Autonomous Networking

Gaia Maselli
Dept. of Computer Science

Today’s plan

- Optimal policy
- Q-learning

Policies

- Up to this point, we've generally talked about a policy as something that is given.
- The policy specifies how an agent behaves.
- Given this way of behaving, we then aim to find the value function.
- But the goal of reinforcement learning is not just to evaluate specific policies.
- Ultimately, *we want to find a policy that obtains as much reward as possible in the long run*

SAPIENZA UNIVERSITÀ DI ROMA

How to find the best possible solution to MDP

↓

How to find the optimal policy

Optimal value function

- To define an optimal policy, we first have to understand what it means for one policy to be better than another

Definition

The **optimal state-value function $v_*(s)$** is the maximum value function over all policies

$$V_*(s) = \max_{\pi} V_{\pi}(s)$$

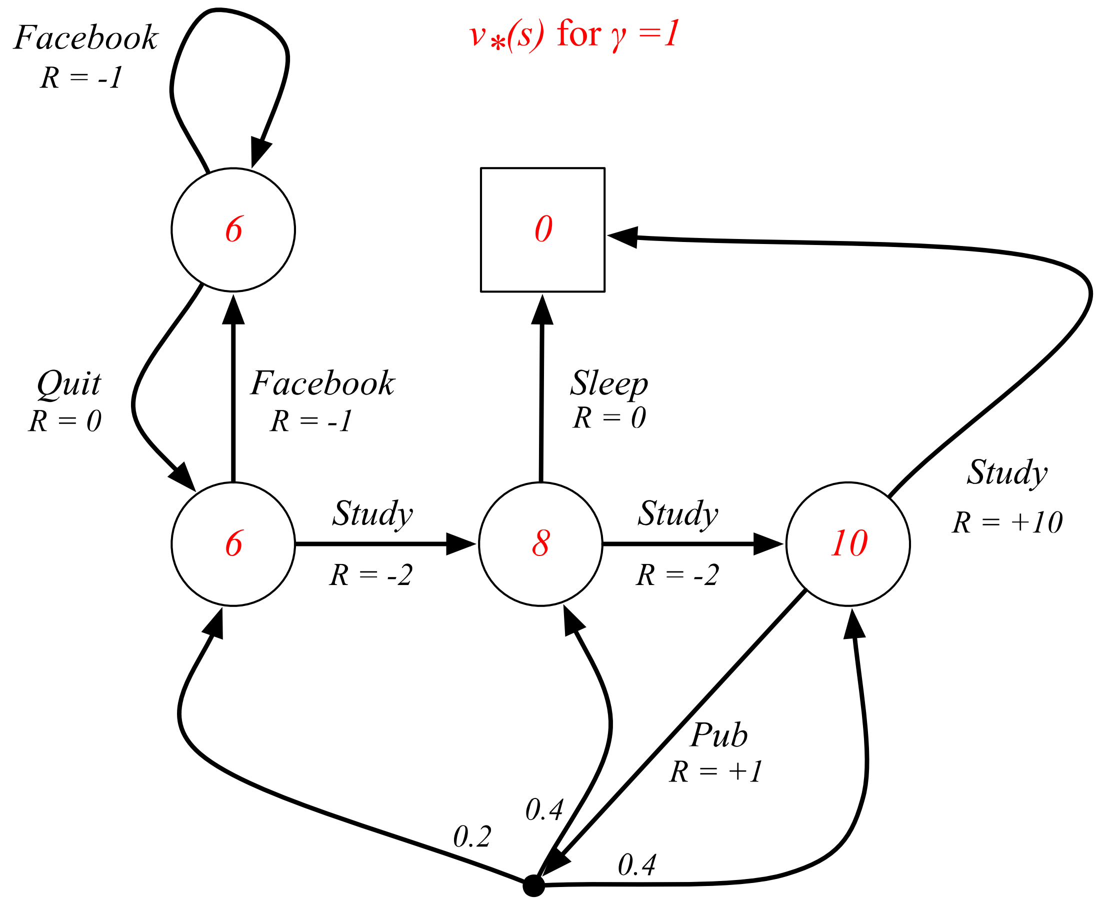
Definition

The **optimal action-value function $q_*(s,a)$** is the maximum action-value function over all policies

