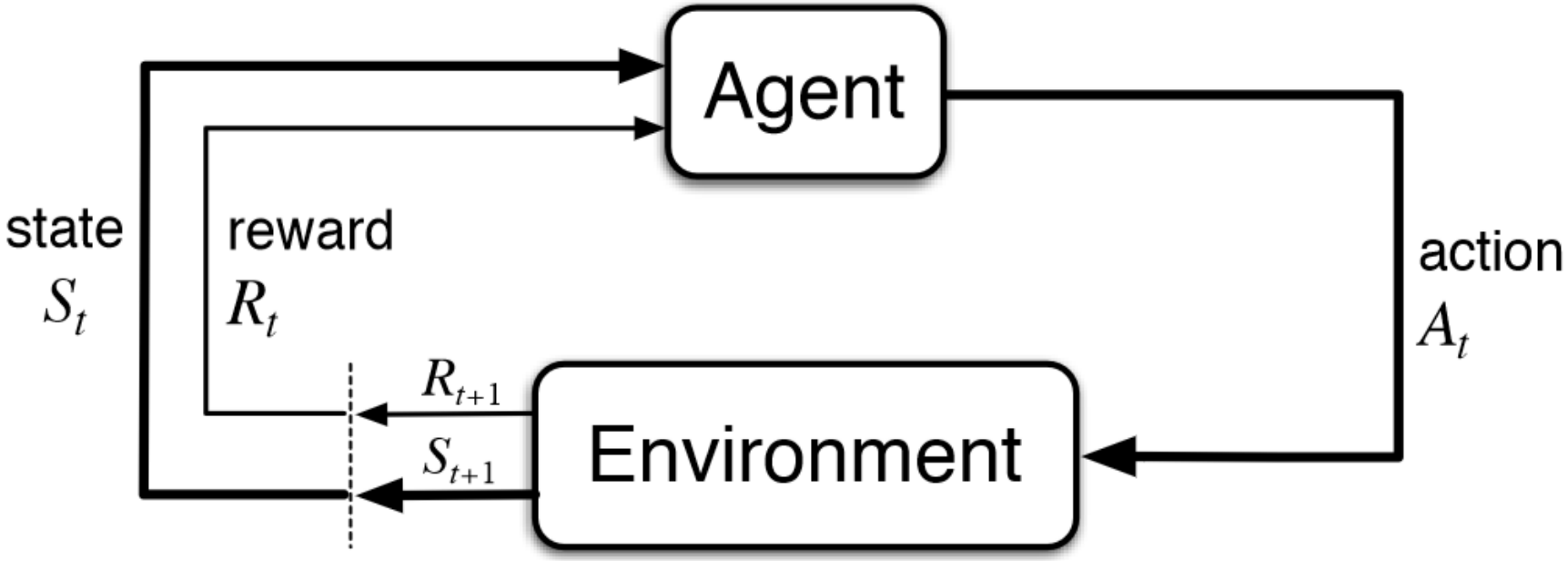
Lecture 1

Markov Decision Process

Value-based methods

Tabular Q-learning

Reinforcement Learning Problem



Source: Sutton & Barto, Introduction to Reinforcement Learning, 2nd Edition

Markov Process (MP)

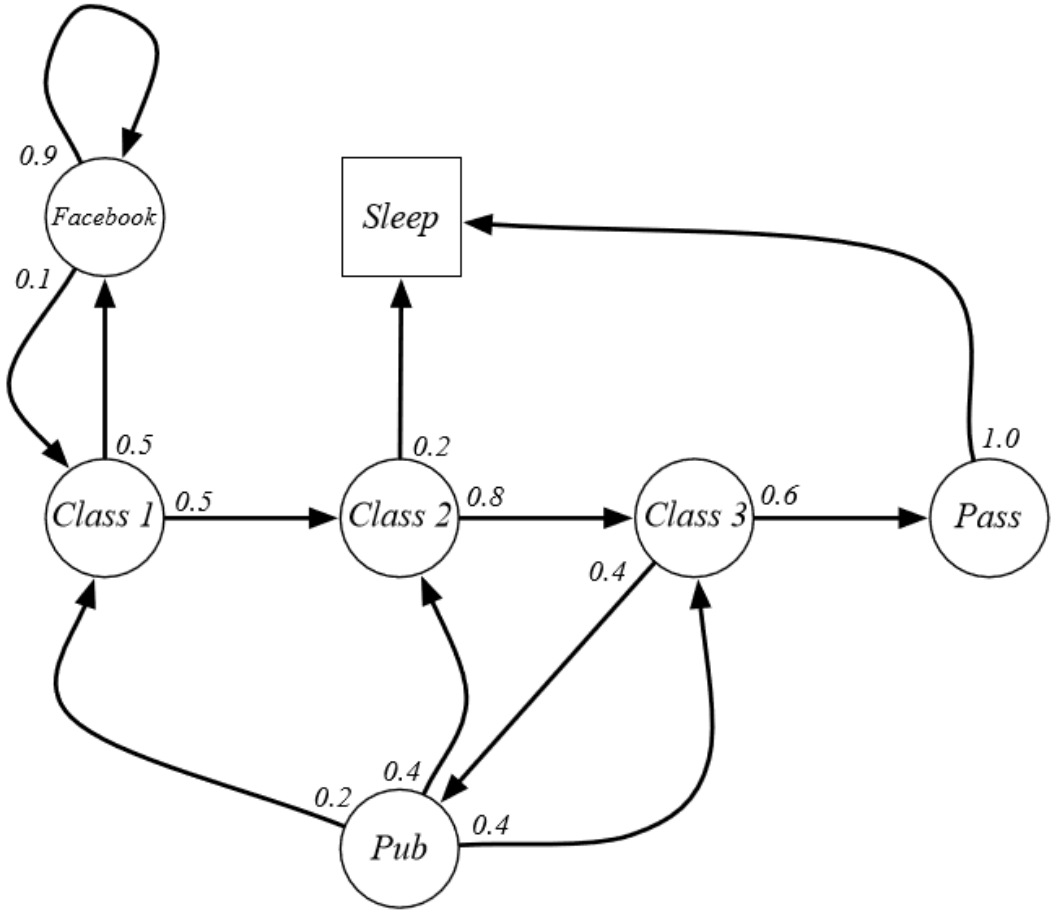
Definition

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

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

Andrey Markov

Markov Process: Example



Source: D. Silver, DeepMind, Lectures

Markov Reward Process (MRP)

