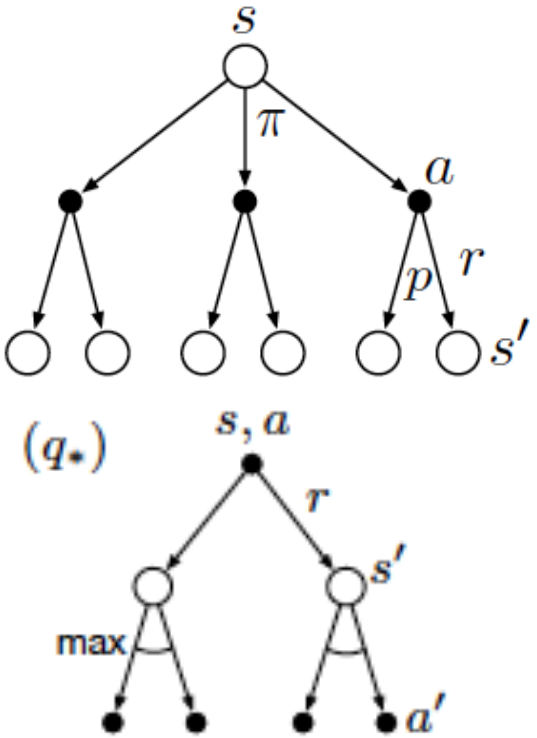
Can we use more information?



Uses the Q-value for actions that weren't chosen from s'

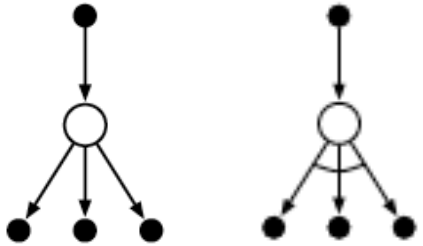


Uses visited states and actions from further in the future

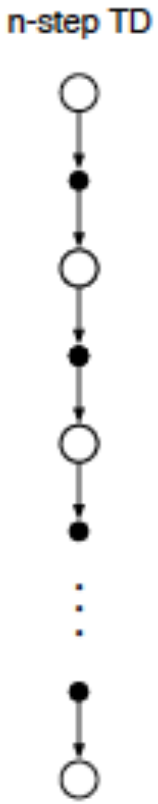


Uses states that would be visited by actions that weren't chosen

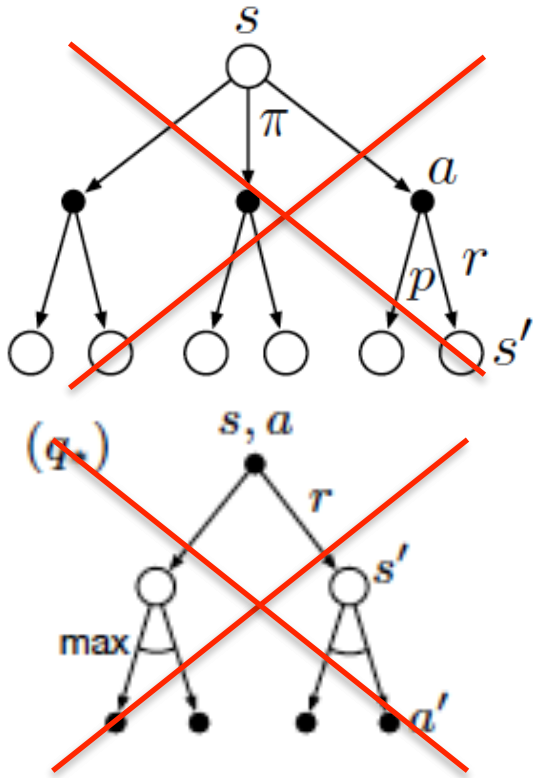
Can we use more information?



