Machine Learning for Industry

Jose M. Peña
STIMA, IDA, LiU

Lecture Block 7: Reinforcement Learning: Q-Learning Algorithm

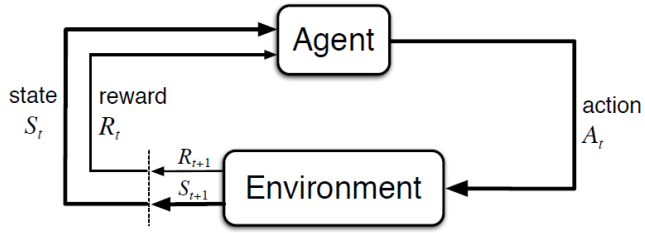
Contents

- Learning through Interaction
- Markov Decision Processes
- Bellman Equations
- Value Iteration
- Policy Iteration
- Monte Carlo Methods
- Temporal-Difference Learning
- Q-Learning and Sarsa

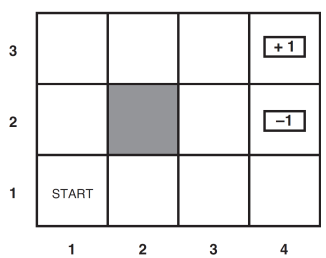
Literature

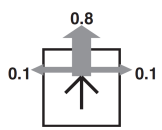
- Main source
  - Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. The MIT Press, 2018. Chapters 1-7. **Available online**.
- Additional source
  - Russel, S. and Norvig, P. *Artifical Intelligence: A Modern Approach*. Pearson, 2010. Chapters 16, 17 and 21.

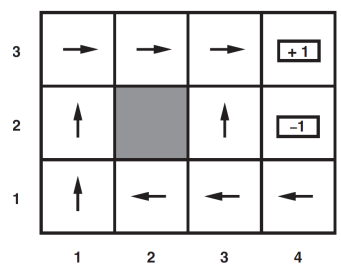
Learning through Interaction

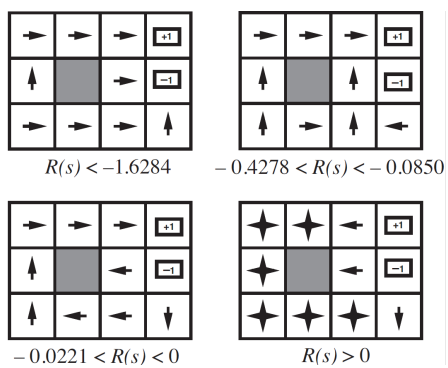


- Agent: The learner and decision maker.
- Environment: The agent interacts with it.
  - State: State of the agent and the environment.
  - Action: The agent decides the next action on the basis of the current state.
  - Reward: Numerical response to the action chosen by the agent. The agent aims to learn how to act so as to maximize the cumulative reward.
  - Trajectory: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_3, R_3, \ldots$
- Example: A robot moves with probability 0.8 in the intended direction, and at the right angles of it otherwise. The reward for non-terminal states is $R(s) = -0.04$. All this is unknown to the robot. Optimal policies shown.







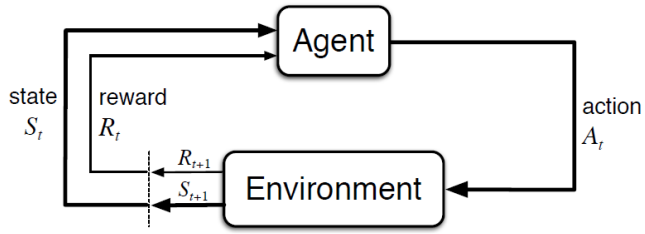


- RL is not supervised learning: Difficult to get examples of desired behavior.
- RL is not unsupervised learning: No aim to find hidden structure.