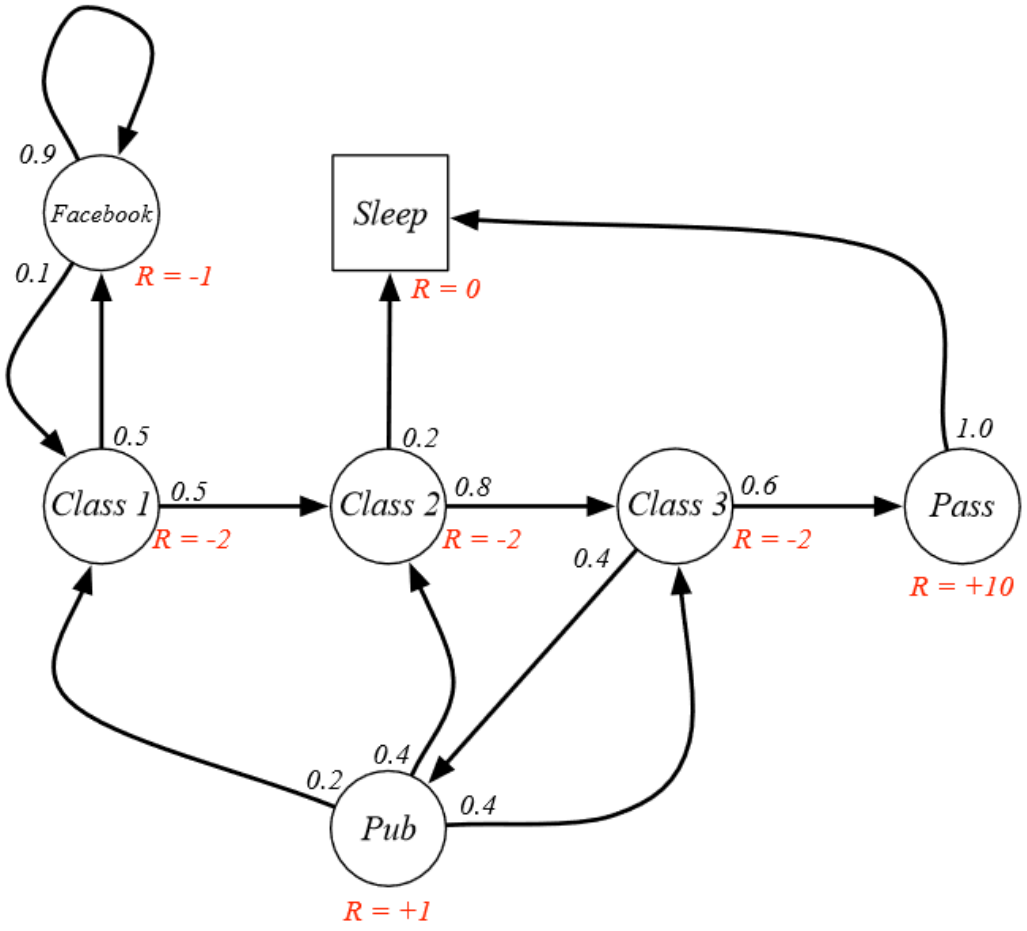
Definition

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

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

Richard Bellman

Markov Reward Process: Example



Source: D. Silver, DeepMind, Lectures

Return

Definition

The *return* G_t is the total discounted reward from time-step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Why discounting?

- Mathematically convenient
- Avoids infinite returns
- Represents uncertainty about the future
- In case of monetary rewards immediate rewards may earn more interest than delayed rewards
- Animal and human behavior demonstrates preference for immediate rewards

Value Function

Definition

The *state value function* $v(s)$ of an MRP is the expected return starting from state s

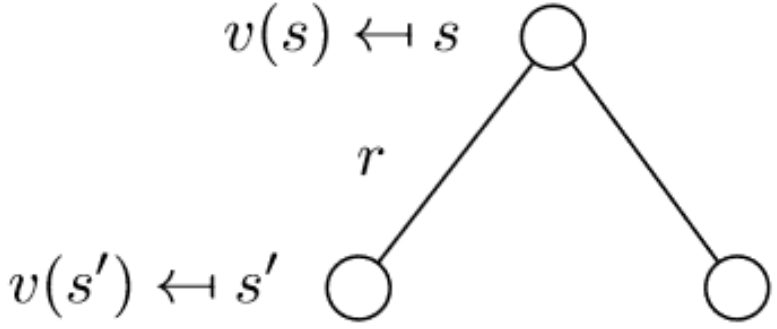
$$v(s) = \mathbb{E}\left[G_t \mid S_t = s\right]$$

Bellman Equation for MRP (1)

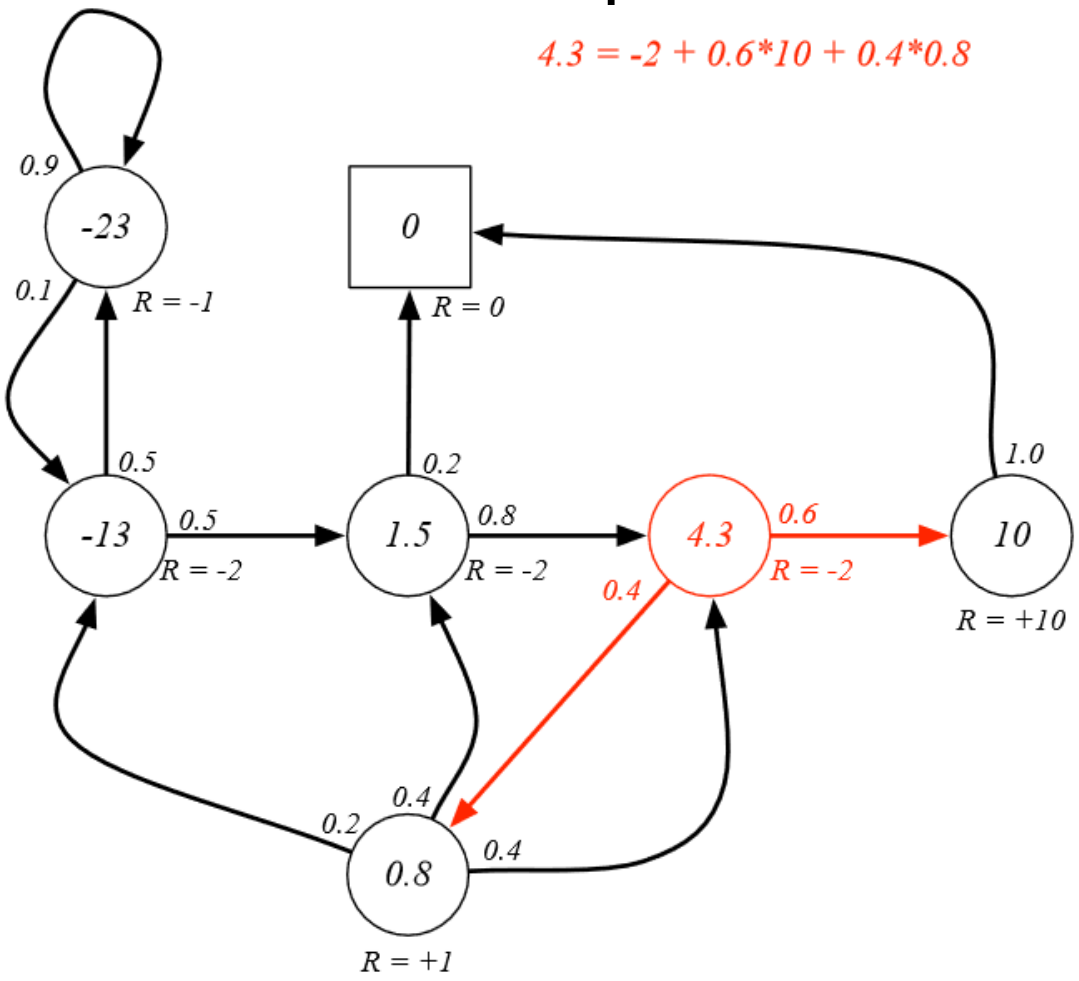
$$
\begin{aligned}
v(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\
&= \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]
\end{aligned}
$$

Bellman Equation for MRP (2)

$$v(s) = \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]$$



Bellman Equation: Example



Source: D. Silver, DeepMind, Lectures

Solving Bellman Equation

- The Bellman Equation is linear equation
- Can be solved in closed form
- Computational complexity $O(n^3)$
- Therefore direct solutions only possible for small MRPs
- Thera iterative methods exist for evaluation

