Advanced TD methods

Herke van Hoof

Big picture so far

Finding optimal policies

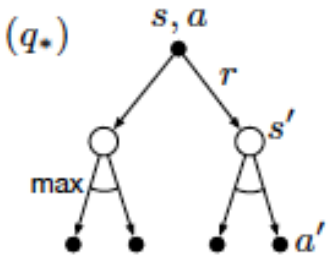
Known MDP dynamics

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

Dynamic programming

Policy iteration

Value iteration



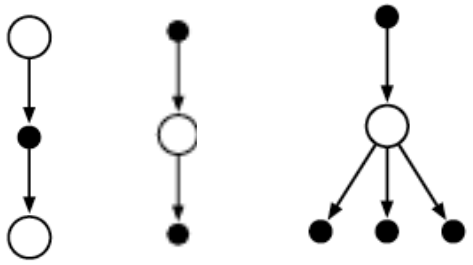
Only **data** from MDP: Reinforcement learning

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

Monte-Carlo

Temporal Difference

TD(0) & (Expected) SARSA



Q-learning, off-policy TD control

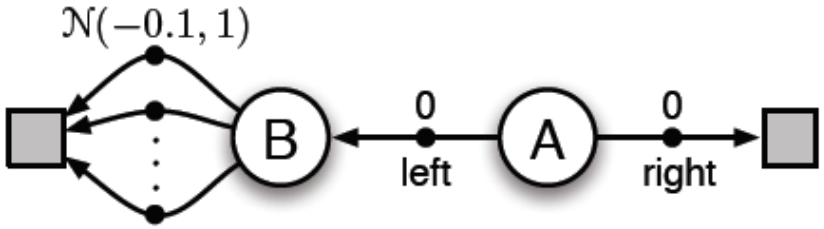
Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    $s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

Behavior doesn't need to converge to greedy

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

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

Maximization bias



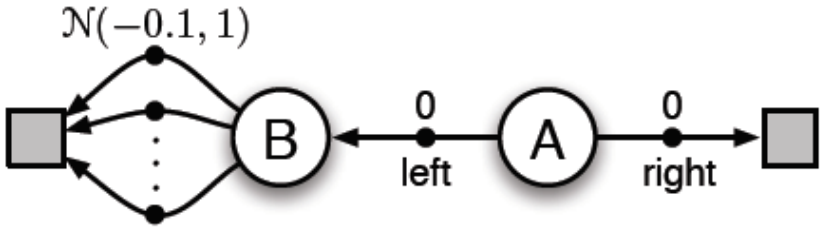
What is the true value of (A,left) and (B, a_1), (B, a_2), etc?

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

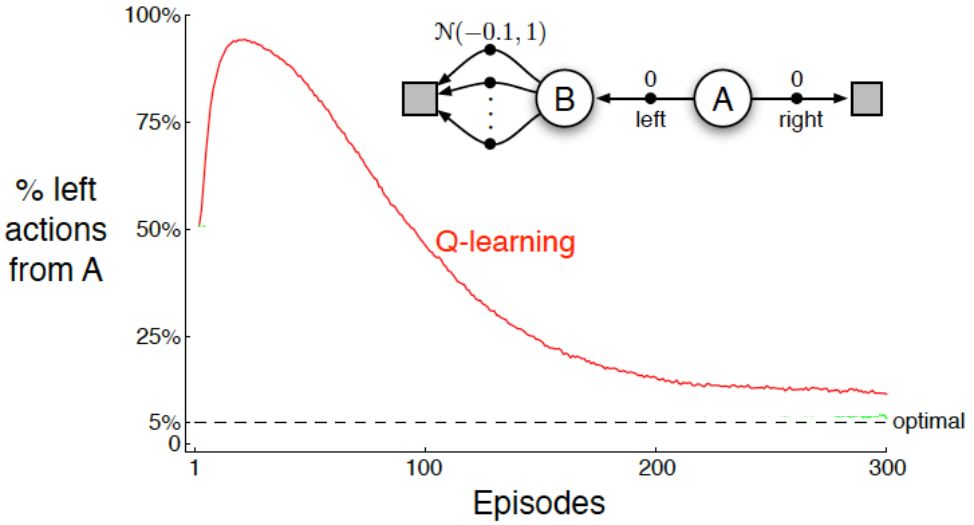
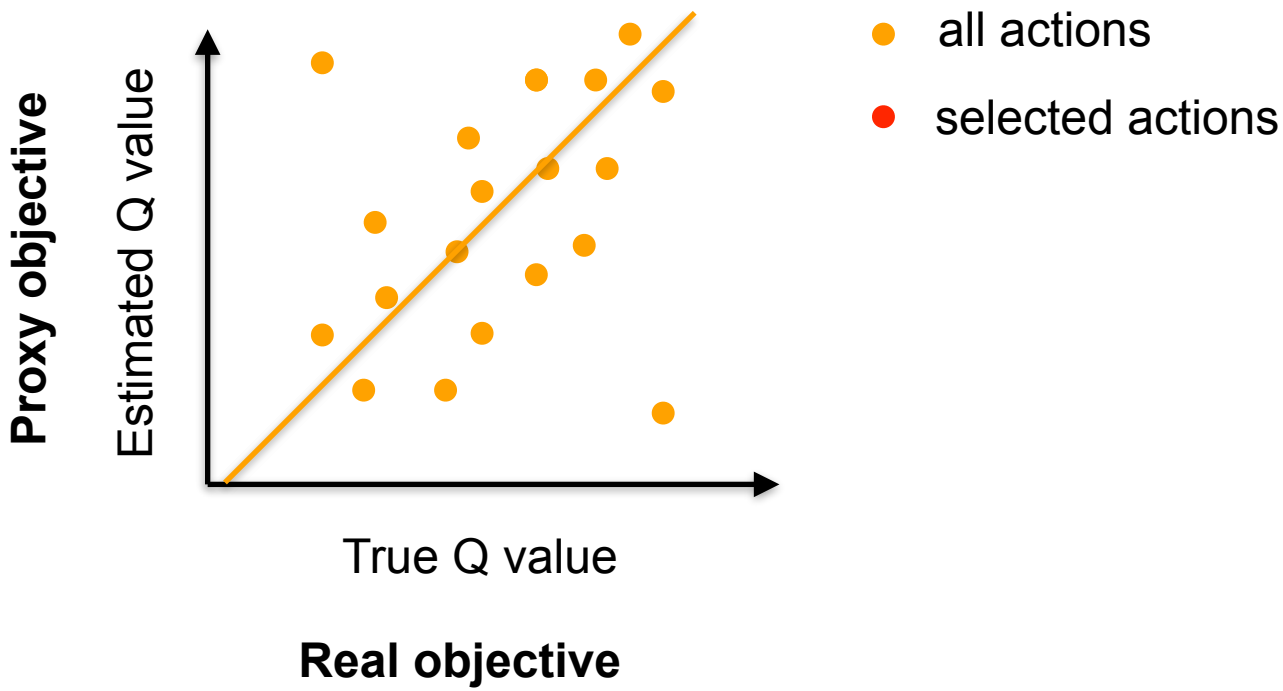
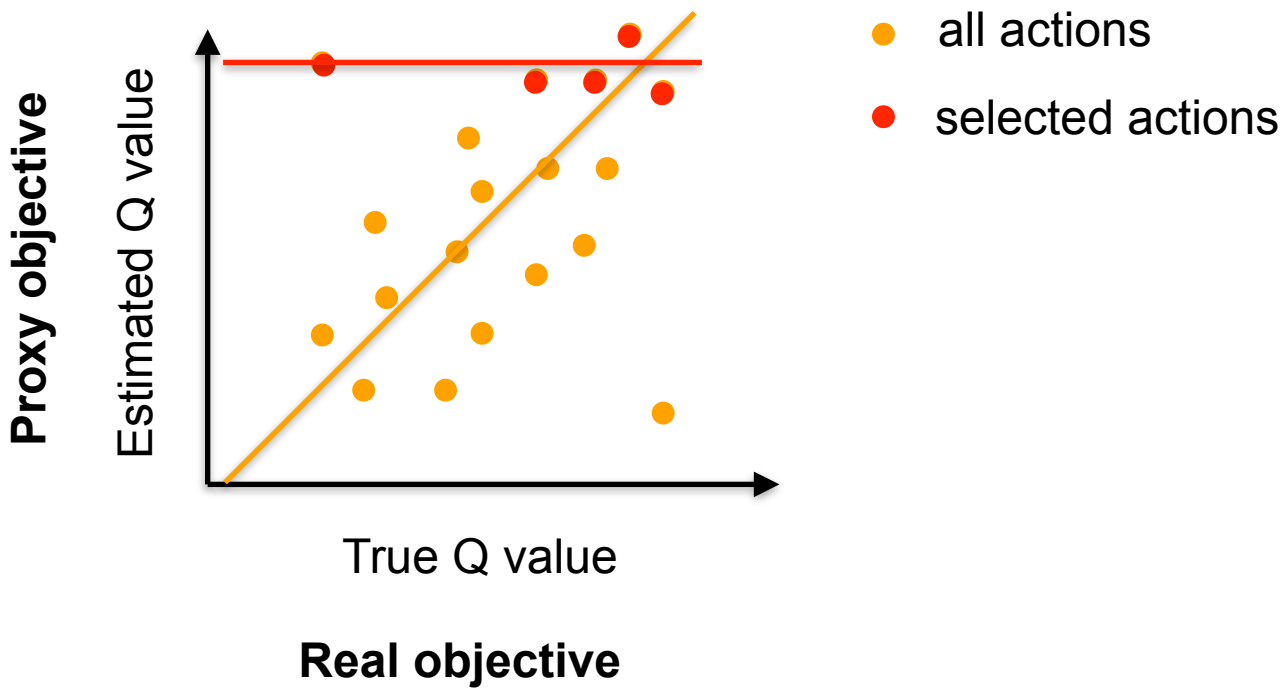


