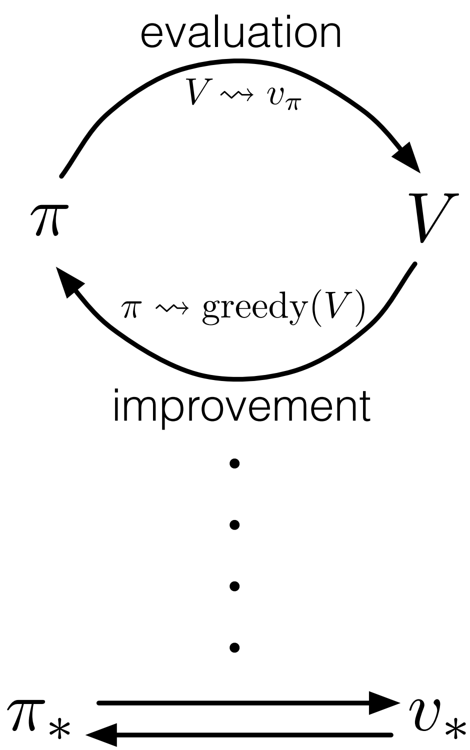
Sequential decision making
Control: Q-learning
What can we say formally about convergence?

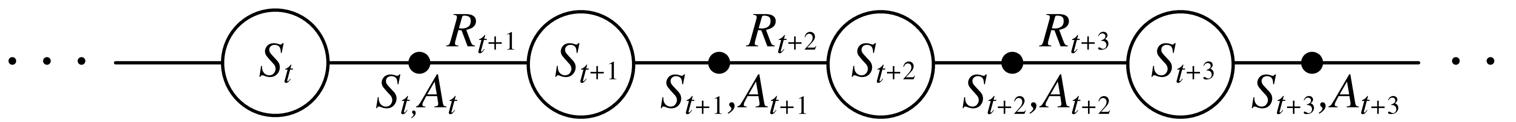
How to do control? GPI!

Generalized Policy Iteration (GPI):
any interaction of policy evaluation and policy improvement, independent of their granularity.



Monte Carlo Estimation of Action Values

Estimate q_π for the current policy π



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t))$$

where $G_t = \sum_{k=1}^{T-t} \gamma^{k-1} R_{t+k}$

and T is the time of entering terminal state

Monte Carlo Estimation of Action Values (Q)

- $q_\pi(s,a)$ - average return starting from state *s* and action *a* following π
- Converges asymptotically *if* every state-action pair is visited
- *Exploring starts:* Every state-action pair has a non-zero probability of being the starting pair

On-policy Monte Carlo Control

- *On-policy:* learn about policy currently executing
- How do we get rid of exploring starts?
  - The policy must be eternally *soft*:
    - $\pi(a|s) > 0$ for all s and a
  - e.g. ε-soft policy:
    - probability of an action = $\frac{\epsilon}{|\mathcal{A}(s)|}$ or $1 - \epsilon + \frac{\epsilon}{|\mathcal{A}(s)|}$

      non-max max (greedy)

- Similar to GPI: move policy *towards* greedy policy (e.g., ε-greedy)
- Converges to best ε-soft policy

Convergence of MC Control

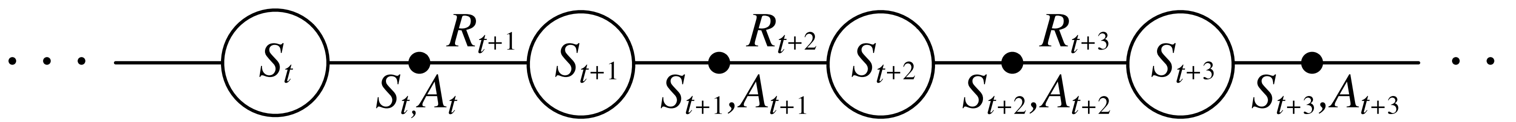
- Greedified policy meets the conditions for policy improvement:

$$\begin{aligned} q_{\pi_k}(s, \pi_{k+1}(s)) &= q_{\pi_k}(s, \arg\max_a q_{\pi_k}(s,a)) \\ &= \max_a q_{\pi_k}(s,a) \\ &\geq q_{\pi_k}(s, \pi_k(s)) \\ &\geq v_{\pi_k}(s). \end{aligned}$$