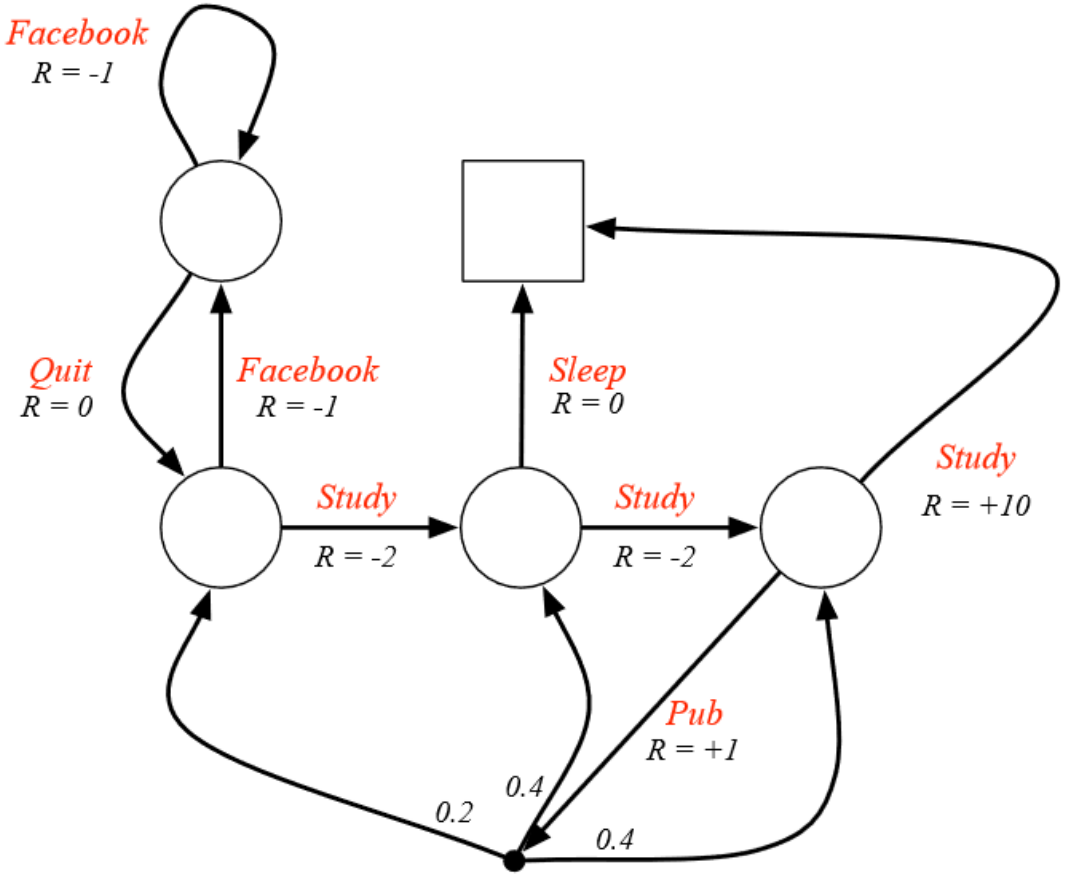
Markov Decision Process

Definition

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

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

Markov Decision Process: Example



Source: D. Silver, DeepMind, Lectures

Policy

Definition

A *policy* π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}\left[A_t = a \mid S_t = s\right]$$

- Policy completely describes agent's behavior
- MDP policies depend on the current state
- Policies are stationary (time-independent)

Value Function

Definition

The *state-value function* $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π

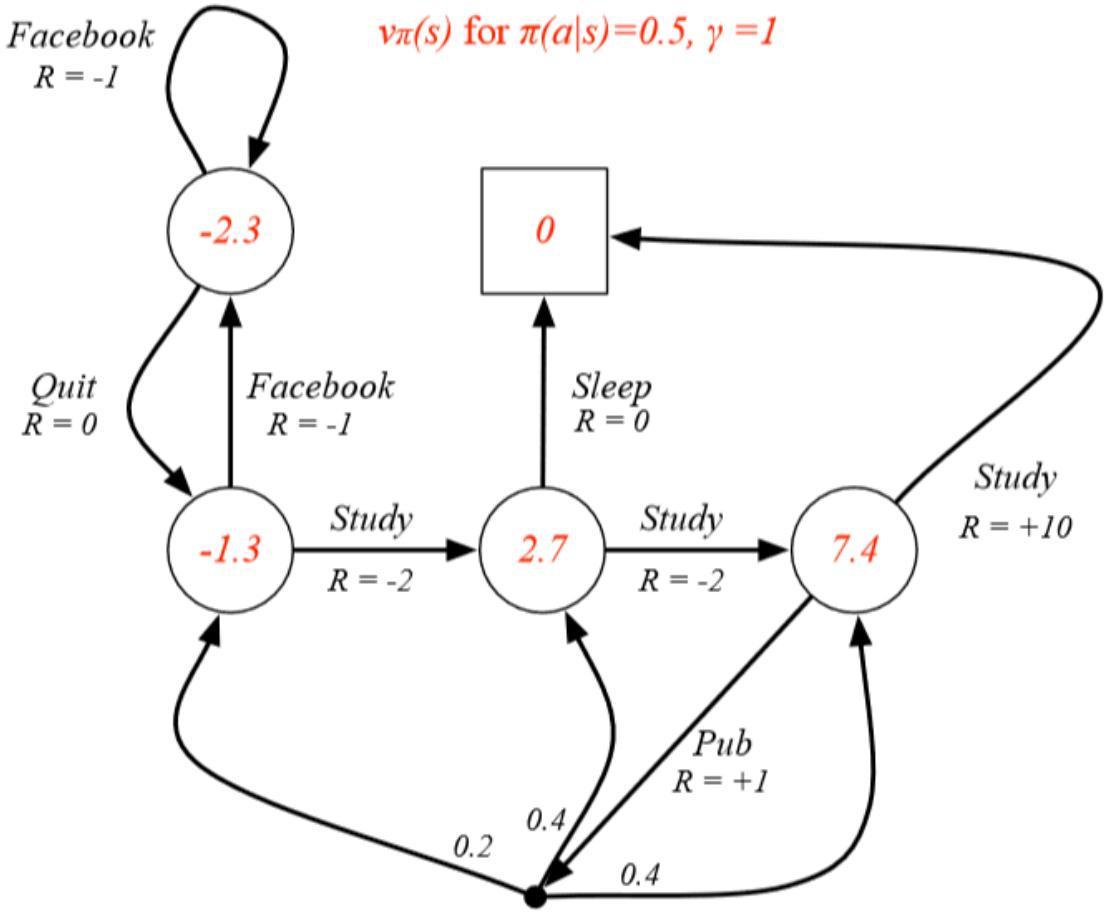
$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

Definition

The *action-value function* $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right]$$