Figure from Sutton & Barto, RL:AI

Maximization bias

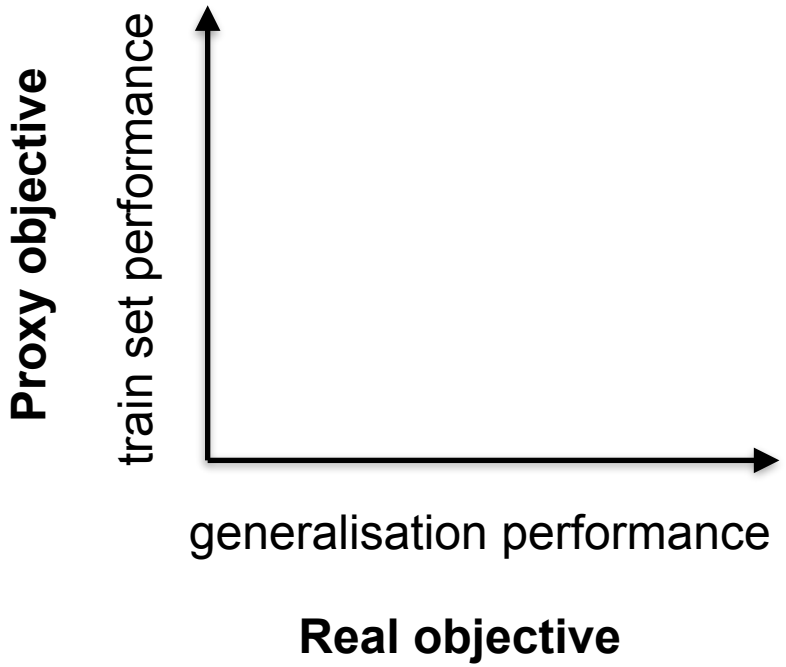


Maximization bias



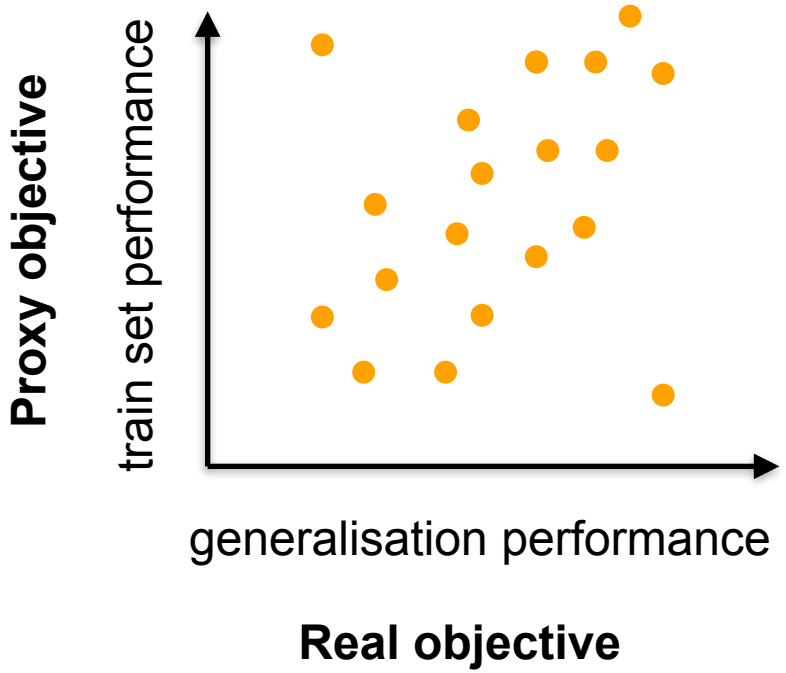
Optimization bias

Maximization bias very similar to optimization bias in supervised learning



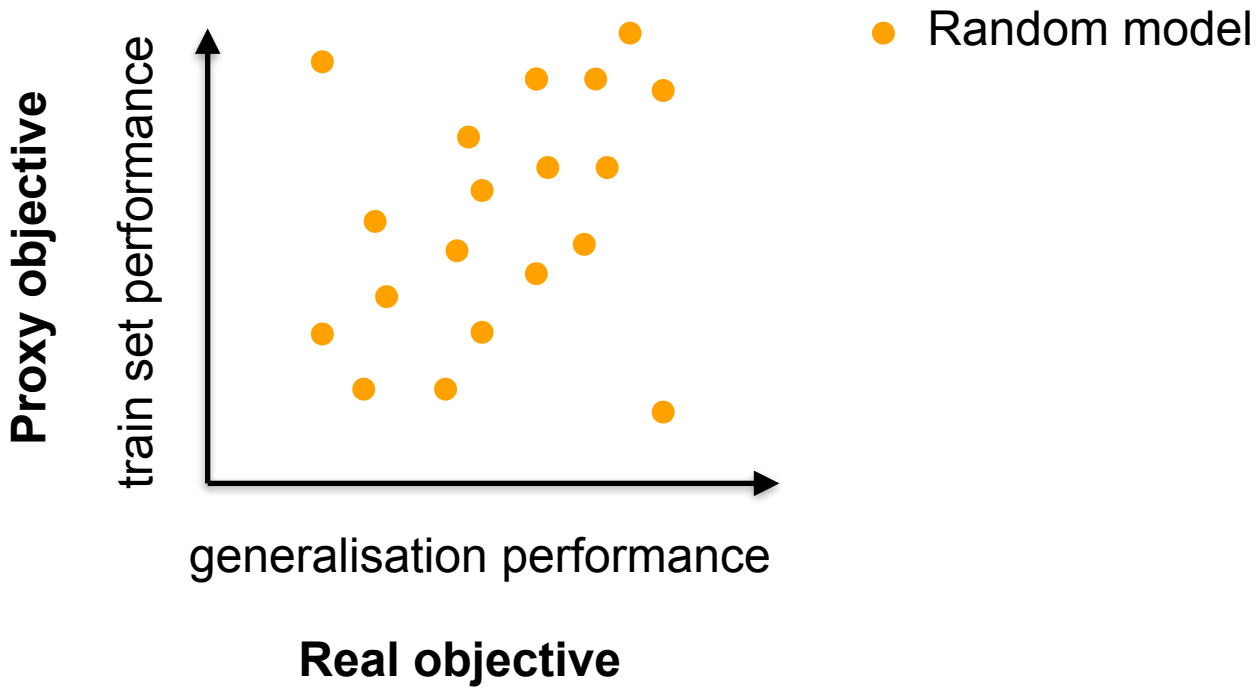
Optimization bias

Maximization bias very similar to optimization bias in supervised learning



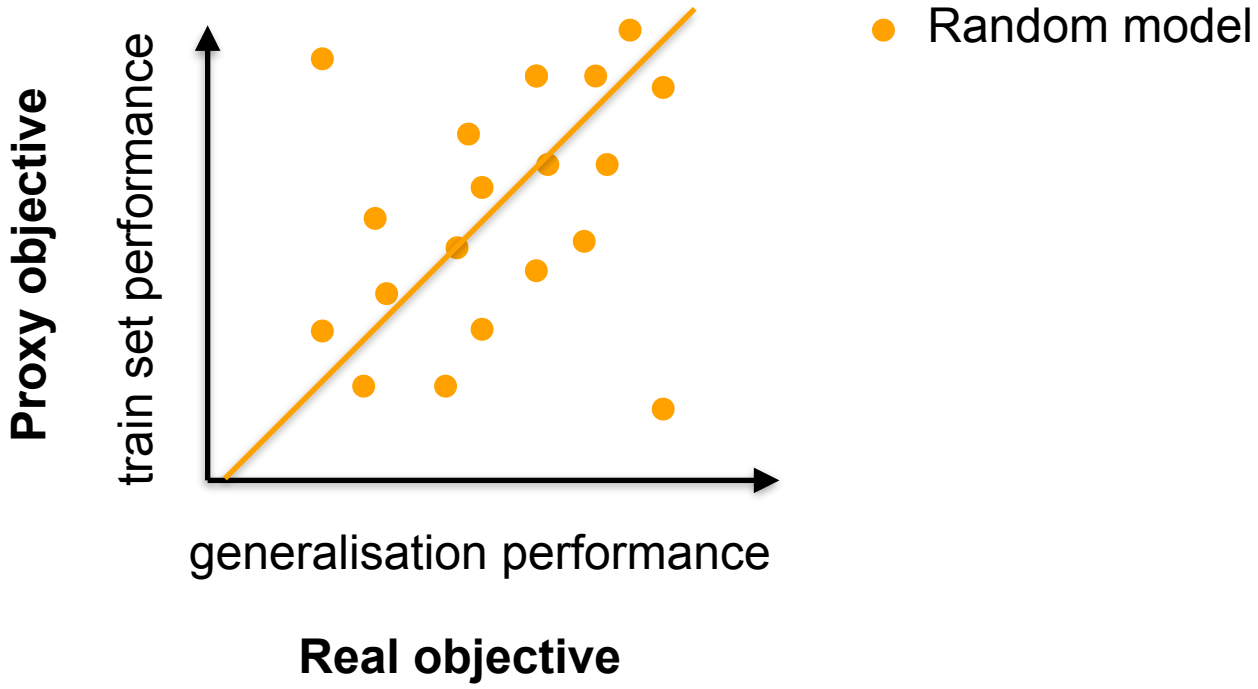
Optimization bias

Maximization bias very similar to optimization bias in supervised learning



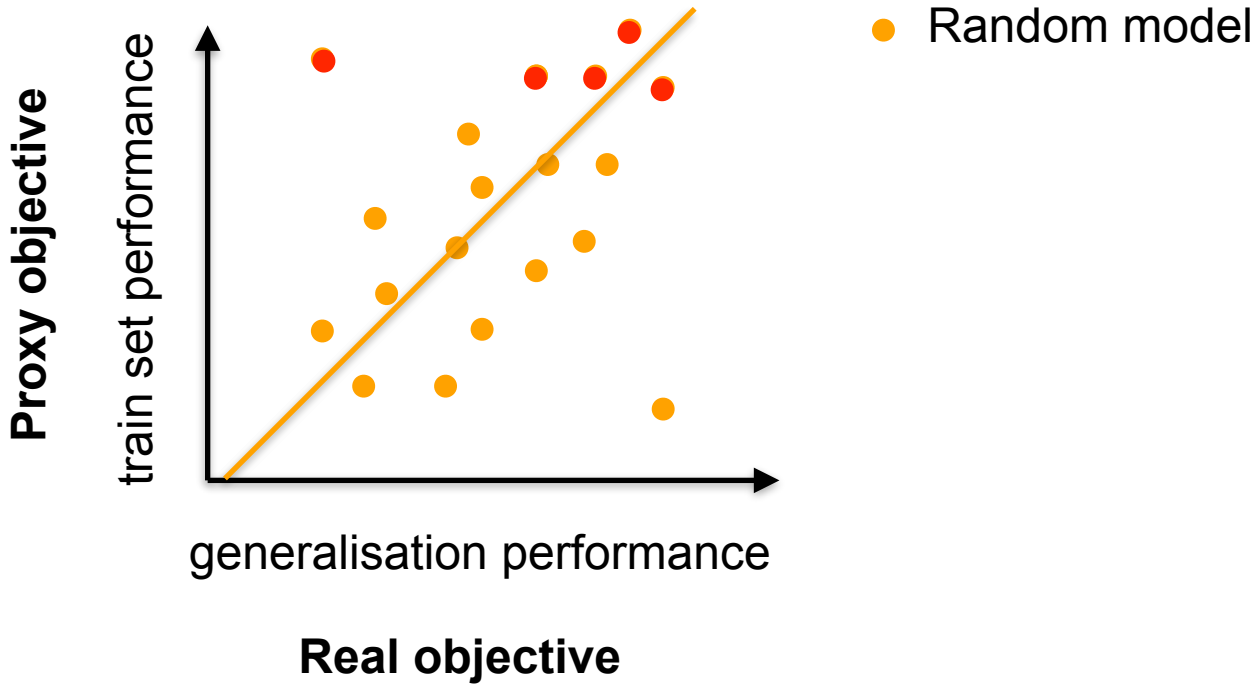
Optimization bias

Maximization bias very similar to optimization bias in supervised learning