Learning through Interaction

Example 3.2: Pick-and-Place Robot Consider using reinforcement learning to control the motion of a robot arm in a repetitive pick-and-place task. If we want to learn movements that are fast and smooth, the learning agent will have to control the motors directly and have low-latency information about the current positions and velocities of the mechanical linkages. The actions in this case might be the voltages applied to each motor at each joint, and the states might be the latest readings of joint angles and velocities. The reward might be +1 for each object successfully picked up and placed. To encourage smooth movements, on each time step a small, negative reward can be given as a function of the moment-to-moment "jerkiness" of the motion. ■

Markov Decision Processes



- We assume that the agent-environment interaction follows a finite Markov decision process:
  - Finite sets of states, actions and rewards. Fully observable state.
  - Markovian and stationary transition model:

$$p(S_t, R_t | S_{0:t-1}, A_{0:t-1}, R_{1:t-1}) = p(S_t, R_t | S_{t-1}, A_{t-1}) = p(S', R | S, A).$$

- The transition model is typically unknown to the agent. Note the **randomness** of the next state and reward. Note also the **effect** of the action on the next state.
- The objective of the agent is to learn how to act so as to maximize the expected discounted return:

$$E[G_t] = E\Big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\Big] = E[R_{t+1} + \gamma G_{t+1}]$$

  where $0 < \gamma \leq 1$ describes our preference between present and future rewards. Note that the expectation is finite if $\gamma < 1$. Note also the infinite horizon. For episodic tasks, $\gamma = 1$ and $R_{t+k+1} = 0$ for all $t + k + 1 > T$.
- To that end, the agent has to learn a **policy** $\pi(a|s)$, i.e. the **probability** of doing action a in state s.

Markov Decision Processes

- The **state value** function $v_\pi(s)$ is the expected return of following policy π starting from state s:

$$v_\pi(s) = E[G_t|S_t = s] = \sum_a \pi(a|s) E[G_t|S_t = s, A_t = a] = \sum_a \pi(a|s) q_\pi(s, a).$$

- The **action value** function $q_\pi(s, a)$ is the expected return of doing action a in state s and then following policy π:

$$q_\pi(s, a) = E[G_t|S_t = s, A_t = a] = E[R_{t+1} + \gamma G_{t+1}|s, a] = \sum_{s', r} p(s', r|s, a)(r + \gamma v_\pi(s')).$$

- We can now redefine the objective of the agent as learning a policy π_* such that

$$v_*(s) \geq v_\pi(s) \text{ for all } \pi, s.$$

For MDPs, there is always at least one such optimal policy. In other words,

$$v_*(s) = max_\pi v_\pi(s) \text{ for all } s, \text{ and } q_*(s, a) = max_\pi q_\pi(s, a) \text{ for all } s, a.$$

Bellman Equations

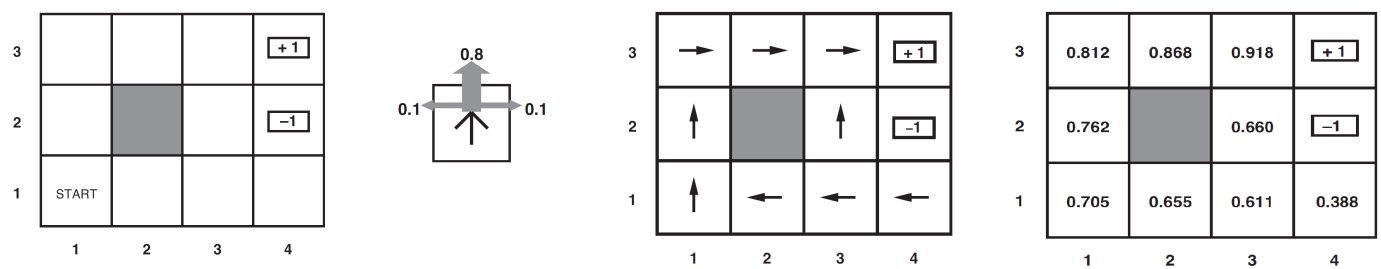
- The state value function satisfies a recursive relationship known as **Bellman equation**:

$$v_\pi(s) = E[G_t|S_t = s] = E[R_{t+1}+\gamma G_{t+1}|s] = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)(r+\gamma v_\pi(s')).$$

- Moreover, v_π is the unique solution to the equations. Note that there are as many equations as unknowns. Since the equations are linear, they can be solved by linear algebra methods in $O(n^3)$. But this requires knowing the transition model.
- Likewise for the action value function:

$$q_\pi(s, a) = \sum_{s',r} p(s', r|s, a)(r + \gamma \sum_{a'} \pi(a'|s')q_\pi(s', a')).$$

- Example: Environment, policy and state values.