State-Value Function: Example



Source: D. Silver, DeepMind, Lectures

Bellman Expectation Equation

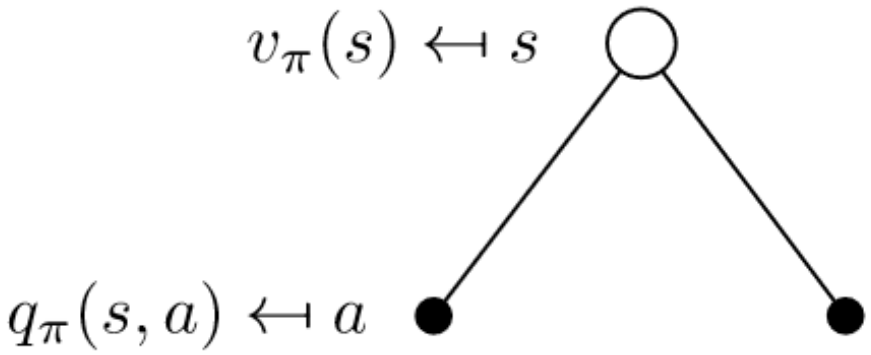
The state-value function can again be decomposed into immediate reward plus discounted value of successor state,

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s \right]$$

The action-value function can similarly be decomposed,

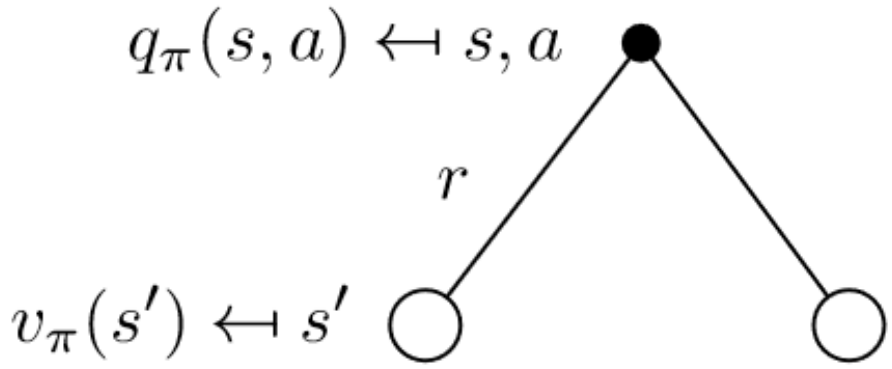
$$q_\pi(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a \right]$$

Bellman Expectation Equation for V^π



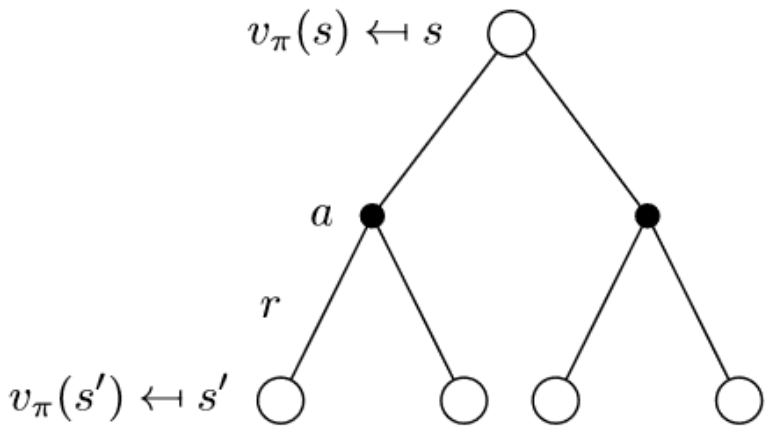
$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

Bellman Expectation Equation for Q^π



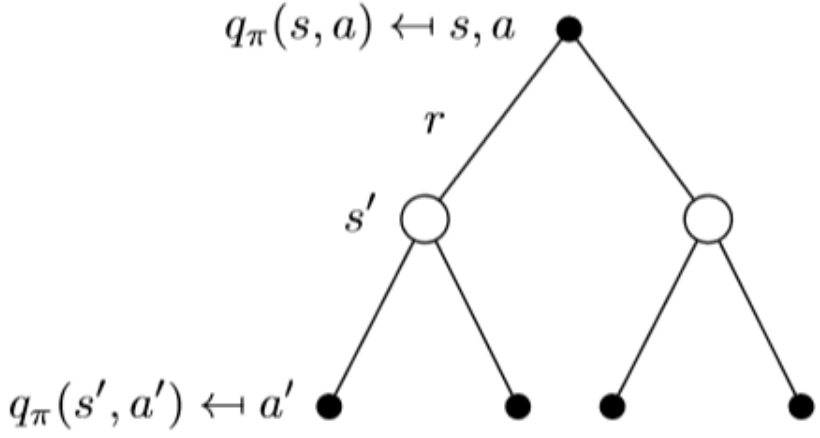
$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

Bellman Expectation Equation for V^π (2)



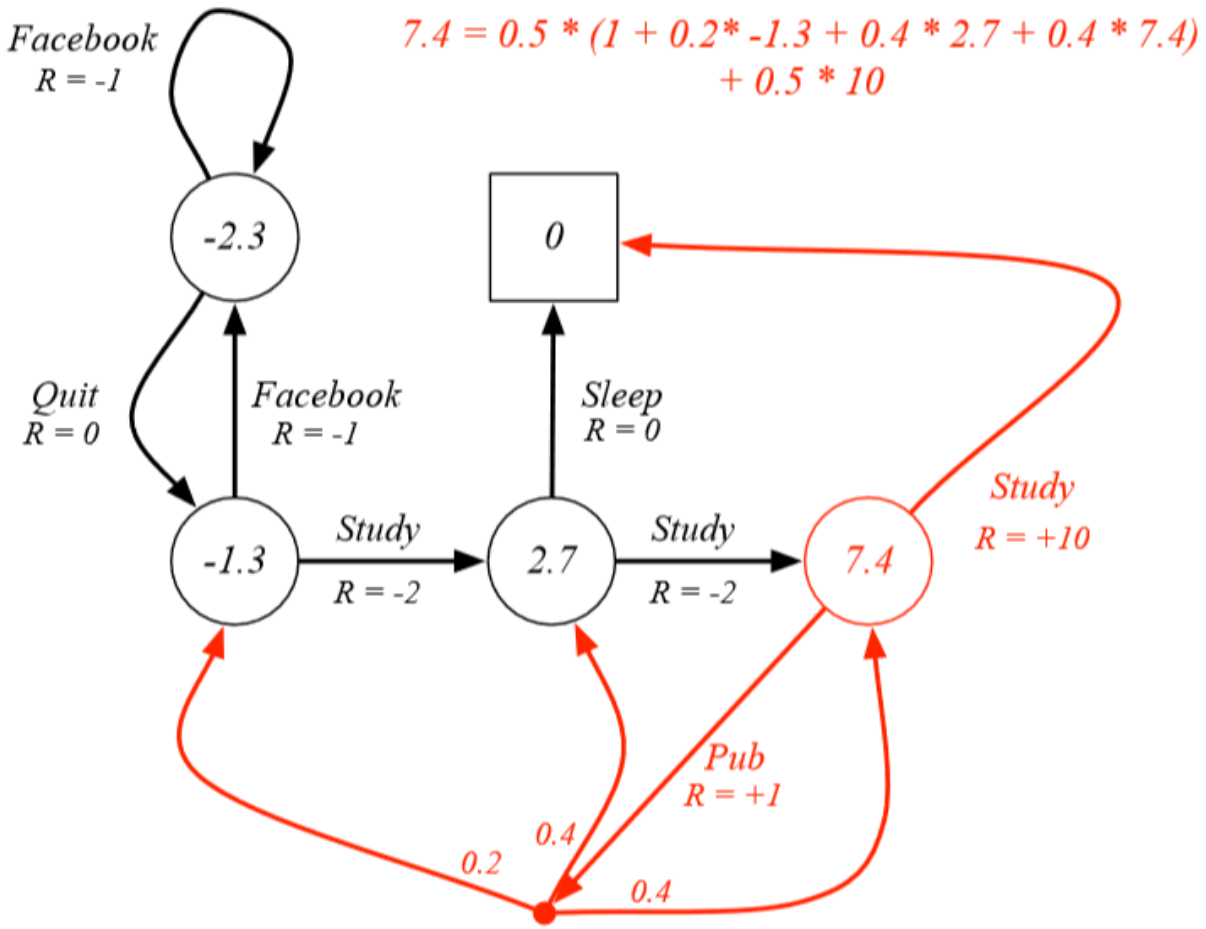
$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