Uses the Q-value for actions that weren't chosen from s'



Uses visited states and actions from further in the future

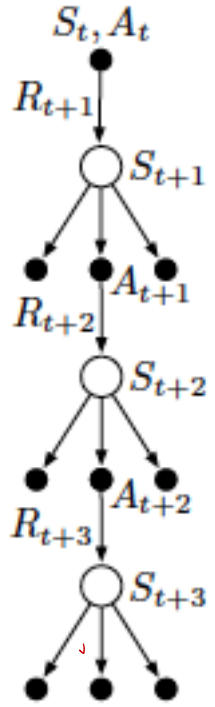


Uses states that would be visited by actions that weren't chosen

N-step tree back-up

Combine these sources of information:

- Future states that we have visited in this path
- Actions from such states that we have not taken, but we know the Q-values



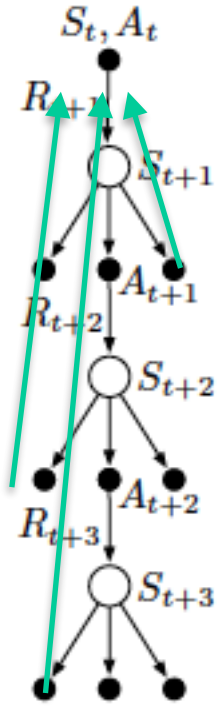
the 3-step tree-backup update

N-step tree back-up

Combine these sources of information:

- Future states that we have visited in this path
- Actions from such states that we have not taken, but we know the Q-values

Q-value at each of the leaf node participates in the update with the rewards obtained on the way



the 3-step tree-backup update

N-step tree back-up

Results in different types of n-step returns

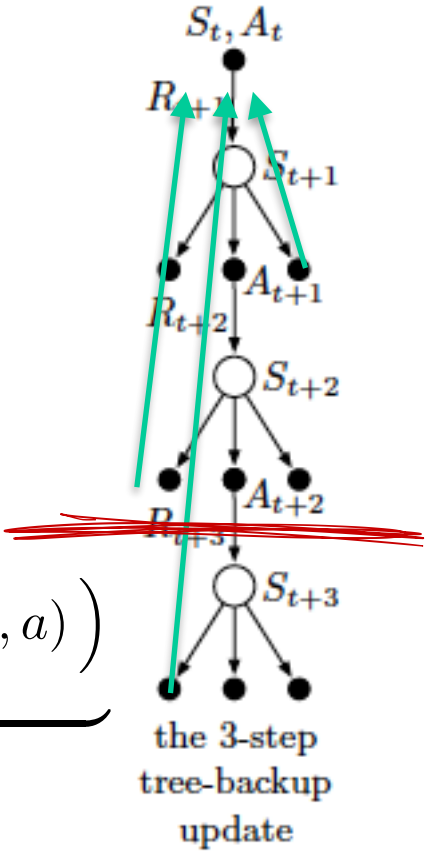
(n=1, n=2, n=3)

Weight each n-step return with the probability of taking the path to that node, e.g. for 2 steps:

$$G_{t:t+2} \doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a)$$

$$+ \gamma \pi(A_{t+1}|S_{t+1}) \underbrace{\left(R_{t+2} + \gamma \sum_{a} \pi(a|S_{t+2}) Q_{t+1}(S_{t+2}, a) \right)}_{G_{t+1:t+2}}$$

prob. of going deeper in tree



the 3-step tree-backup update

N-step tree back-up

Results in different types of n-step returns

(n=1, n=2, n=3)

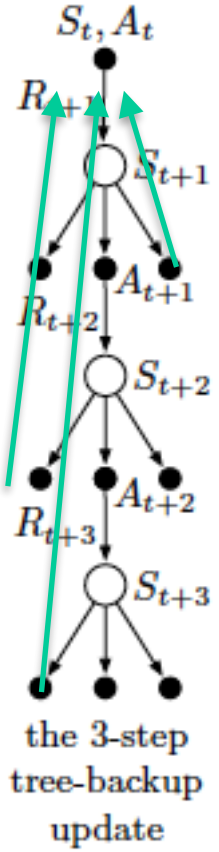
Weight each n-step return with the probability of taking the path to that node, e.g. for 2 steps:

$$G_{t:t+2} \doteq R_{t+1} + \gamma \sum_{a \neq A_{t+1}} \pi(a|S_{t+1}) Q_{t+1}(S_{t+1}, a) + \gamma \pi(A_{t+1}|S_{t+1}) \underbrace{\left(R_{t+2} + \gamma \sum_{a} \pi(a|S_{t+2}) Q_{t+1}(S_{t+2}, a) \right)}_{G_{t+1:t+2}}$$