Bellman Equations

- The Bellman equations of an optimal policy are

$$v_*(s) = \max_a q_*(s,a) = \max_a \sum_{s',r} p(s',r|s,a)(r + \gamma v_*(s'))$$

and

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)(r + \gamma \max_{a'} q_*(s',a')).$$

- As before, v_* and q_* are the unique solutions to these equations. Note however that the equations are now non-linear due to the max operator and, thus, harder to solve. Again, this requires knowing the transition model.
- Once we have v_* or q_*, it is easy to determine an optimal policy:

$$\pi_*(a|s) = \arg\max_a \sum_{s',r} p(s',r|s,a)(r + \gamma v_*(s'))$$

or

$$\pi_*(a|s) = \arg\max_a q_*(s,a).$$

- This route to solve RL problems has two more drawbacks, in addition to requiring knowing the transition model: For large state spaces, solving the Bellman equations is time consuming, and storing the optimal policy as a look-up table is space consuming.
- Note that the optimal policy is deterministic. So, we can consider only deterministic policies without loss of generality, i.e. $\pi(s)$ instead of $\pi(a|s)$.

Value Iteration

- We can avoid solving the Bellman equations for the state values of an optimal policy by turning them into update rules.

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $\quad v \leftarrow V(s)$
| $\quad V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
| $\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$

- VI converges asymptotically for $\gamma < 1$. Since the Bellman optimality equations have a unique solution, VI converges to v_*.
- VI still requires knowing the transition model.

Value Iteration

Example 4.3: Gambler's Problem A gambler has the opportunity to make bets on the outcomes of a sequence of coin flips. If the coin comes up heads, he wins as many dollars as he has staked on that flip; if it is tails, he loses his stake. The game ends when the gambler wins by reaching his goal of \$100, or loses by running out of money. On each flip, the gambler must decide what portion of his capital to stake, in integer numbers of dollars. This problem can be formulated as an undiscounted, episodic, finite MDP. The state is the gambler's capital, $s \in \{1, 2, \ldots, 99\}$ and the actions are stakes, $a \in \{0, 1, \ldots, \min(s, 100 - s)\}$. The reward is zero on all transitions except those on which the gambler reaches his goal, when it is $+1$. The state-value function then gives the probability of winning from each state. A policy is a mapping from levels of capital to stakes. The optimal policy maximizes the probability of reaching the goal. Let p_h denote the probability of the coin coming up heads. If p_h is known, then the entire problem is known and it can be solved, for instance, by value iteration. Figure 4.3 shows the change in the value function over successive sweeps of value iteration, and the final policy found, for the case of $p_h = 0.4$. This policy is optimal, but not unique. In fact, there is a whole family of optimal policies, all corresponding to ties for the argmax action selection with respect to the optimal value function.