Bellman Expectation Equation for Q^π (2)



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

Bellman Expectation Equation: Example



Source: D. Silver, DeepMind, Lectures

Optimal Value Function

Definition

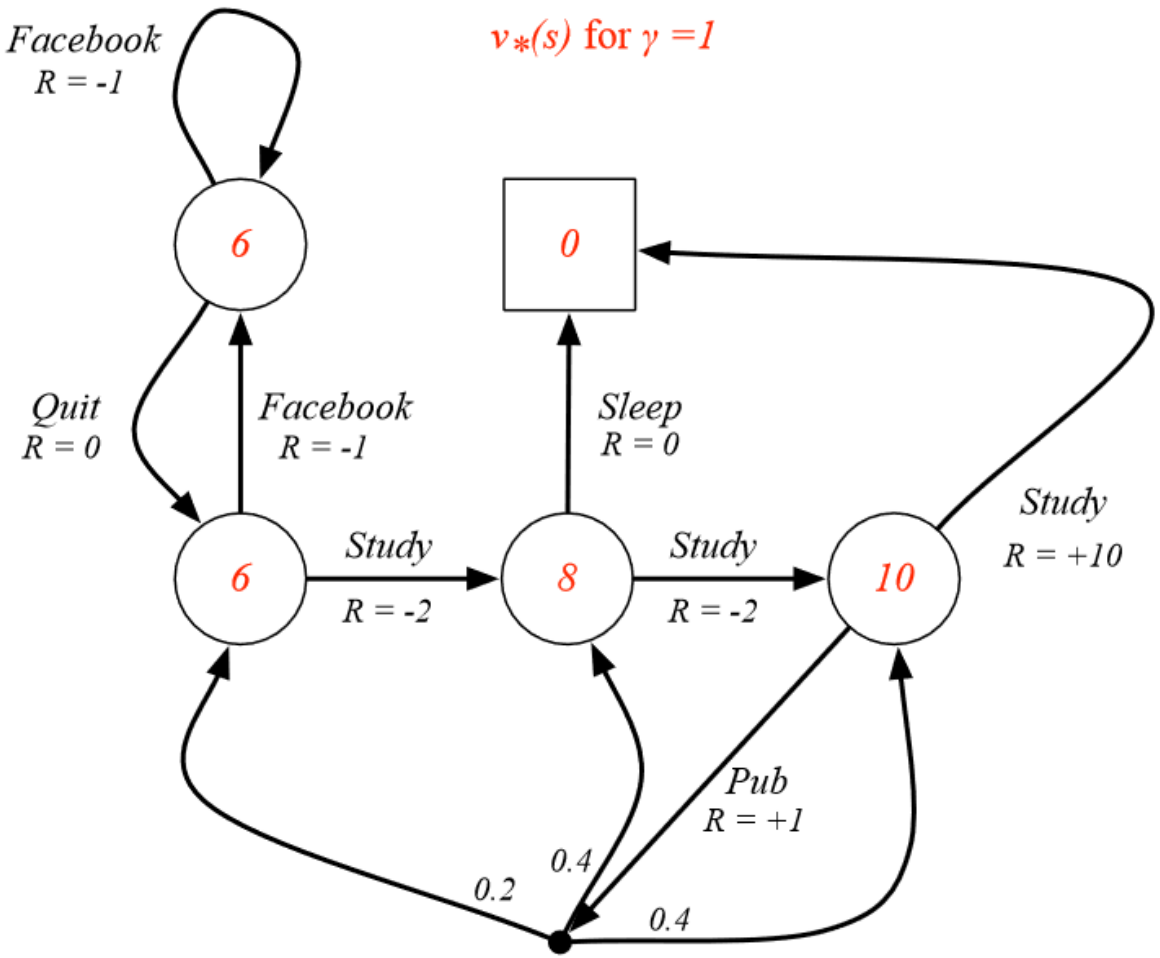
The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies

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

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

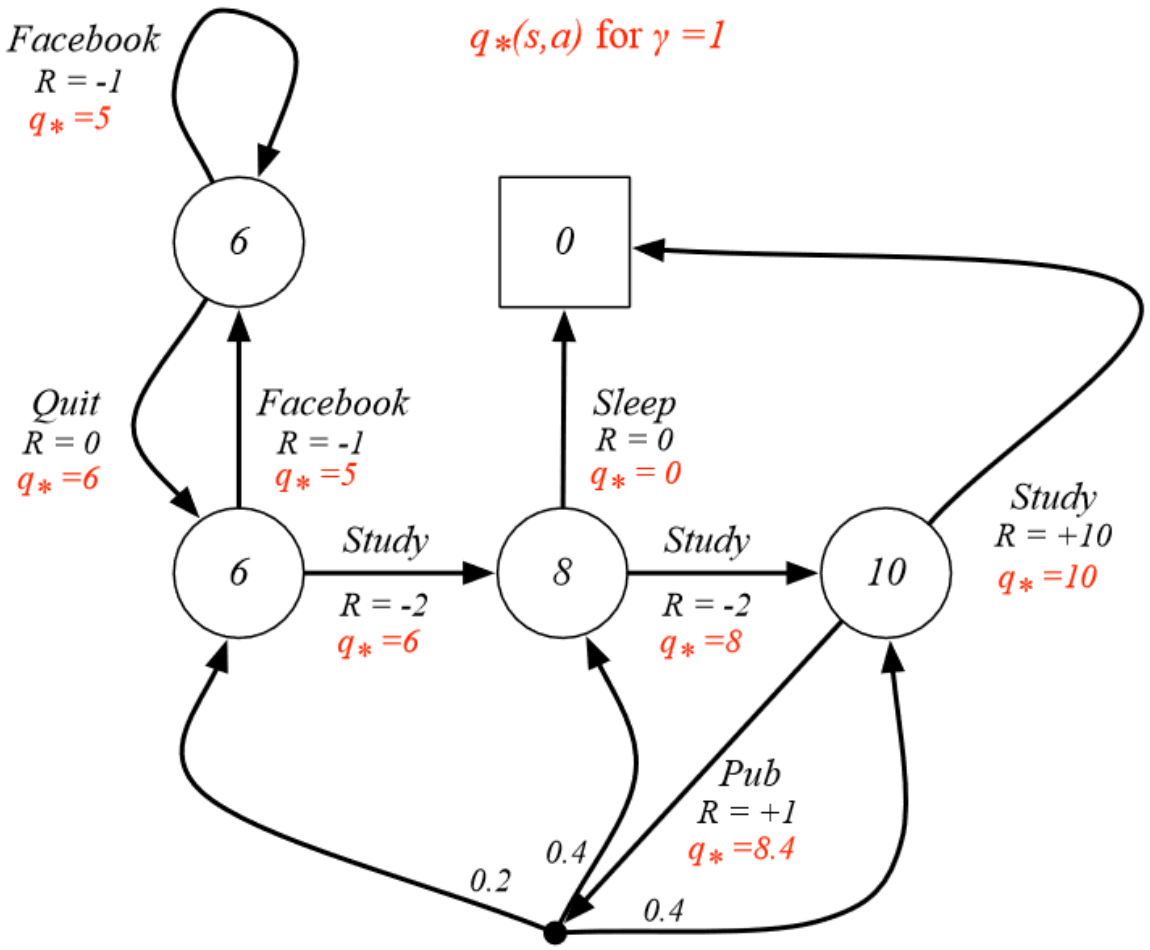
$$q_*(s, a) = \max_{\pi} q_\pi(s, a)$$

Optimal Value Function: Example



Source: D. Silver, DeepMind, Lectures

Optimal Action-Value Function: Example



Source: D. Silver, DeepMind, Lectures

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 π_* 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 Optimal Policy