$$q_*(s,a) = \max_{\pi} q_{\pi}(s,a)$$

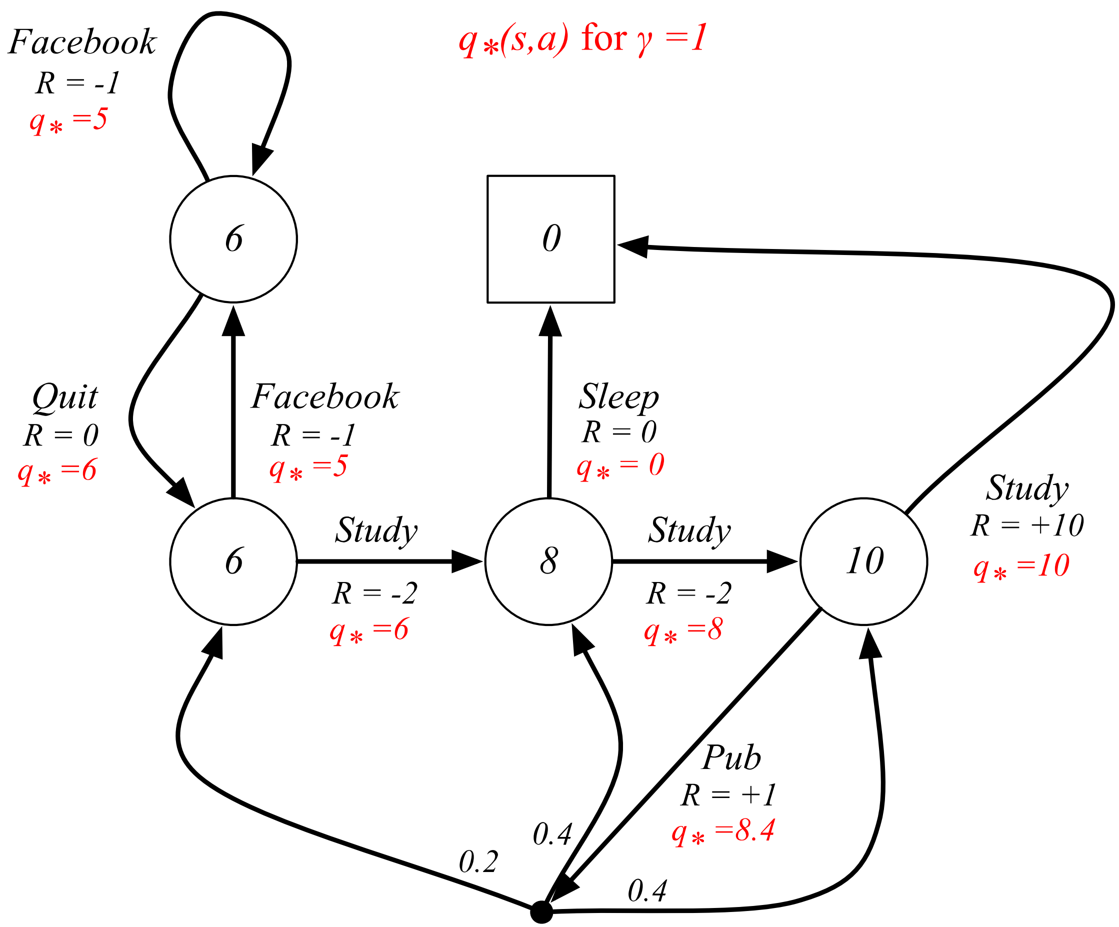
- The optimal value function specifies the best possible performance in the MDP
- An MDP is "solved" when we know the optimal value fn.

Example: Optimal Value Function for Student MDP



- V* says how good is to be in each state
- it does not say how to behave

Example: Optimal Action-Value Function for Student MDP



Optimal policy

- Define a partial ordering over policies

$$\pi \geq \pi' \text{ if } V_{\pi}(s) \geq V_{\pi'}(s), \forall s$$

Theorem

For any Markov Decision Process

- There exists an optimal policy $\pi*$ that is better than or equal to all other policies, $\pi * \geq \pi, \forall \pi$
- All optimal policies achieve the optimal value function, $V_{\pi_*}(s) = V_*(s)$
- All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s, a) = q_*(s, a)$