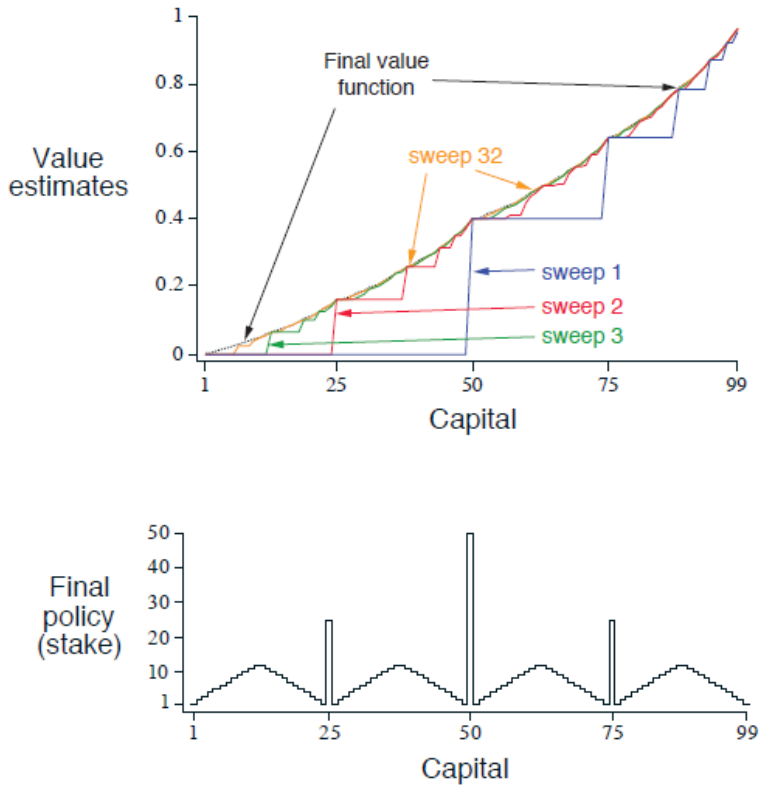


Figure 4.3: The solution to the gambler's problem for $p_h = 0.4$. The upper graph shows the value function found by successive sweeps of value iteration. The lower graph shows the final policy.

Policy Iteration

- Policy evaluation: Turn the Bellman equations into update rules as in VI.

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \,|\, s, \pi(s)) \big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \,|\, s, a) \big[r + \gamma V(s')\big]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Policy Iteration

- Policy improvement: If $q_\pi(s, \pi'(s)) \geq v_\pi(s)$ for all s, then $v_{\pi'}(s) \geq v_\pi(s)$ for all s. Therefore, we can modify π into a better policy π' by taking

$$\pi'(s) = \arg\max_a q_\pi(s, a) \text{ for all } s.$$

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\left[r + \gamma V(s')\right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Policy Iteration

- PI terminates since each iteration improves the policy and there is a finite number of policies. When PI terminates, $\pi(s) = \arg\max_a q_\pi(s, a)$ for all s and, thus, the Bellman optimality equations hold and, thus, π is optimal. Once more, PI requires that the transition model is known.

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Monte Carlo Methods

- **Episodic vs continuing** RL problems: Some problems are naturally divided into episodes of finite (but maybe different) length, e.g. games are divided into matches. Each episode ends in a terminal state, followed by a reset to the starting state or a random state.
- We will see Monte Carlo methods just for episodic tasks. Only on the completion of an episode are value estimates and policies changed. So, in principle, MC methods are not suitable for step-by-step incremental (i.e., online) computation.
- MC methods do **not** require knowing the transition model.

Monte Carlo Methods

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

- To ensure that each state-action pair is selected infinitely often so as to produce reliable estimates, ϵ-soft policies are used, i.e. $\pi(a|s) \geq \epsilon/|\mathcal{A}(s)|$. In particular, ϵ-greedy policies are used: Choose the action with maximal estimated value with probability $1-\epsilon$, and a random one with probability ϵ.
- The algorithm converges asymptotically to the optimal ϵ-greedy policy.

Monte Carlo Methods

- The previous method is called **on-policy** because it evaluates and improves the same policy that is used to make decisions and generate the episodes.
- The problem with on-policy methods is that they have to behave non-optimally to explore all the actions and find the optimal ones. This makes them converge to a **near-optimal** policy that still explores.
- **Off-policy** methods solve this problem by using two policies: They learn about a (deterministic) target policy π, whereas they use a (stochastic) behavior policy b to generate episodes. They require coverage, i.e. if $\pi(a|s) > 0$ then $b(a|s) > 0$.
- The probability of a trajectory under policy π starting from state S_t is

$$p(A_t, S_{t+1}, A_{t+1}, \ldots, S_T | S_t, A_{t:T-1} \sim \pi) = \prod_{k=t}^{T-1} \pi(A_k|S_k) p(S_{k+1}|S_k, A_k).$$

- The relative probability of the trajectory under the target policy π and the behavior policy b (a.k.a. importance sampling ratio) is

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

  which is independent of the transition model.