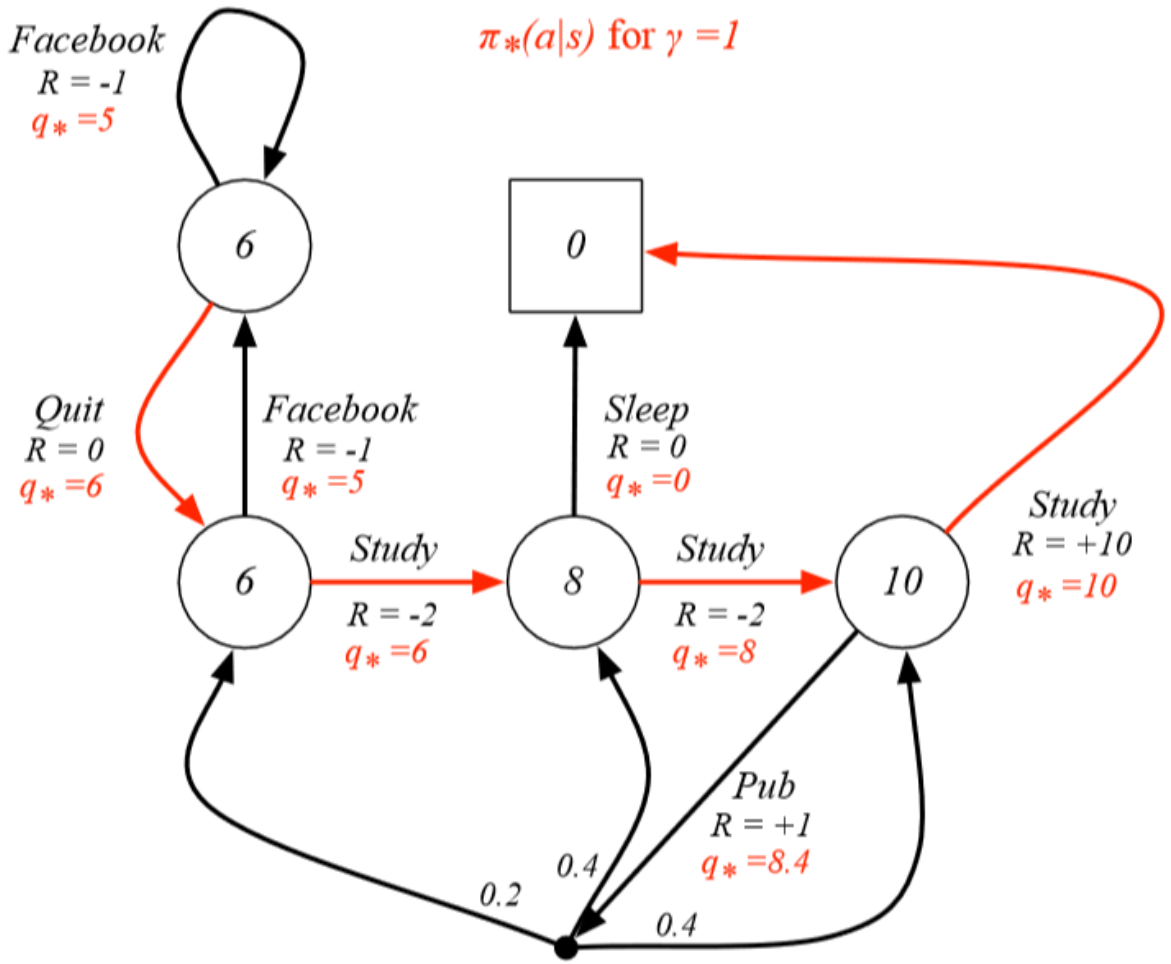
An optimal policy can be found by maximising over $q_*(s, a)$,

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

There is always a deterministic optimal policy for any MDP

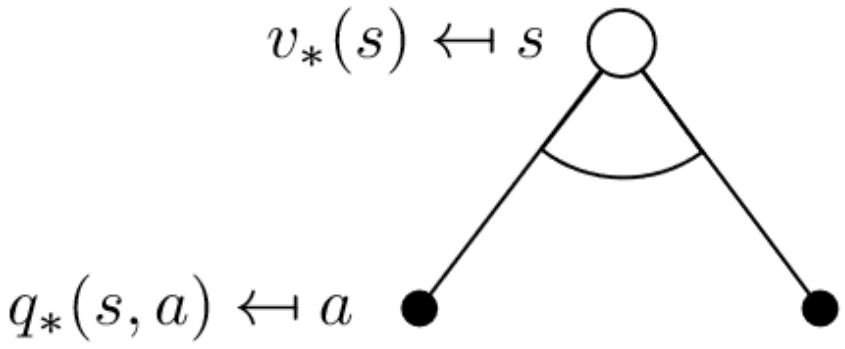
If we know $q_*(s, a)$, we immediately have the optimal policy