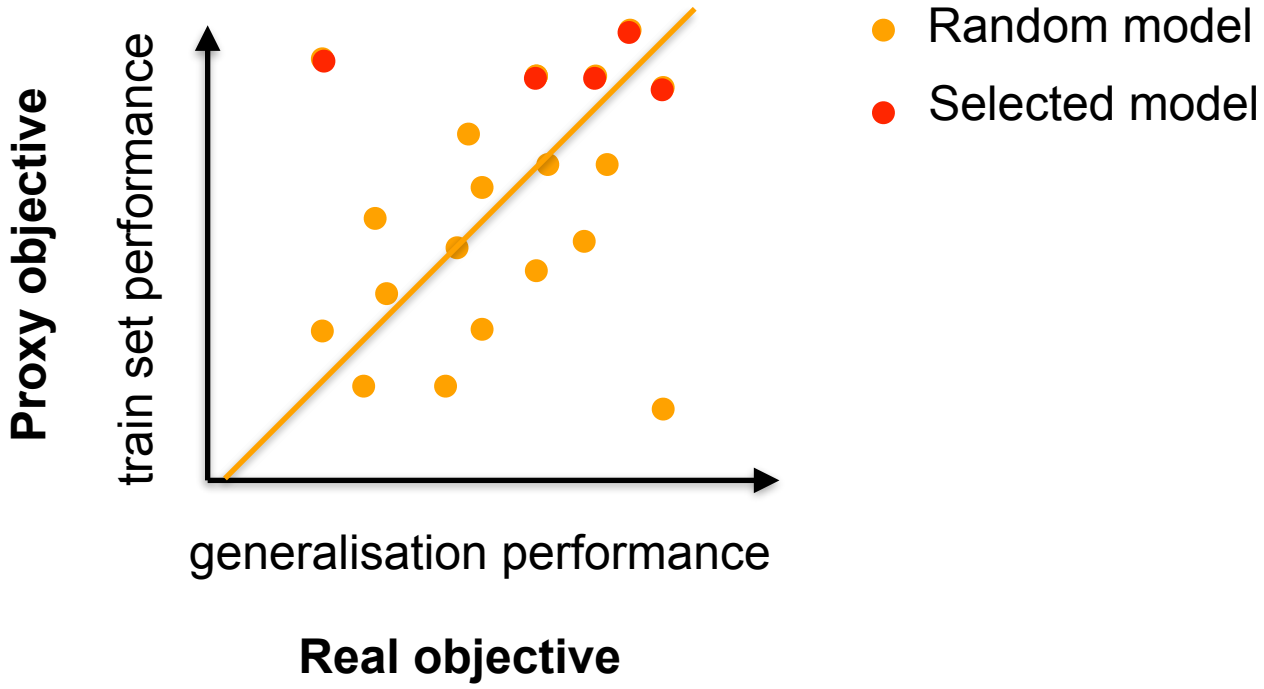
Optimization bias

Maximization bias very similar to optimization bias in supervised learning



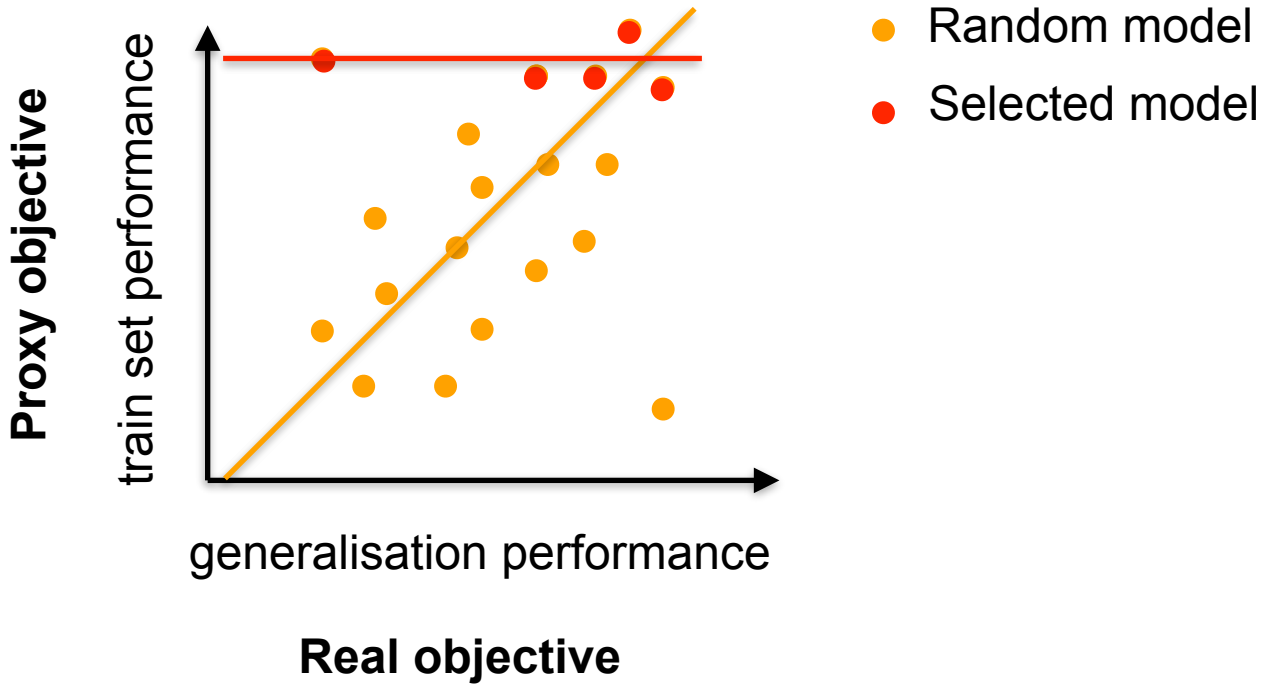
Optimization bias

Maximization bias very similar to optimization bias in supervised learning



Optimization bias

Maximization bias very similar to optimization bias in supervised learning



Optimisation bias

The update rule of Q-learning with discrete actions:

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

Image:
Scott Fujimoto

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

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

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**