Monte Carlo Methods

- The relative probability of the trajectory under the target policy π and the behavior policy b (a.k.a. importance sampling ratio) is

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

  which is independent of the transition model.

- Then,

$$v_b(s) = E[G_t|S_t = s] \text{ and } v_\pi(s) = E[\rho_{t:T-1}\, G_t|S_t = s]$$

  because

$$p(A_t, S_{t+1}, R_{t+1}, A_{t+1}, \ldots, S_T, R_T|S_t, A_{t:T-1} \sim \pi)$$

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

$$= \rho_{t:T-1}\, p(A_t, S_{t+1}, R_{t+1}, A_{t+1}, \ldots, S_T, R_T|S_t, A_{t:T-1} \sim b).$$

- We can estimate $v_\pi(s)$ as follows (a.k.a. weighted importance sampling):

$$V(s) = \sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1}\, G_t \Big/ \sum_{t\in\mathcal{T}(s)} \rho_{t:T(t)-1}$$

  where $\mathcal{T}(s)$ are the time steps in which s is visited, $T(t)$ is the termination step of the episode containing step t, and G_t is the return from t to $T(t)$. We number the time steps increasingly across episode boundaries.

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

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

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

- Episode-by-episode incremental implementation to minimize storage space.
- The algorithm converges asymptotically to π_*, although the actions are selected according to b.

Temporal-Difference Learning

- Note that if s were always followed by s' and r, then $v_\pi(s) = r + \gamma v_\pi(s')$ by the Bellman equation and, thus, $0 = r + \gamma v_\pi(s') - v_\pi(s)$. We can try to enforce this constraint by executing π one step from s and, then, update the estimate of $v_\pi(s)$ as

$$v_\pi(s) \leftarrow v_\pi(s) + \alpha(r + \gamma v_\pi(s') - v_\pi(s))$$

  where $\alpha > 0$ is the learning rate. Finally, repeat. This method converges asymptotically if e.g. $\alpha(t) = O(1/N(s,t))$. This result also holds for stochastic transition models, since the number of times that s is followed by s' in the sampled episodes is proportional to the transition probability.
- TD learning does **not** require knowing the transition model.