- And thus must be $\geq \pi_k$ by the policy improvement theorem
- This assumes exploring starts and infinite number of episodes for MC policy evaluation
- To solve the latter:
  - update only to a given level of performance
  - alternate between evaluation and improvement per episode

TD-Style Learning for Action-Values

Estimate q_π for the current policy π



After every transition from a nonterminal state, S_t, do this:

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

If S_{t+1} is terminal, then define $Q(S_{t+1}, A_{t+1}) = 0$

Sarsa: On-Policy TD Control

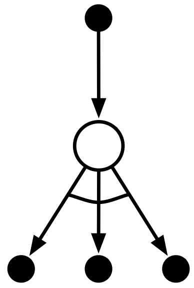
Turn this into a control method by always updating the policy to be greedy with respect to the current estimate:

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
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

Q-Learning: Off-Policy TD Control

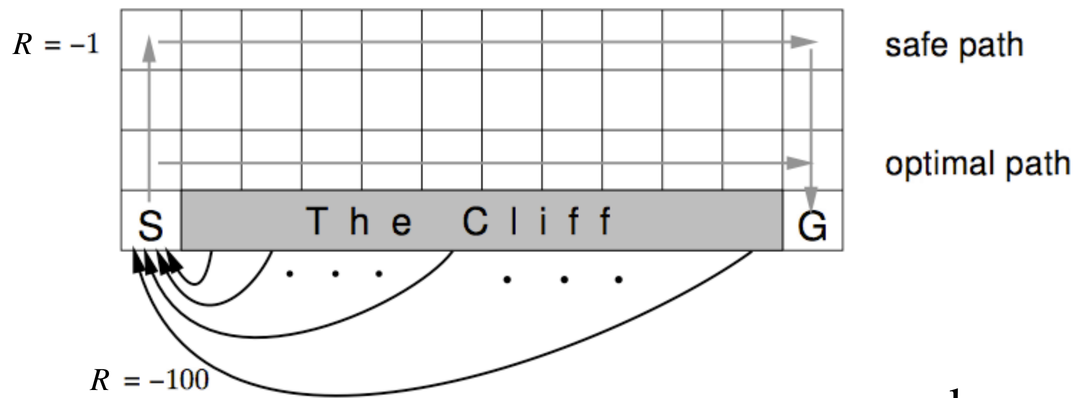
One-step Q-learning:

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

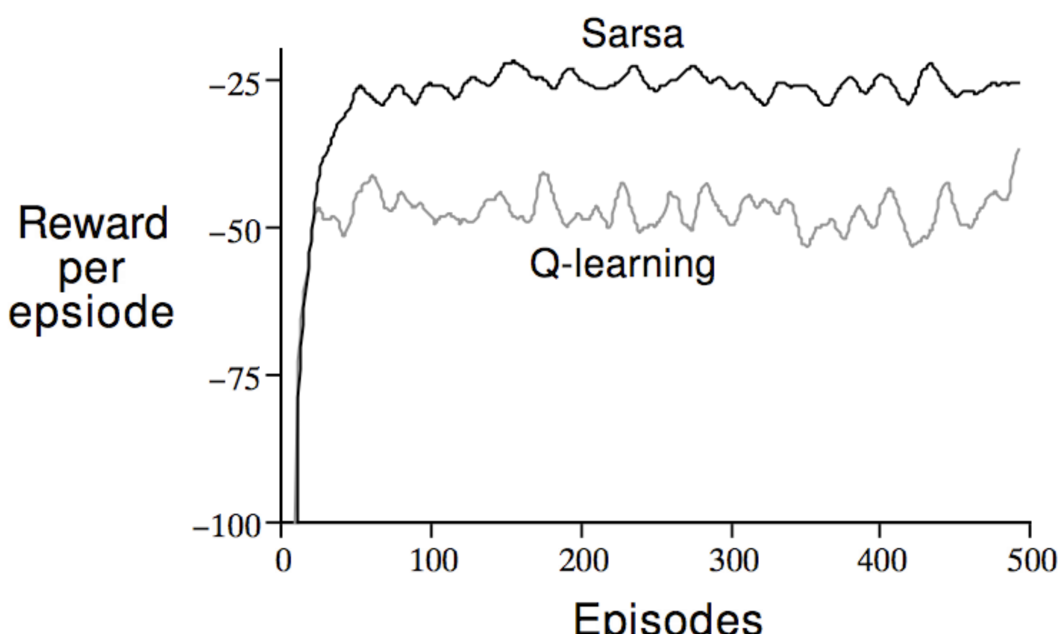


Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Repeat (for each step of episode):
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, 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