Each sample used to update either Q1 or Q2 (say, with 50% probability)
Behavior can be based on Q1 and Q2 (e.g., their average)

Maximization bias

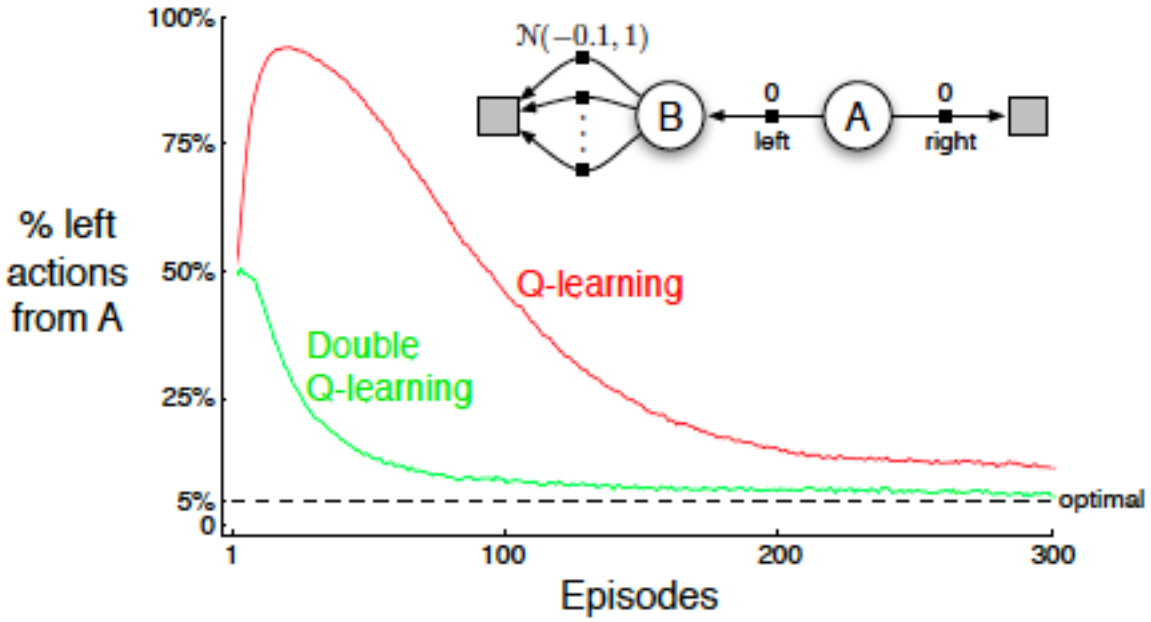


Figure from Sutton & Barto, RL:AI

Big picture so far

Finding optimal policies

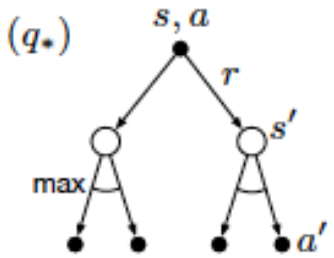
Known MDP dynamics

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

Dynamic programming

Policy iteration

Value iteration



Only **data** from MDP: Reinforcement learning

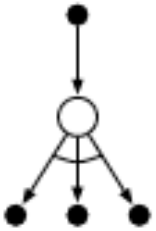
$$D = \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}_{i=1\ldots N}$$

Monte-Carlo

Temporal Difference

SARSA (on policy)