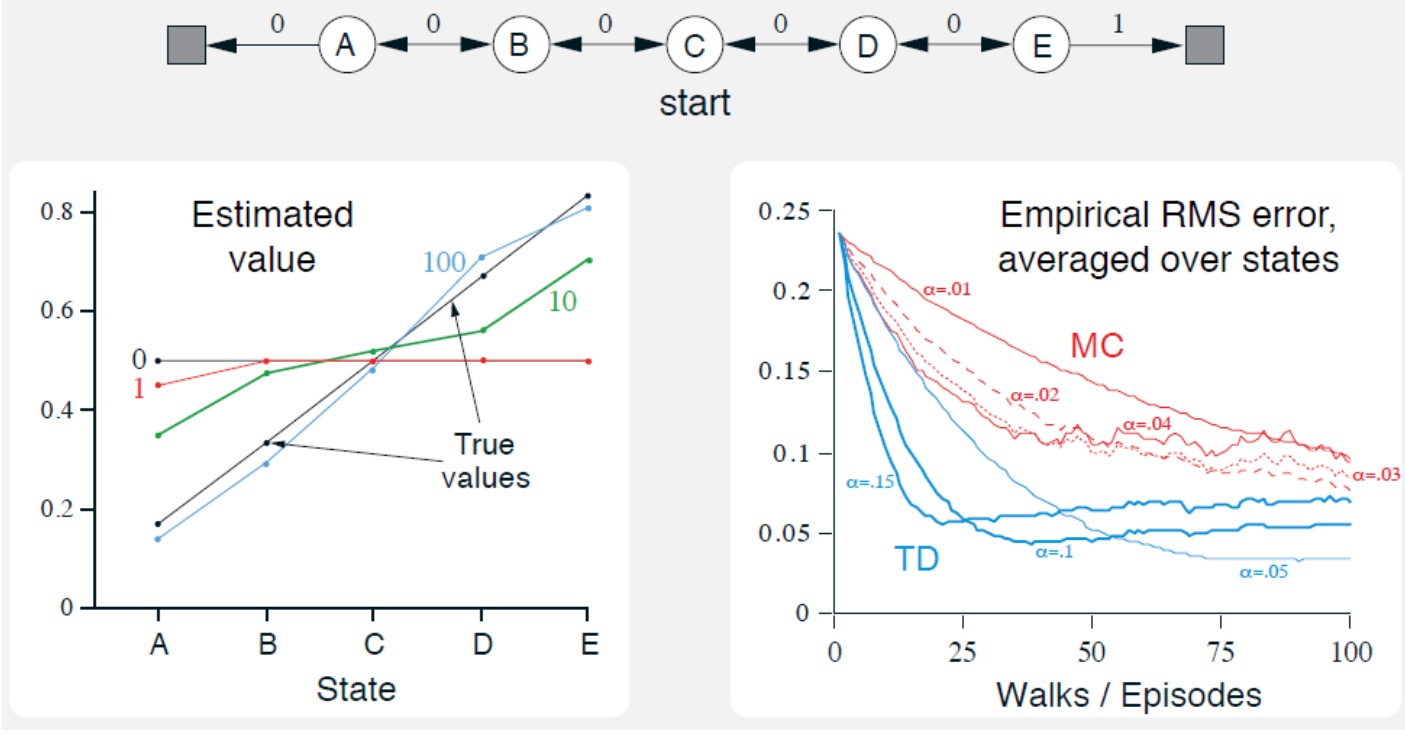
Tabular TD(0) for estimating v_π

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 $V(terminal) = 0$

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

Temporal-Difference Learning

- TD learning updates the estimate $V(S)$ based on another estimate $V(S')$ (a.k.a. **bootstrapping**), unlike the MC methods which wait until the end of an episode to update the estimate.
- Therefore, TD learning is step-by-step incremental, unlike MC methods that are episode-by-episode incremental. This suits continuing tasks or episodic tasks with long episodes. It also seems to make TD learning converge faster in practice.
- Example: Consider the transition model below. No actions. Labels are rewards. Equal probability of moving right or left. The boxes are terminal states. Moreover, $\gamma = 1$.



Q-Learning and Sarsa

- **Q-learning**: Off-policy adaptation of TD learning from state values to optimal action values. Specifically, if s and a were always followed by s' and r, then $q_*(s,a) = r + \gamma \max_{a'} q_*(s',a')$ by the Bellman optimality equation and, thus, $0 = r + \gamma \max_{a'} q_*(s',a') - q_*(s,a)$ and, thus, we can update the estimate of $q_*(s,a)$ as

$$q_*(s,a) \leftarrow q_*(s,a) + \alpha(r + \gamma \max_{a'} q_*(s',a') - q_*(s,a)).$$

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal,\cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
    $S \leftarrow S'$
  until S is terminal

- Converges asymptotically to q_* if e.g. an ϵ-greedy policy is used to keep updating all the state-action pairs.

Q-Learning and Sarsa

- **Sarsa**: On-policy adaptation of TD learning from state values to action values. Specifically, if s and a were always followed by r, s' and a' (hence the name), then $q_\pi(s,a) = r + \gamma q_\pi(s',a')$ by the Bellman equation and, thus, $0 = r + \gamma q_\pi(s',a') - q_\pi(s,a)$ and, thus, we can update the estimate of $q_\pi(s,a)$ as

$$q_\pi(s,a) \leftarrow q_\pi(s,a) + \alpha(r + \gamma q_\pi(s',a') - q_\pi(s,a)).$$