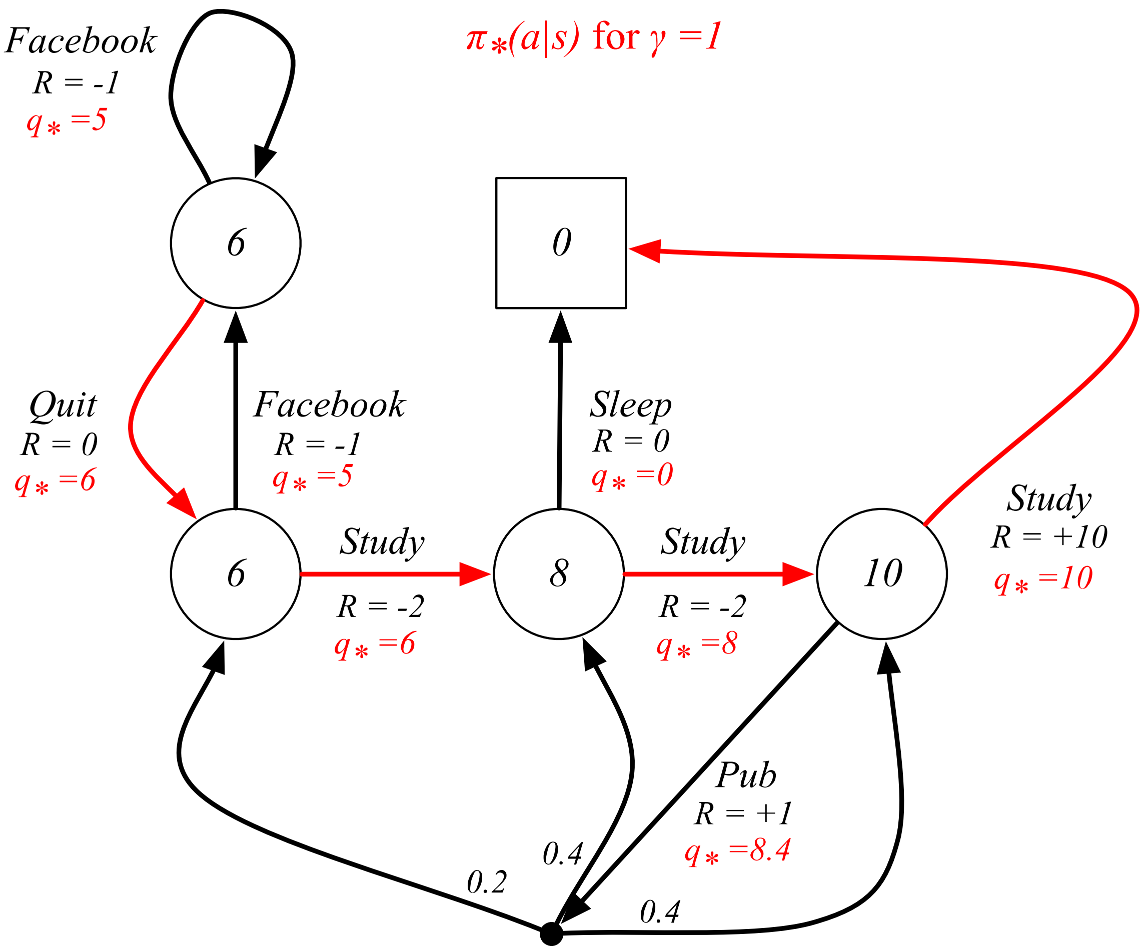
Finding an optimal policy

- An optimal policy can be found by maximising over q∗(s,a)

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, q_*(s,a) \\ 0 & otherwise \end{cases}$$

- There is always a deterministic optimal policy for any MDP
- If we know q∗(s,a), we immediately have the optimal policy

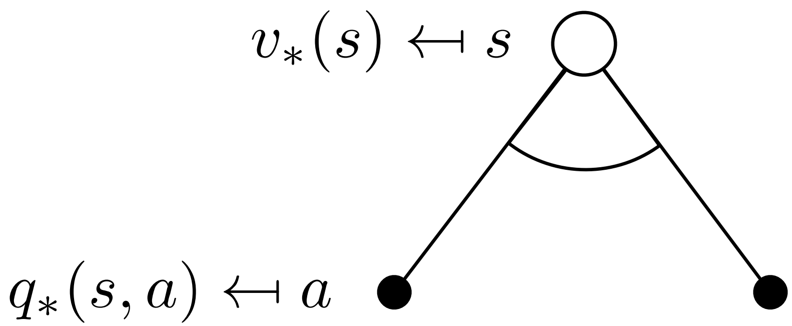
Example: Optimal Policy for Student MDP



How do we get q_* values?