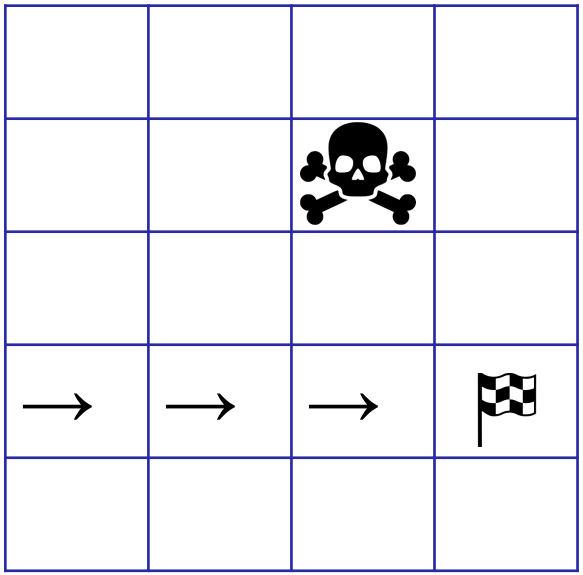
Q-learning (off policy)

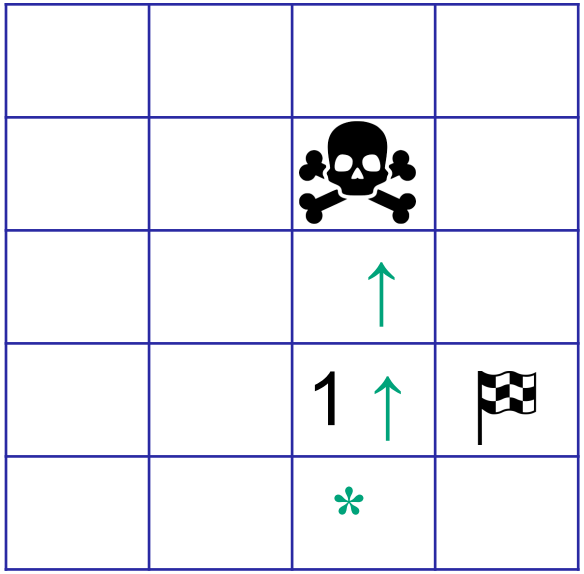


Bigger picture: MC and TD

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



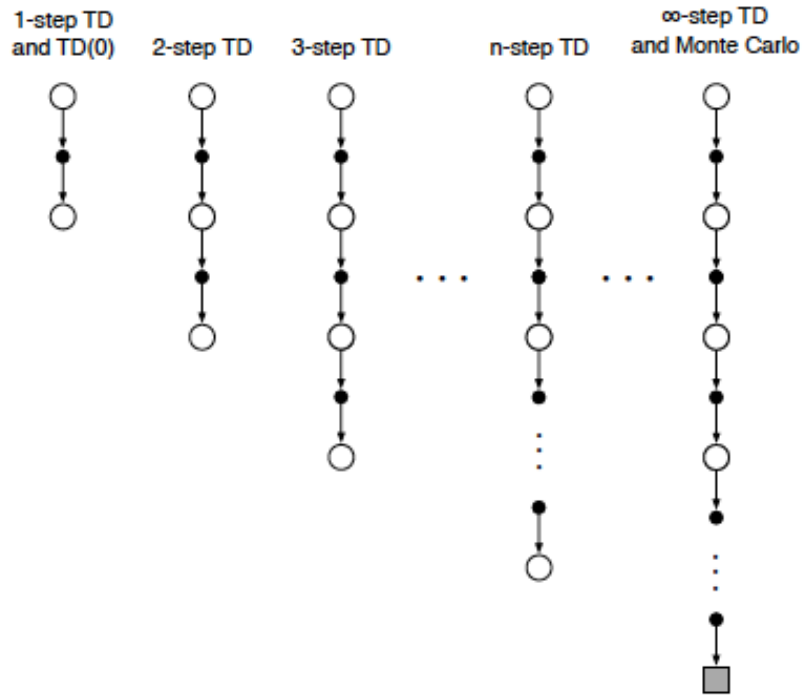


Bigger picture: MC and TD

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

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

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

Then update:

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

N-step prediction

Which N works best depends on the problem

Example: 19-state random-walk

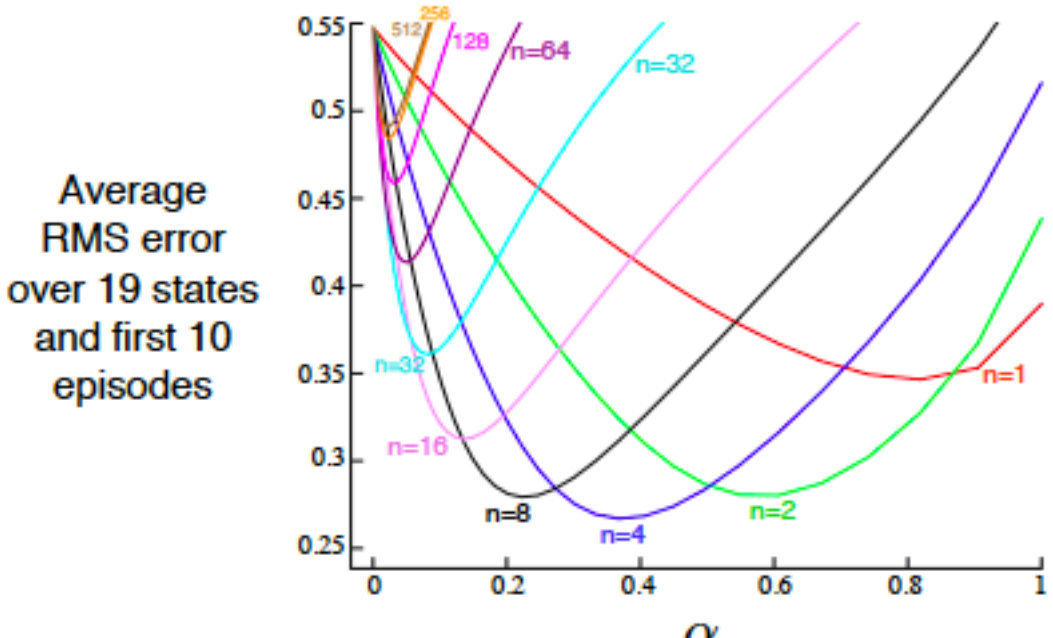
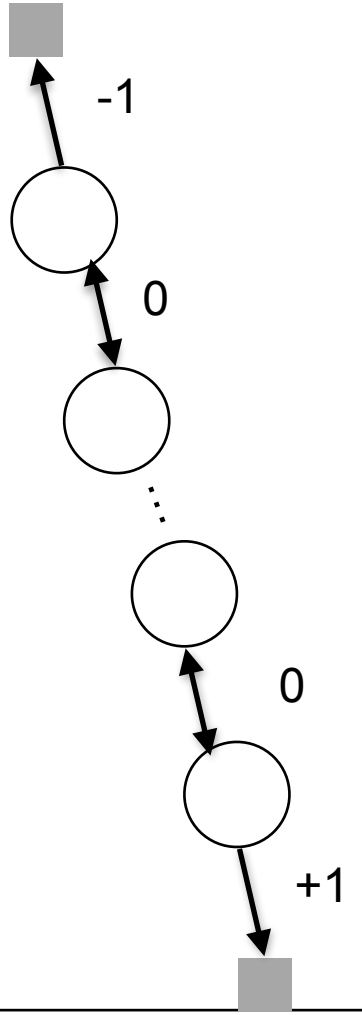


Figure from Sutton & Barto, RL:AI



N-step control

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

N-step Sarsa:

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

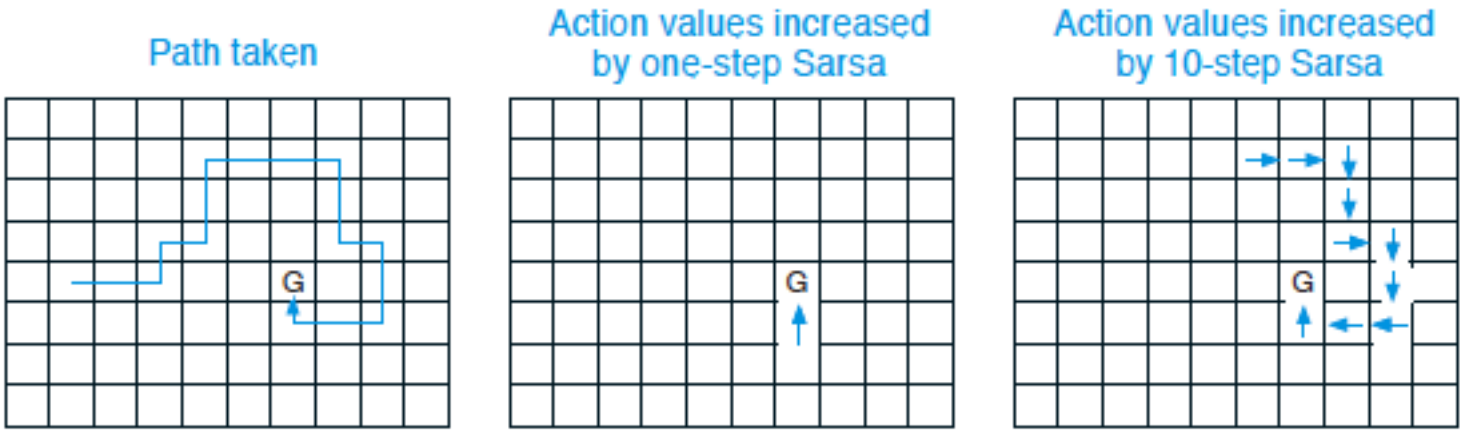
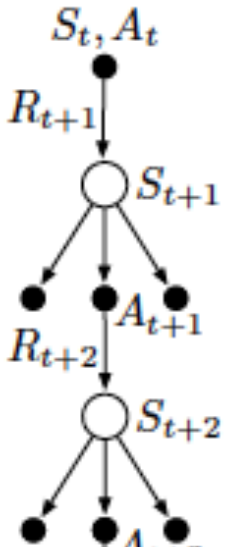


Figure from Sutton & Barto, RL:AI

Off-policy N-step learning

There are also off-policy algorithms for N-step learning.
We won't cover this in detail, but for a small glimpse:

- We can use importance weights, that depend on the ratio between actions probability of the target- and behavior policy for all *n* actions when we do an update
- We can estimate targets that 'would have been' achieved if the target policy was followed - like in expected SARSA.
  This leads to so-called *tree backups*



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

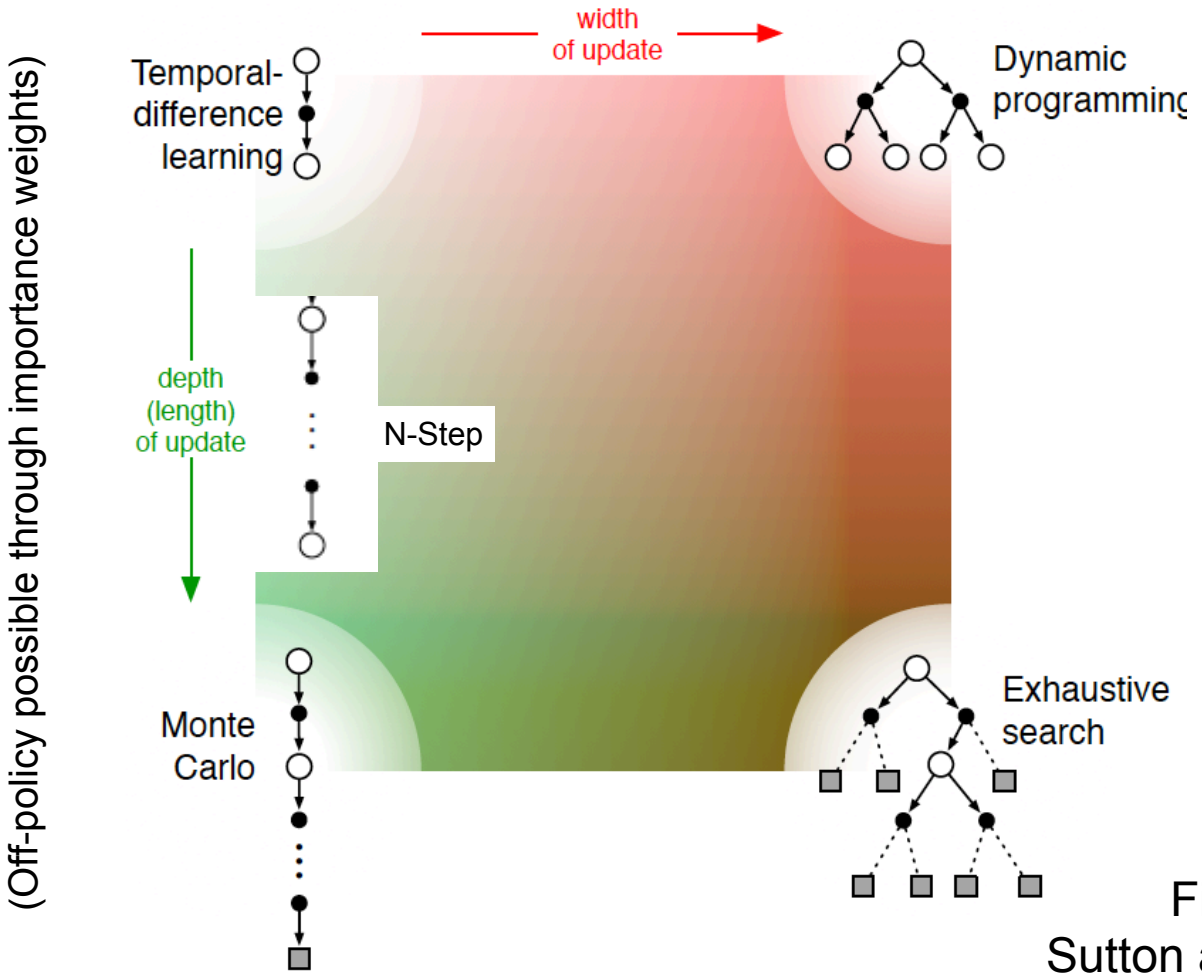


Figure from
Sutton and Barto RL:AI

Control methods

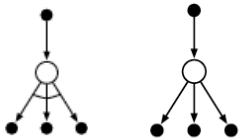
On-policy (narrow)

Sarsa

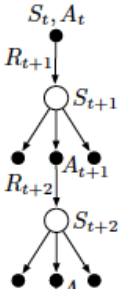
N-step TD

Monte-Carlo control

Off-policy (wide)

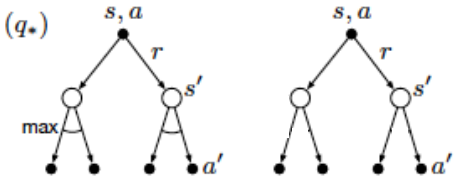


Q-learning & Expected SARSA



(Tree-back-up)

Dynamic programming (wide)



Q-value iteration
Policy evaluation

Depth

Limitations so far

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

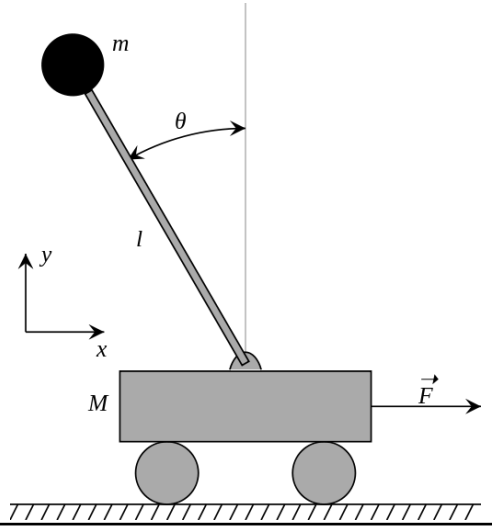
However, we have many lectures left. What do we need beyond what we've covered so far?