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

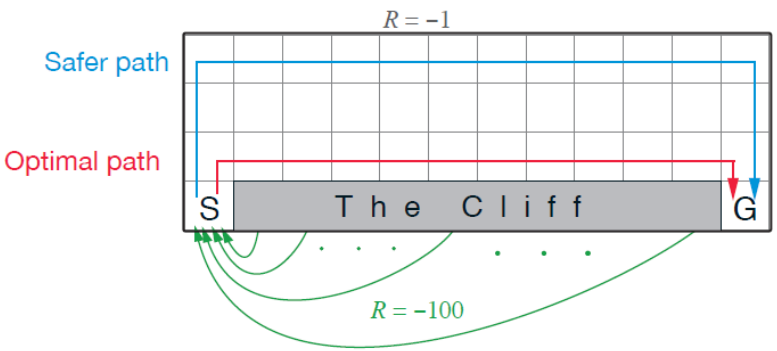
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal,\cdot) = 0$

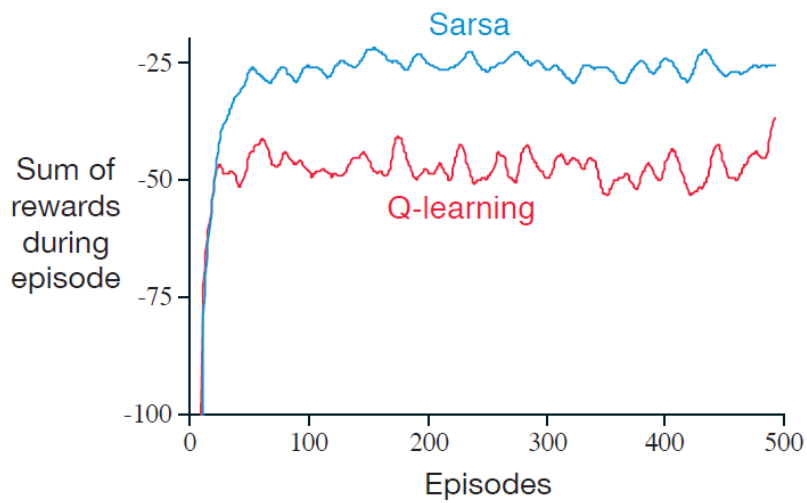
Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

- Converges asymptotically to the optimal ϵ-greedy policy and, thus, to π_* if e.g. $\epsilon = 1/t$.

Q-Learning and Sarsa

- Example: Undiscounted, episodic task, with start and goal states, and up/down/right/left actions. Reward is -1 on all transitions except those marked as "The Cliff" which are -100. Moreover, $\epsilon = 0.1$. Q-learning learns the optimal path, which means falling off the cliff sometimes due to the ϵ-greedy policy. Sarsa learns the longer but safer and more profitable path. If ϵ is gradually reduced, then both methods converge asymptotically to the optimal path.





- Q-learning updates with the best action regardless of the action chosen (off-policy), whereas Sarsa updates with the action selected (on-policy). Q-learning learns for what the agent would like to happen, and Sarsa for what will actually happen, e.g. ignoring or adapting to the adversaries ?