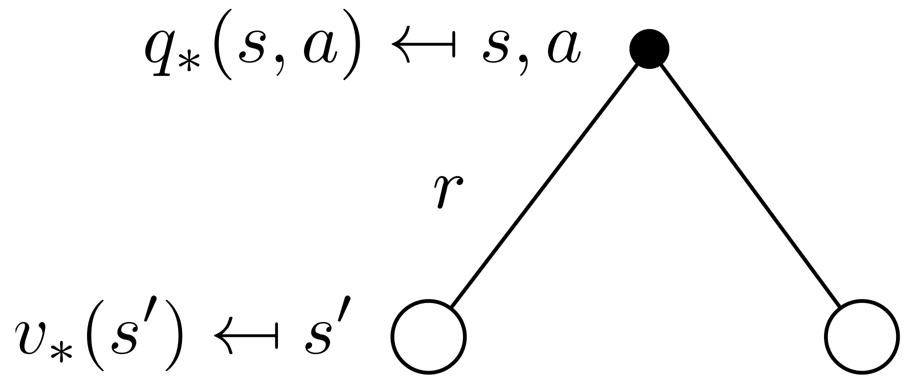
Bellman Optimality Equation for v_*

- The optimal value functions are recursively related by the Bellman optimality equations:



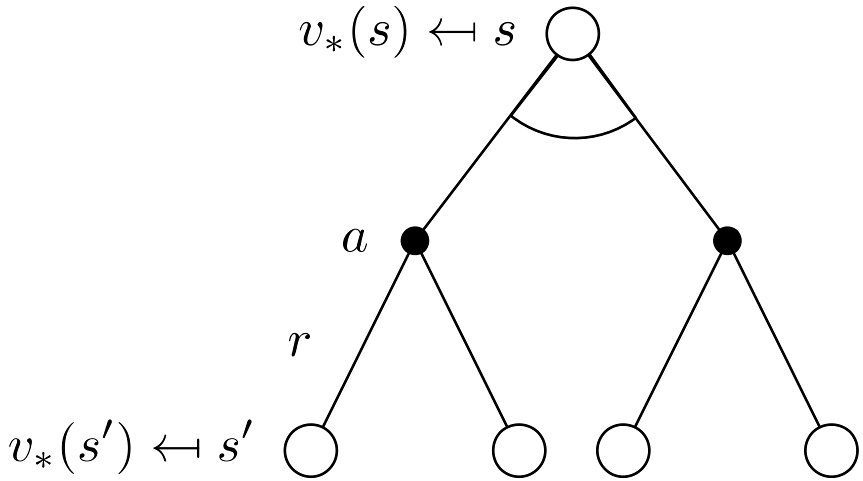
$$v_*(s) = \max_a q_*(s,a)$$

Bellman Optimality Equation for Q*



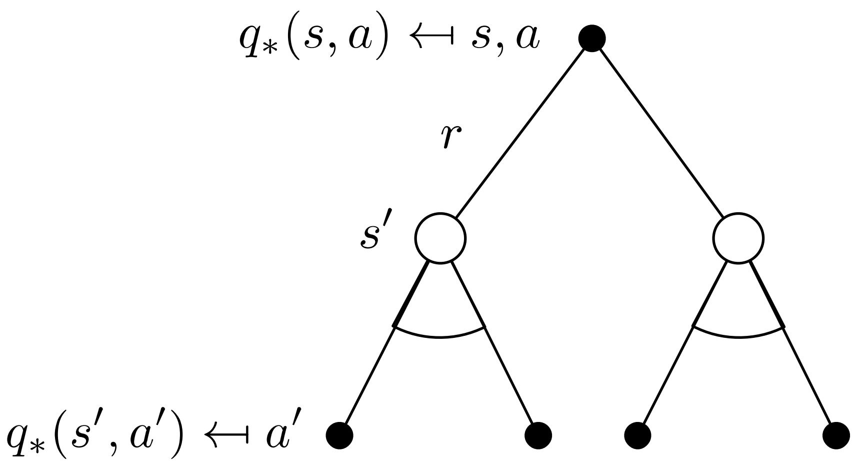
$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

Bellman Optimality Equation for V∗ (2)