$G_{t:t+n}$

$G_{t+1:t+n}$

Generalises for more than two steps!



the 3-step tree-backup update

N-step tree back-up

In the tree-backup, all actions are weighted by π.

(behavior policy)

The sampling distribution changes where longer updates are possible but does not change expected value.

Thus: N-step tree backup also works for **off-policy data without importance sampling**

So many methods...

We have talked about **a lot** of methods so far, how to keep organized?

The main differences in methods is captured by a few dimensions:

- Estimate V (only prediction) or Q (allows control)? (State- or State-action)
- Off-policy or on-policy
- ‘Wide’ updates versus ‘narrow' updates
- ‘Deep’ updates versus ‘shallow’ updates

Prediction methods

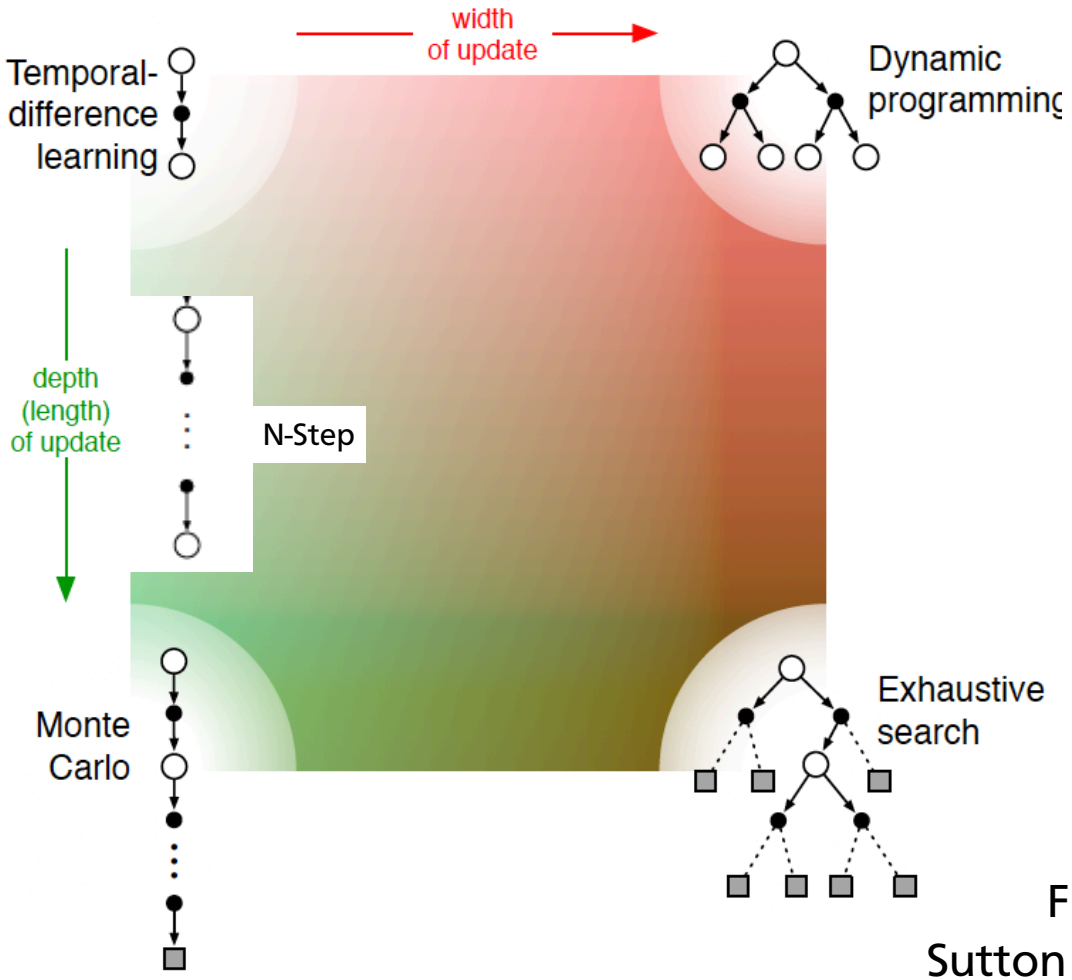


Figure from
Sutton and Barto RL:AI

Control methods

On-policy (narrow)

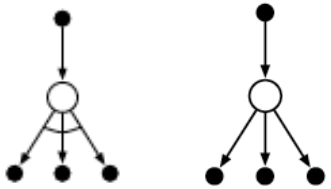
Sarsa

N-step TD

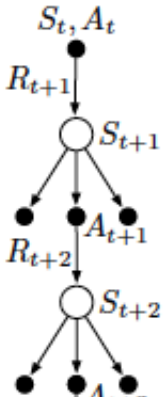