Optimal Policy: Example



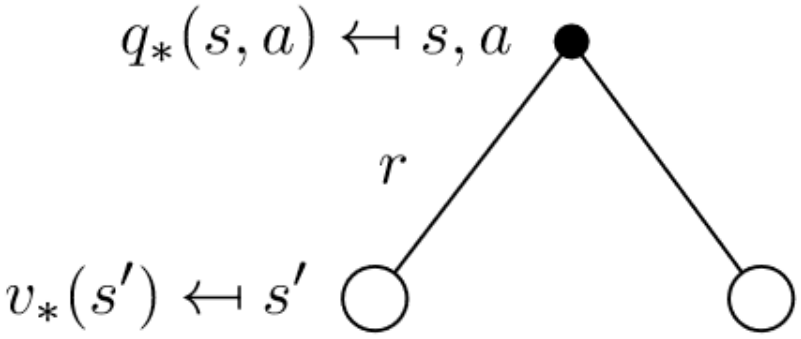
Source: D. Silver, DeepMind, Lectures

Bellman Optimality Equation for V^*



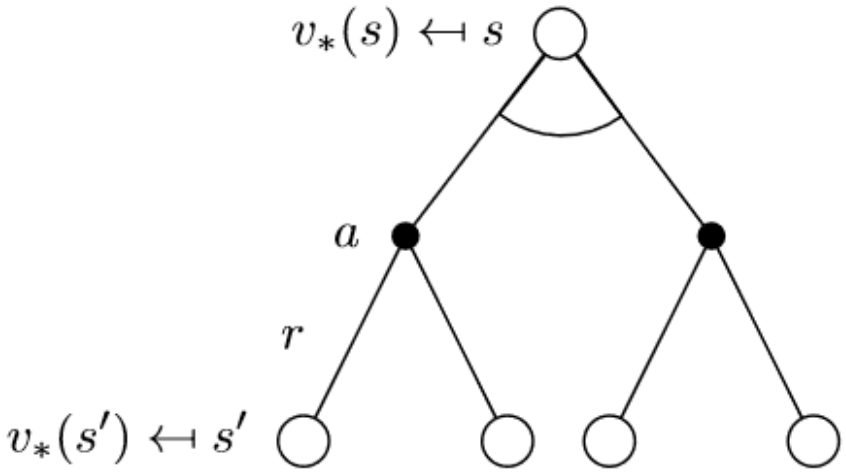
$$v_*(s) = \max_a q_*(s,a)$$

Bellman Optimality Equation for Q^*



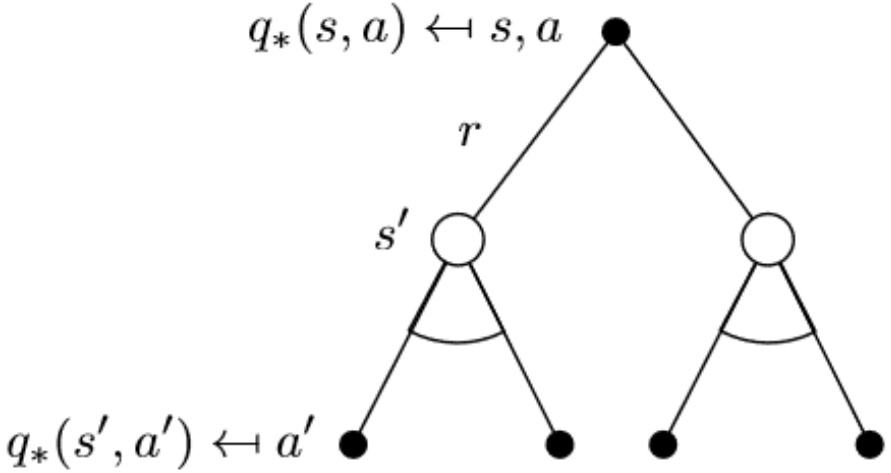
$$q_*(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

Bellman Optimality Equation for V^* (2)



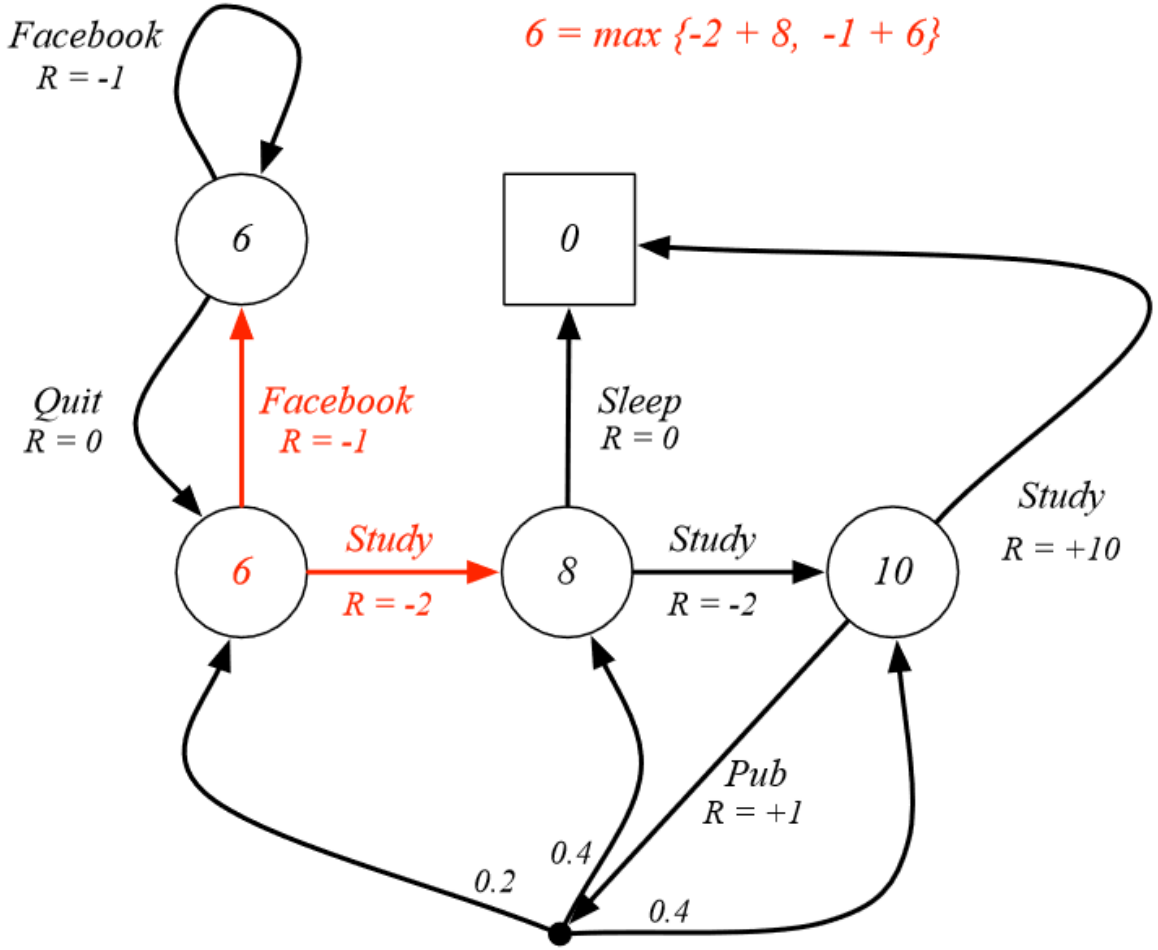
$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

Bellman Optimality Equation for Q^* (2)