Cliffwalking



ε−greedy, ε = 0.1



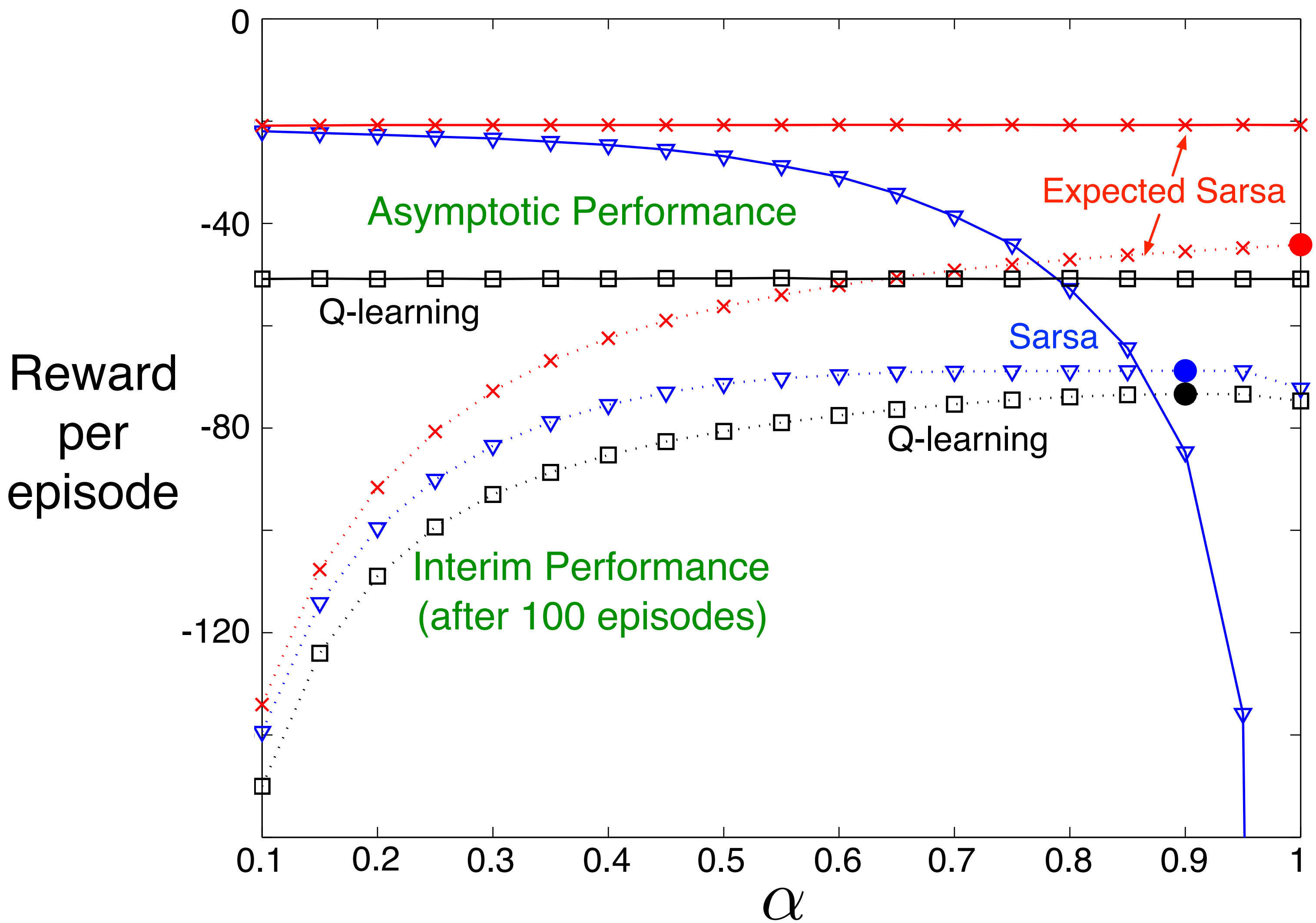
Expected Sarsa

- Instead of the *sample* value-of-next-state, use the expectation!

$$
\begin{aligned}
Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma\mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t)\Big] \\
&\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma\sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) - Q(S_t, A_t)\Big]
\end{aligned}
$$

- Expected Sarsa's performs better than Sarsa (but costs more)

Performance on the Cliff-walking Task



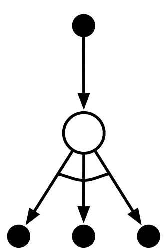
Off-policy Expected Sarsa

- Expected Sarsa generalizes to arbitrary behavior policies μ
  - in which case it includes Q-learning as the special case in which π is the greedy policy

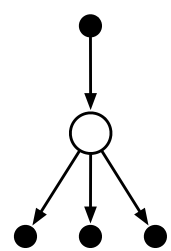
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \mathbb{E}[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t) \Big]$$