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…

Only 'color' of state is observed
Start in one of the green states
Optimal $p(a = \text{left} \mid \blacksquare)$ = ?

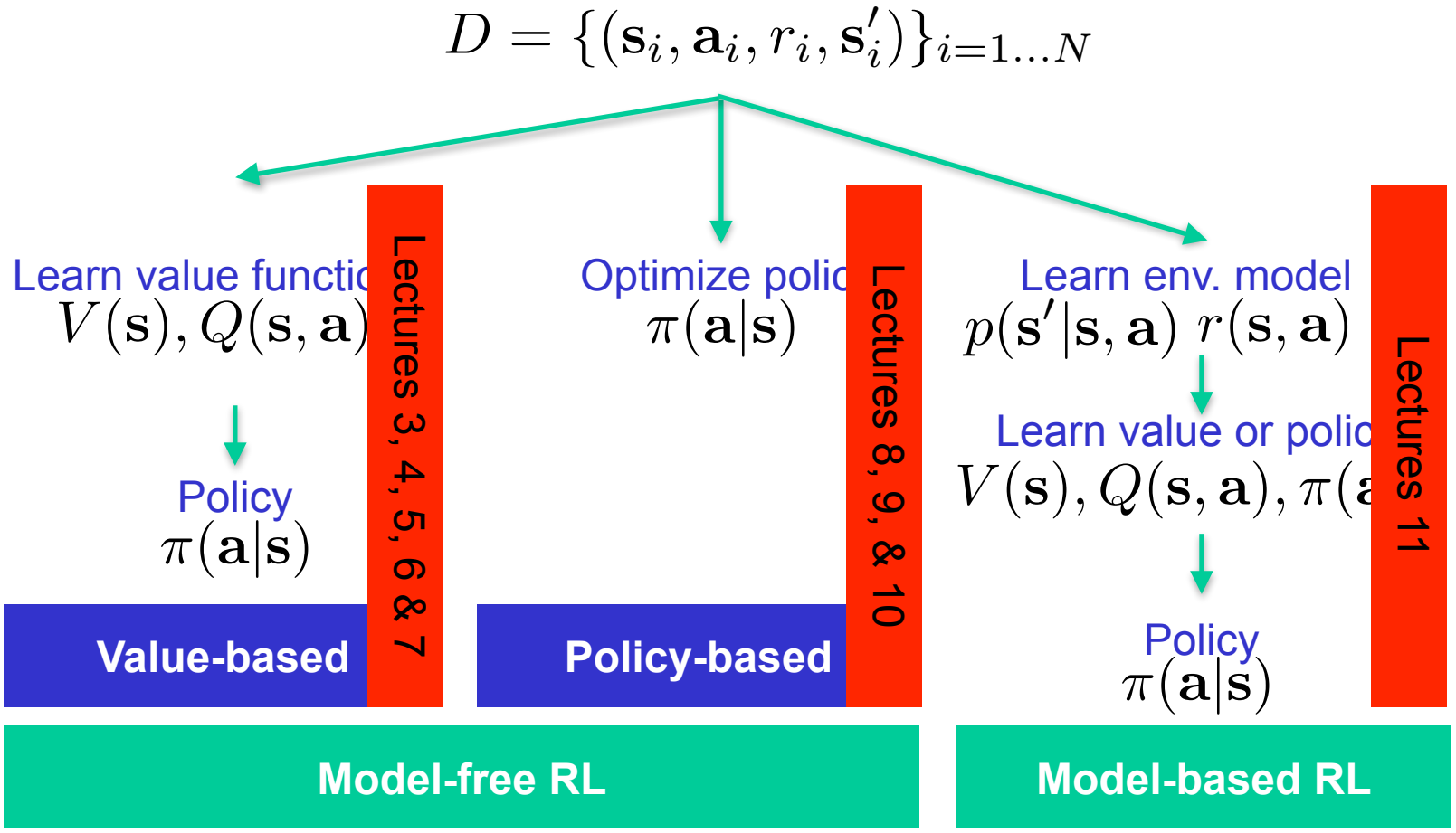
Limitation so far: sample efficiency

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



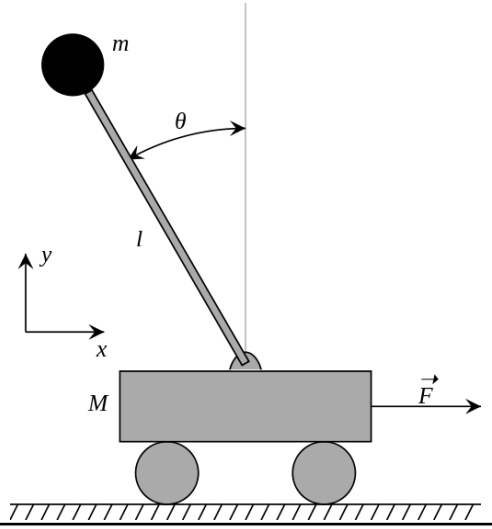
Thanks to Jan Peters

Handling large state spaces

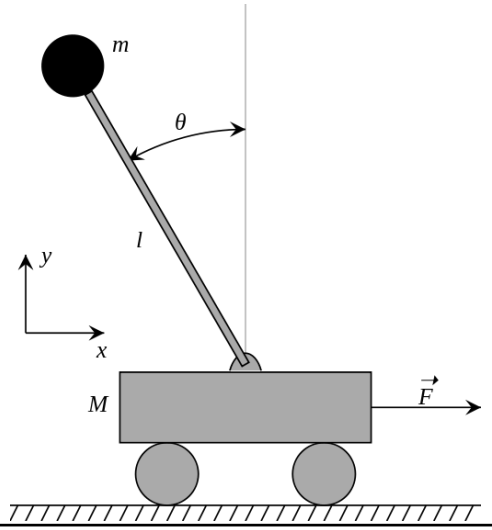
With a huge number of states, several problems:

- The Q/V table might be too large to fit in memory
- It might take too long to collect enough experiences to fill up the table
- The state space might be continuous (infinitely many states)

Today, we'll look at how to overcome these problems in the *policy evaluation* (i.e., prediction) setting. *Control* next week!

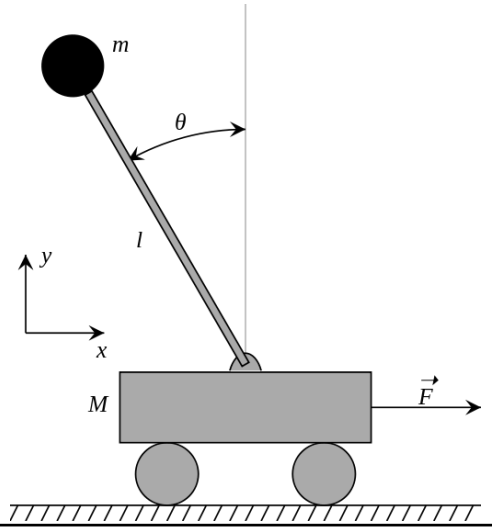


Handling large state spaces



Handling large state spaces

Luckily, certain small changes in state usually don't impact the value/action too much. **We can generalize.**



Why approximation

So we would like to represent the value function

- In a compact way (requires approximation)
- In a way that allows generalizing an experience to nearby states
- While still being close to the true value function

This is similar to the goal of supervised learning

- Very flexible function might not generalise well (overfitting)
- Inflexible functions are not very expressive
- Need to find a good balance, need for expressivity <> data availability