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

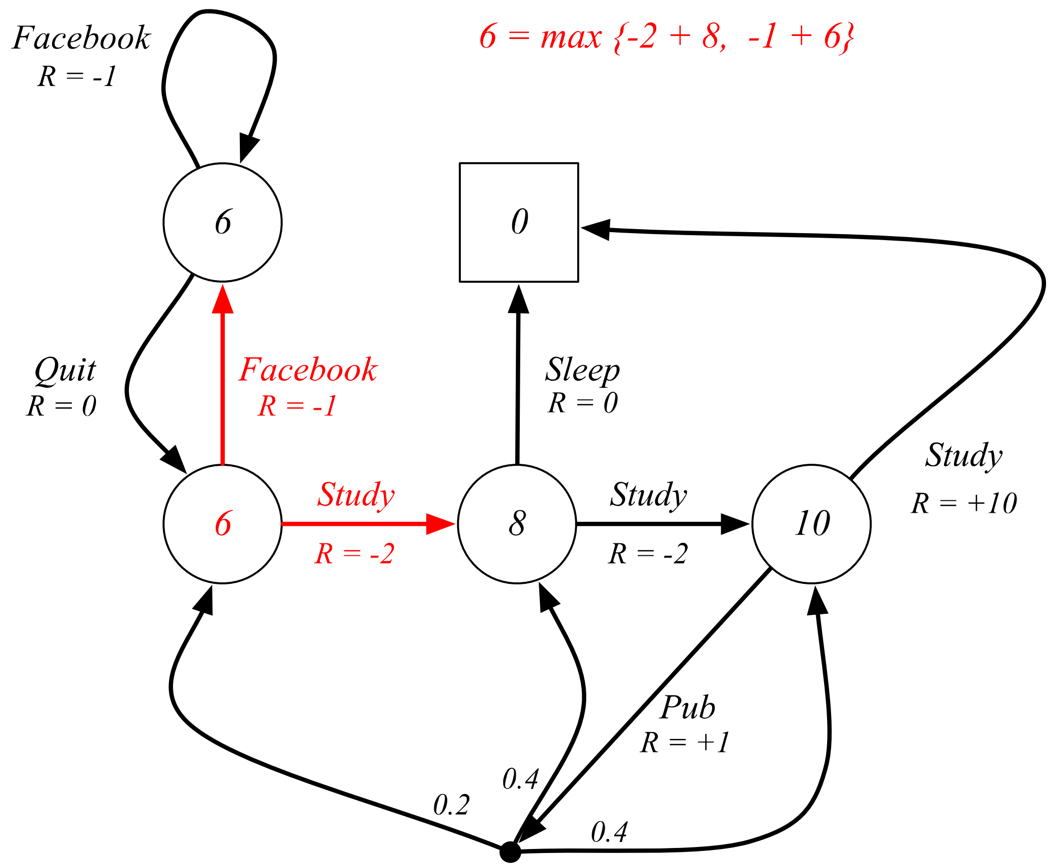
Bellman Optimality Equation for Q* (2)



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

Example: Bellman Optimality Equation in Student MDP

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



Solving the Bellman Optimality Equation

- Bellman Optimality Equation is non-linear
- No closed form solution (in general)
- Many iterative solution methods
  - Value Iteration
  - Policy Iteration
  - **Q-learning**
  - Sarsa

Q-learning

- Temporal Difference (TD) Learning
- Off-policy
- Q-learning (off-policy TD control)

Temporal Difference(TD) learning

- TD methods learn directly from experience
- **Prediction problem**; the problem of estimating the value function v_Π for a given policy Π
- At each step the state value function is updated
- TD error: how much should you adjust the V-value for the previous state

Temporal Difference(TD) learning

- At each step the state value function is updated

$$V(S_t) \leftarrow V(S_t) + \alpha \Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \Big]$$

- This TD method is called TD(0), or one-step TD, because it is a special case of the TD(λ) and n-step TD methods

TD example: driving home

- Each day as you drive home from work, you try to predict how long it will take to get home
- As you wait in traffic, you already know that your initial estimate of 30 minutes was too optimistic. Must you wait until you get home before increasing your estimate for the initial state?