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

Nothing changes here



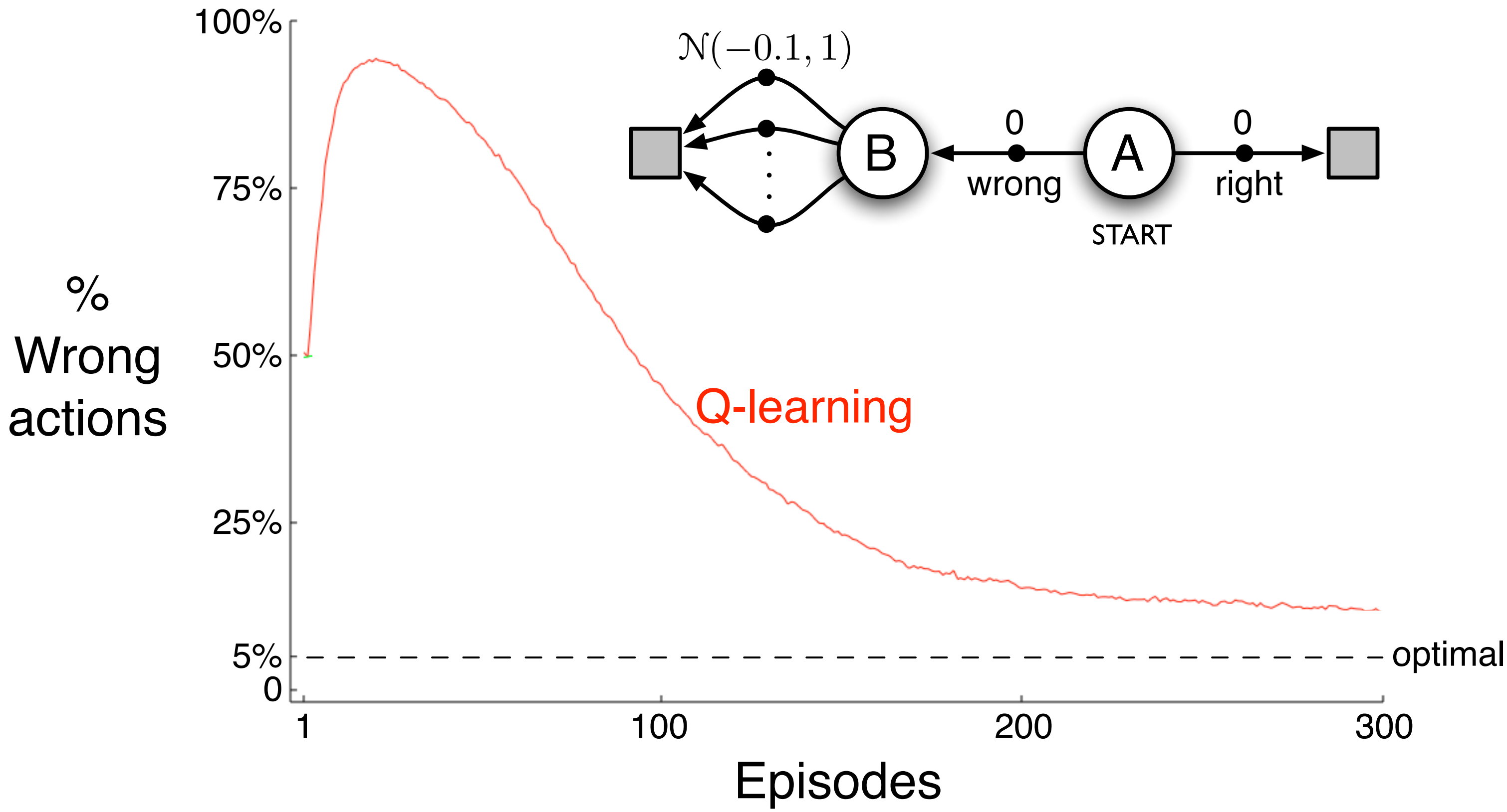
Q-learning



Expected Sarsa

- This idea seems to be new

Maximization Bias Example



Tabular Q-learning: $$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big]$$

Hado van Hasselt 2010

Double Q-Learning

- Train 2 action-value functions, Q_1 and Q_2
- Do Q-learning on both, but
  - never on the same time steps (Q_1 and Q_2 are indep.)
  - pick Q_1 or Q_2 at random to be updated on each step
- If updating Q_1, use Q_2 for the value of the next state:

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha\Big(R_{t+1} + Q_2\big(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t)\Big)$$

- Action selections are (say) ε-greedy with respect to the sum of Q_1 and Q_2

Hado van Hasselt 2010

Double Q-Learning

Initialize $Q_1(s,a)$ and $Q_2(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily
Initialize $Q_1(\textit{terminal-state}, \cdot) = Q_2(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q_1 and Q_2 (e.g., ε-greedy in $Q_1 + Q_2$)
        Take action A, observe R, S'
        With 0.5 probabilility:

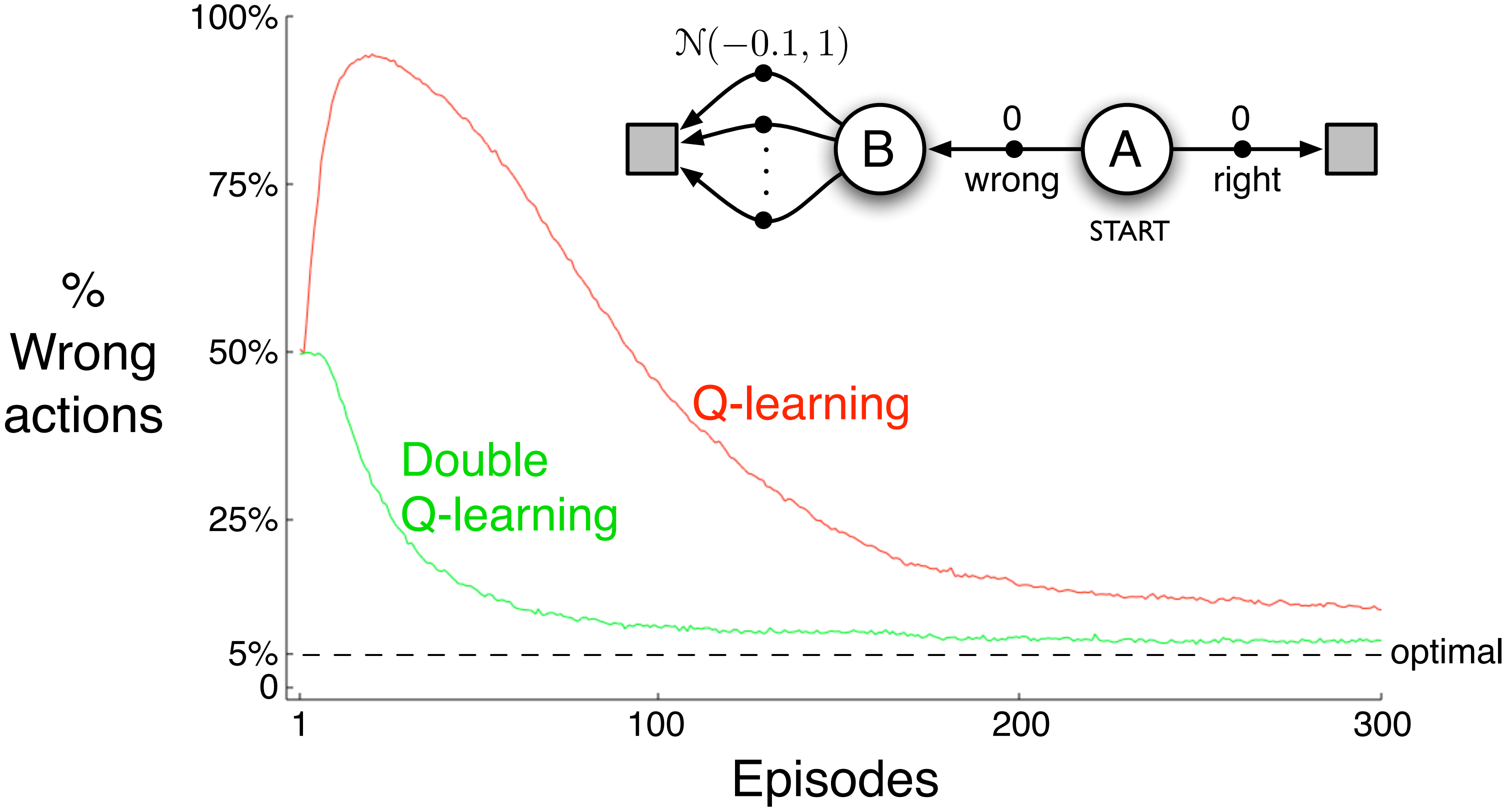
$$Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big(R + \gamma Q_2\big(S', \arg\max_a Q_1(S',a)\big) - Q_1(S,A)\Big)$$

        else:

$$Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big(R + \gamma Q_1\big(S', \arg\max_a Q_2(S',a)\big) - Q_2(S,A)\Big)$$

        $S \leftarrow S'$;
    until S is terminal

Example of Maximization Bias



Double Q-learning:

$$Q_1(S_t, A_t) \leftarrow Q_1(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q_2\big(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t) \Big]$$

Summary

- Extend prediction to control by employing some form of GPI
  - On-policy control: Sarsa, Expected Sarsa
  - Off-policy control: Q-learning, Expected Sarsa
- Avoiding maximization bias with Double Q-learning

Markov Process

A Markov process is a memoryless random process, i.e. a sequence of random states $S_1, S_2, ...$ with the Markov property.

Definition

A *Markov Process* (or *Markov Chain*) is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix,
  $\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$

State Transition Matrix

For a Markov state s and successor state s', the *state transition probability* is defined by

$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

State transition matrix $\mathcal{P}$ defines transition probabilities from all states s to all successor states s',

$$\mathcal{P} = \textit{from} \overset{\textit{to}}{\begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix}}$$

where each row of the matrix sums to 1.

Markov Reward Process

A Markov reward process is a Markov chain with values.