An example

Let's consider just splitting the set of all states into groups

We'll only store one representative value for each group

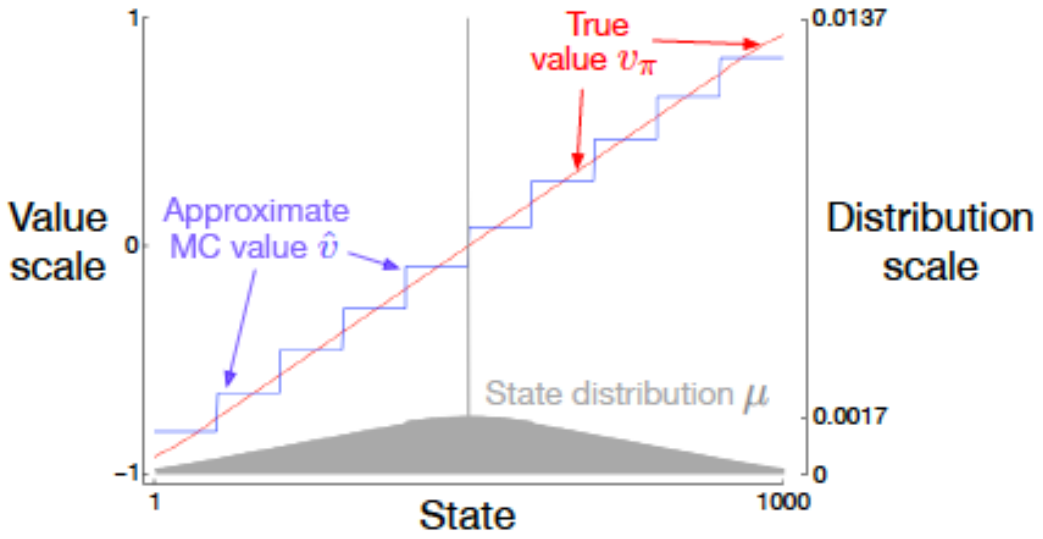
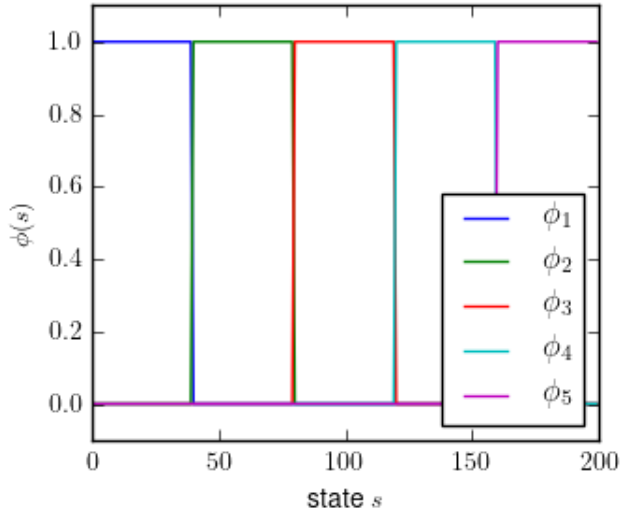
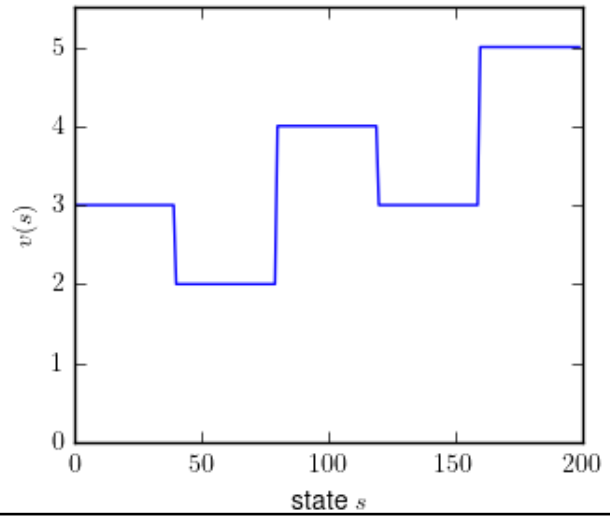


Figure: Sutton & Barto. RL:AI

An example



One-hot encoding:
$\phi_1(s) = 1$ iff s is in group one, etc.



Parameters

Features

$$\hat{v}(s, \mathbf{w}) = \mathbf{w}^T \boldsymbol{\phi}(s) = \sum_i w_i \cdot \phi_i(s)$$

Parametrized approximate value

Figure: Sutton & Barto. RL:AI

Objective

What are the ‘best’ representative values **w**?

$$\overline{\mathrm{VE}}(\mathbf{w}) \doteq \sum_{s \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, \mathbf{w}) \right]^2$$

Not all states are equally important!

Objective

What are the 'best' representative values **w**?

$$\overline{\mathrm{VE}}(\mathbf{w}) \doteq \sum_{s \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, \mathbf{w}) \right]^2$$

Not all states are equally important!

Today: On-policy distribution

- How often is each state visited by the policy?
- We don't exactly know μ typically.
  We'll see we can easily sample from it, though.

Stochastic gradient descent

Let's follow the negative gradient! But how to approximate

$$\nabla_{\mathbf{w}} \overline{\mathrm{VE}}(\mathbf{w}) = \nabla_{\mathbf{w}} \sum_{s \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, \mathbf{w}) \right]^2$$

Stochastic gradient descent

Let's follow the negative gradient! But how to approximate

$$\nabla_{\mathbf{w}} \overline{\mathrm{VE}}(\mathbf{w}) = \nabla_{\mathbf{w}} \sum_{s \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, \mathbf{w}) \right]^2$$

- Pick random samples from *μ(s)* (how?)
- Generate unbiased estimates of target v_π (how?)

Stochastic gradient descent

Let's follow the negative gradient! But how to approximate

$$\nabla_{\mathbf{w}} \overline{\mathrm{VE}}(\mathbf{w}) = \nabla_{\mathbf{w}} \sum_{s \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, \mathbf{w}) \right]^2$$

- Pick random samples from *μ(s)* (how?)

  As *μ* is on-policy distribution, simply pick random visited state

- Generate unbiased estimates from v_π (how?)

  Use the return *G*

Stochastic gradient descent

$$\nabla_{\mathbf{w}} \overline{\mathrm{VE}}(\mathbf{w}) = \nabla_{\mathbf{w}} \sum_{s \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, \mathbf{w}) \right]^2$$

Plug in sampled transitions and returns

Stochastic gradient descent

$$\nabla_{\mathbf{w}} \overline{\mathrm{VE}}(\mathbf{w}) = \nabla_{\mathbf{w}} \sum_{s \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, \mathbf{w}) \right]^2$$

Plug in sampled transitions and returns

$$\begin{aligned} &\approx \nabla_{\mathbf{w}} \sum_t \left(G_t - \hat{v}(s_t, \mathbf{w}) \right)^2 \\ &= -2 \sum_t \left(G_t - \hat{v}(s_t, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{v}(s_t, \mathbf{w}) \end{aligned}$$

Stochastic gradient descent

$$\nabla_{\mathbf{w}} \overline{\mathrm{VE}}(\mathbf{w}) = \nabla_{\mathbf{w}} \sum_{s \in \mathcal{S}} \mu(s) \left[v_\pi(s) - \hat{v}(s, \mathbf{w}) \right]^2$$

Plug in sampled transitions and returns

$$\approx \nabla_{\mathbf{w}} \sum_t \left(G_t - \hat{v}(s_t, \mathbf{w}) \right)^2$$

$$= -2 \sum_t \left(G_t - \hat{v}(s_t, \mathbf{w}) \right) \nabla_{\mathbf{w}} \hat{v}(s_t, \mathbf{w})$$

This leads to the gradient Monte Carlo update

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left[G_t - \hat{v}\left(S_t, \mathbf{w}_t\right) \right] \nabla \hat{v}\left(S_t, \mathbf{w}_t\right)$$

- (Use samples and returns from on-policy distribution)

Stochastic gradient descent

Similar approach to gradient MC but with bootstrapping?

- Next lecture!

How about learning with approximation for control?

- Next week!

What you should know

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

What is maximisation bias?

How can tabular value-based methods be categorized?

What is gradient MC and why is it useful?

Thanks for your attention

Feedback?

h.c.vanhoof@uva.nl