State	*Elapsed Time (minutes)*	*Predicted Time to Go*	*Predicted Total Time*
leaving office, friday at 6	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

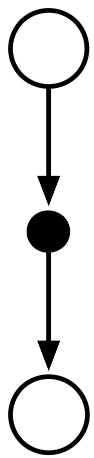
TD(0) algorithm

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 $V(\textit{terminal}) = 0$

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

TD(0) sample update



TD(0)

- The value estimate for the state node at the top of the backup diagram is updated on the basis of the **one sample transition** from it to the immediately following state
- Sample updates are based on a single sample successor rather than on a complete distribution of all possible successors
- TD error

$$\delta_t \doteq R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

Q-learning

- **Control problem**: finding an optimal policy
- Q-learning uses TD prediction for the control problem
  - Applies TD to the Q-value
- Off-policy TD control: the policy used to generate behavior, called the *behavior* policy, may be unrelated to the policy that is evaluated and improved, called the *target* policy

TD applied to Q value

- TD error: how much should you adjust the Q-value for the previous state

Discount factor

$$TD(s_t, a_t) = r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)$$

Immediate reward for the action taken a_t

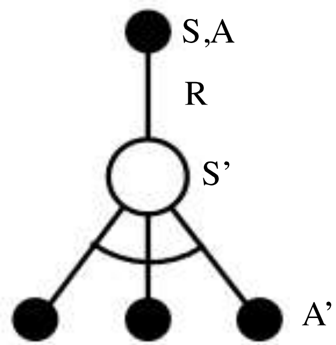
The maximum Q-value available from the current state taking any action

Bellman equation

- The Bellman equation tells us what new value to use as the Q-value for the action taken in the previous state
- Relies on both the old Q-value for the action taken in the previous state and what has been learned after moving to the next state
- Includes a learning rate parameter (α) that defines how quickly Q-values are adjusted

$$Q^{new}(s_t, a_t) = Q^{old}(s_t, a_t) + \alpha\, TD(s_t, a_t)$$

Q-Learning Control Algorithm



$$Q(S, A) \leftarrow Q(S, A) + \alpha \left(R + \gamma \max_{a'} Q(S', a') - Q(S, A) \right)$$

- Q-learning control converges to the optimal action-value function, $Q(s, a) \rightarrow q_*(s, a)$

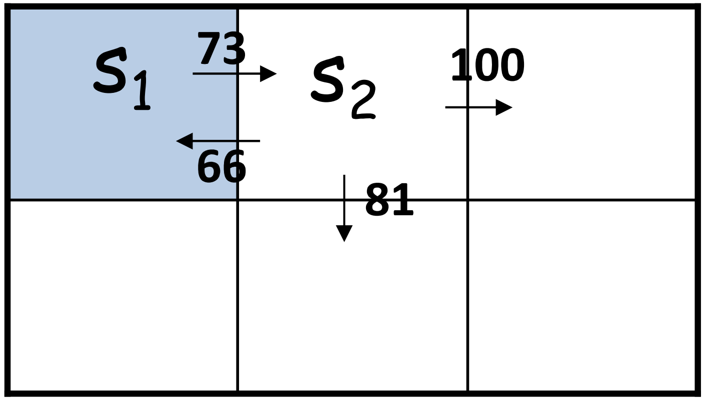
Q-learning algorithm

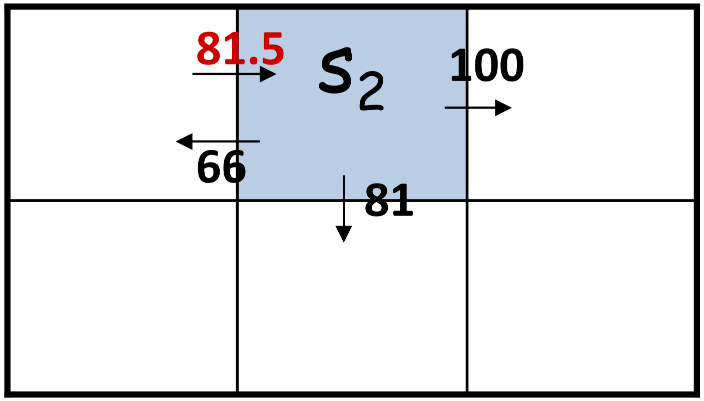
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

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 $Q(\mathit{terminal}, \cdot) = 0$

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

Example





$\gamma = 0.9, \ \ \alpha = 0.5, \ \ r = 0$ for non-terminal states

$$Q(s_1, right) = Q(s_1, right) + \alpha\left(r + \gamma \max_{a'} Q(s_2, a') - Q(s_1, right)\right)$$
$$= 73 + 0.5(0 + 0.9 \max\ \{66,81,100\} - 73)$$
$$= 73 + 0.5(17)$$
$$= \ 81.5$$