Definition

A *Markov Reward Process* is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{P}$ is a state transition probability matrix,
  $\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}\left[R_{t+1} \mid S_t = s\right]$
- γ is a discount factor, $\gamma \in [0, 1]$

Bellman Equation in Matrix Form

The Bellman equation can be expressed concisely using matrices,

$$v = \mathcal{R} + \gamma \mathcal{P} v$$

where v is a column vector with one entry per state

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{11} & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

Solving the Bellman Equation

- The Bellman equation is a linear equation
- It can be solved directly:

$$
\begin{aligned}
v &= \mathcal{R} + \gamma \mathcal{P} v \\
(I - \gamma \mathcal{P})\, v &= \mathcal{R} \\
v &= (I - \gamma \mathcal{P})^{-1} \mathcal{R}
\end{aligned}
$$

- Computational complexity is $O(n^3)$ for n states
- Direct solution only possible for small MRPs
- There are many iterative methods for large MRPs, e.g.
  - Dynamic programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning

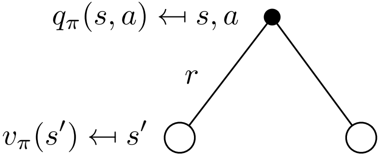
Policies (2)

- Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π
- The state sequence $S_1, S_2, ...$ is a Markov process $\langle \mathcal{S}, \mathcal{P}^\pi \rangle$
- The state and reward sequence $S_1, R_2, S_2, ...$ is a Markov reward process $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
- where

$$\mathcal{P}^\pi_{s,s'} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}^a_{ss'}$$

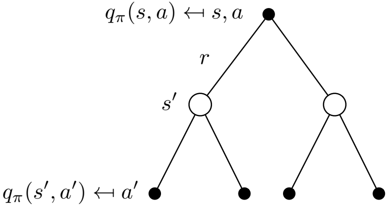
$$\mathcal{R}^\pi_s = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}^a_s$$

Bellman Expectation Equation for Q^π



$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

Bellman Expectation Equation for q_π (2)



$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

Value Function Space

- Consider the vector space $\mathcal{V}$ over value functions
- There are $|\mathcal{S}|$ dimensions
- Each point in this space fully specifies a value function $v(s)$
- What does a Bellman backup do to points in this space?
- We will show that it brings value functions *closer*
- And therefore the backups must converge on a unique solution

Value Function ∞-Norm

- We will measure distance between state-value functions u and v by the ∞-norm
- i.e. the largest difference between state values,

$$||u - v||_\infty = \max_{s \in \mathcal{S}} |u(s) - v(s)|$$

Bellman Expectation Backup is a Contraction

- Define the *Bellman expectation backup operator* T^π,

$$T^\pi(v) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v$$

- This operator is a γ-contraction, i.e. it makes value functions closer by at least γ,

$$\begin{aligned} ||T^\pi(u) - T^\pi(v)||_\infty &= ||\left(\mathcal{R}^\pi + \gamma \mathcal{P}^\pi u\right) - \left(\mathcal{R}^\pi + \gamma \mathcal{P}^\pi v\right)||_\infty \\ &= ||\gamma \mathcal{P}^\pi (u - v)||_\infty \\ &\leq ||\gamma \mathcal{P}^\pi ||u - v||_\infty||_\infty \\ &\leq \gamma ||u - v||_\infty \end{aligned}$$

Contraction Mapping Theorem

Theorem (Contraction Mapping Theorem)

For any metric space $\mathcal{V}$ that is complete (i.e. closed) under an operator $T(v)$, where T is a γ-contraction,

- *T converges to a unique fixed point*
- *At a linear convergence rate of γ*

Convergence of Iter. Policy Evaluation and Policy Iteration

- The Bellman expectation operator T^π has a unique fixed point
- v_π is a fixed point of T^π (by Bellman expectation equation)
- By contraction mapping theorem
- Iterative policy evaluation converges on v_π
- Policy iteration converges on v_*

Bellman Optimality Backup is a Contraction

- Define the *Bellman optimality backup operator* T^*,

$$T^*(v) = \max_{a \in \mathcal{A}} \mathcal{R}^a + \gamma \mathcal{P}^a v$$

- This operator is a γ-contraction, i.e. it makes value functions closer by at least γ (similar to previous proof)

$$||T^*(u) - T^*(v)||_\infty \leq \gamma ||u - v||_\infty$$

Convergence of Value Iteration

- The Bellman optimality operator T^* has a unique fixed point
- v_* is a fixed point of T^* (by Bellman optimality equation)
- By contraction mapping theorem
- Value iteration converges on v_*