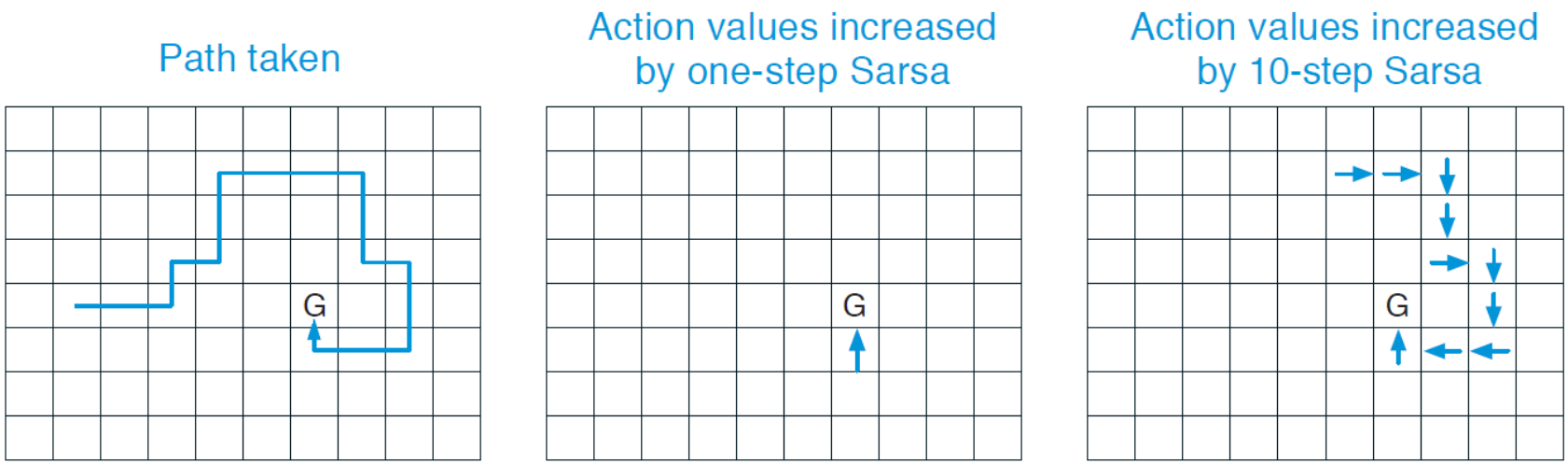
Q-Learning and Sarsa

- The previous Sarsa is a 1-step Sarsa. We can extend it to 2-step Sarsa:

$$q_\pi(s,a) = r + \gamma q_\pi(s',a') = r + \gamma(r' + \gamma q_\pi(s'',a'')) = r + \gamma r' + \gamma^2 q_\pi(s'',a'')$$

$$q_\pi(s,a) \leftarrow q_\pi(s,a) + \alpha(r + \gamma r' + \gamma^2 q_\pi(s'',a'') - q_\pi(s,a)).$$

- n-step Sarsa is actually a family of methods, with 1-step Sarsa and on-policy MC as extreme members. In some cases, intermediate members outperform either of the two extreme members.



Q-Learning and Sarsa

n-step Sarsa for estimating $Q \approx q_*$ or q_π

Initialize $Q(s,a)$ arbitrarily, for all $s \in \mathcal{S}, a \in \mathcal{A}$
Initialize π to be ε-greedy with respect to Q, or to a fixed given policy
Algorithm parameters: step size $\alpha \in (0,1]$, small $\varepsilon > 0$, a positive integer n
All store and access operations (for S_t, A_t, and R_t) can take their index mod $n+1$

Loop for each episode:
  Initialize and store $S_0 \neq$ terminal
  Select and store an action $A_0 \sim \pi(\cdot|S_0)$
  $T \leftarrow \infty$
  Loop for $t = 0, 1, 2, \ldots$:
  | If $t < T$, then:
  | Take action A_t
  | Observe and store the next reward as R_{t+1} and the next state as S_{t+1}
  | If S_{t+1} is terminal, then:
  | $T \leftarrow t+1$
  | else:
  | Select and store an action $A_{t+1} \sim \pi(\cdot|S_{t+1})$
  | $\tau \leftarrow t - n + 1$ (τ is the time whose estimate is being updated)
  | If $\tau \geq 0$:
  | $G \leftarrow \sum_{i=\tau+1}^{\min(\tau+n,T)} \gamma^{i-\tau-1} R_i$
  | If $\tau + n < T$, then $G \leftarrow G + \gamma^n Q(S_{\tau+n}, A_{\tau+n})$ $(G_{\tau:\tau+n})$
  | $Q(S_\tau, A_\tau) \leftarrow Q(S_\tau, A_\tau) + \alpha\left[G - Q(S_\tau, A_\tau)\right]$
  | If π is being learned, then ensure that $\pi(\cdot|S_\tau)$ is ε-greedy wrt Q
  Until $\tau = T - 1$

Summary

- Learning through Interaction
- Markov Decision Processes
- Bellman Equations
- Value Iteration
- Policy Iteration
- Monte Carlo Methods
- Temporal-Difference Learning
- Q-Learning and Sarsa

Thank you

AI COMPETENCE FOR SWEDEN

LINKÖPING UNIVERSITY