Monte-Carlo control

Depth

Off-policy (wide)



Q-learning & Expected SARSA



Tree-back-up

Limitations so far

We've seen quite a few methods so far, including famous algorithms like SARSA and Q-learning

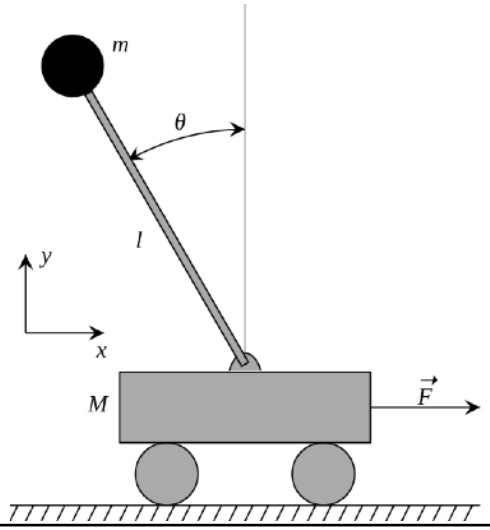
However, we have many lectures left. What do we need beyond what we've covered so far?

Limitations so far: State and action space

All methods so far requiring storing Q- or V- functions as a table ('tabular methods').

Storing a table only works for discrete, finite and 'small' sets of states and actions.

How about huge discrete or continuous state and action sets?

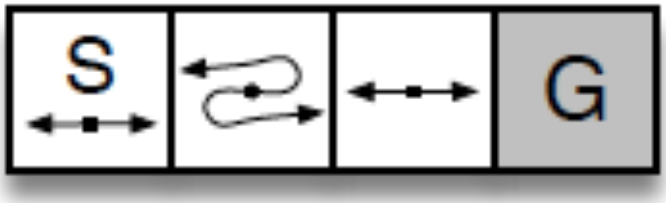


Limitations so far: pre-set stochasticity

We have seen different methods to randomise policies for exploration. But all require us to set `how random'. (epsilon, temperature)

Can we learn the amount of randomness?

Useful if state can't be fully determined...



Sutton & Barto. Reinforcement Learning: An introduction.

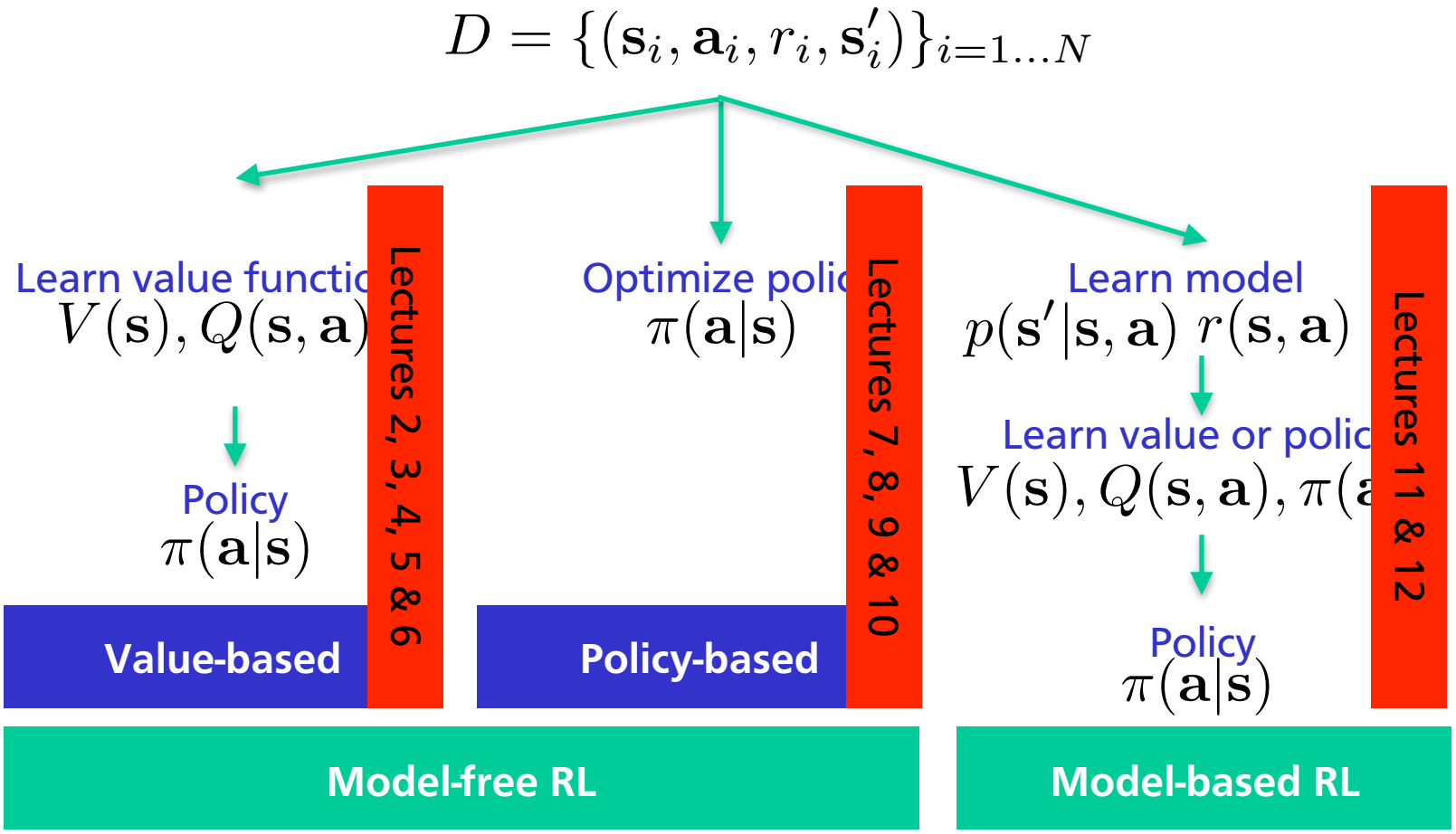
Limitation so far: sample efficiency

We have to try many times to learn whether an action is good or not (requiring many data points)

Can we instead ‘plan ahead’ using internal predictions about the effects of our actions?

Tengrain / Flicker / CC BY-NC 2.0

Big picture: How to learn policies



Thanks to Jan Peters

What you should know

What are advantages of TD(0), N-step, and Monte-Carlo

What are some off-policy and on-policy methods for TD-learning and their properties?

What is maximisation bias?

How can value-function methods be categorized?

Feedback?

h.c.vanhoof@uva.nl

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