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

Bellman Optimality Equation: Example

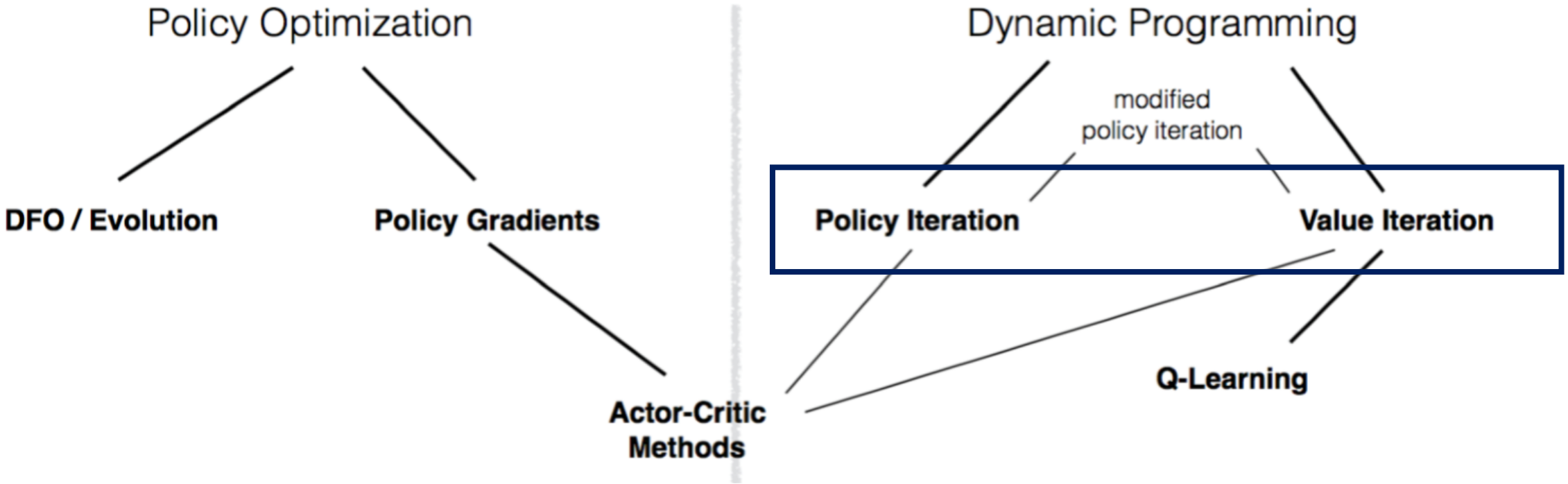


Source: D. Silver, DeepMind, Lectures

Solving Bellman Optimality Equation

- Bellman Optimality Equation is non-linear
- No closed form solution
- Many iterative solution methods exist:
  - Value Iteration (today)
  - Policy Iteration (today)
  - Q-learning
  - SARSA

Types of RL Algorithms



Source: J. Schulman, OpenAI, Lectures

Dynamic Programming

- Methods for solving complex problems which have two properties:
  - Optimal substructure
    - *Principle of optimality*
    - Optimal solution can be decomposed into subproblems
  - Overlapping subproblems:
    - Subroblems recur many times
    - Solutions to subproblems may be stored and reused
- Markov Decision Process satisfies both properties:
  - Bellman equation is recursive
  - Value/Q-value function stores and reuses solutions

Principle of Optimality

- Any optimal policy can be subdivided into two components:
  - Optimal first action $A*$
  - Followed by optimal policy from successor state S'

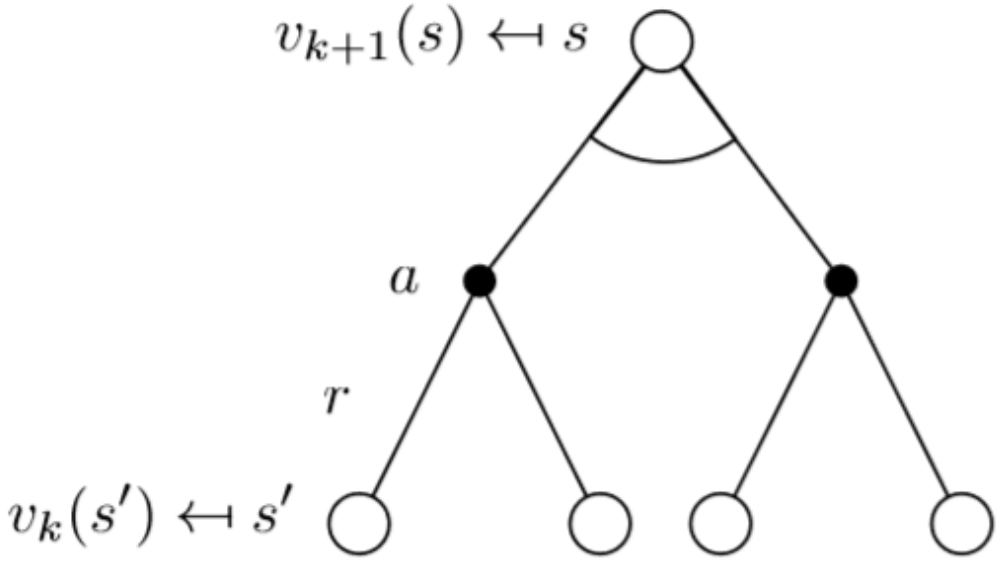
Theorem (Principle of Optimality)

A policy $\pi(a|s)$ achieves the optimal value from state s, $v_\pi(s) = v_*(s)$, if and only if

- *For any state s' reachable from s*
- *π achieves the optimal value from state s', $v_\pi(s') = v_*(s')$*

Source: D. Silver, DeepMind, Lectures

Value Iteration



$v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v^*$

Directly find optimal policy by iteratively applying Bellman optimality equation:

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

Value Iteration

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

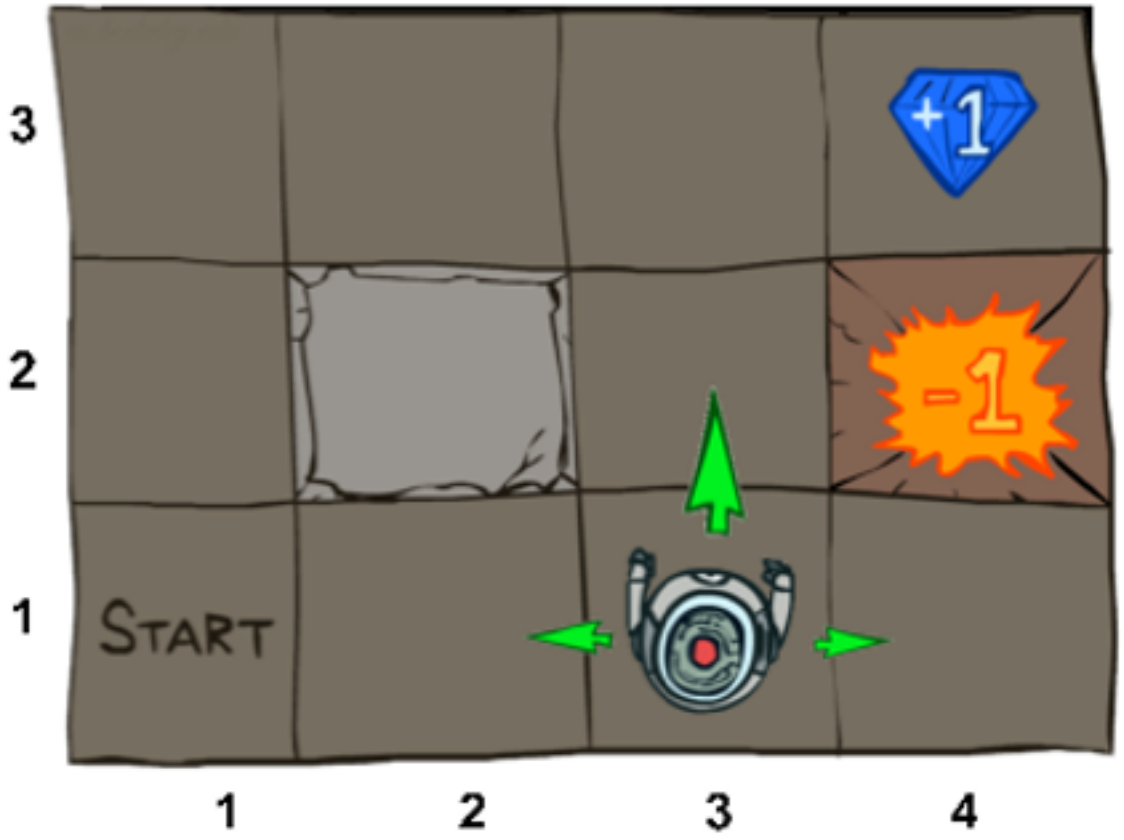
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

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

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

Source: Sutton & Barto, Introduction to Reinforcement Learning, 2nd Edition

Value Iteration: Example



Discount: 0.9

Noise: 0.2

Source: P. Abbeel, UC Berkley, Lectures

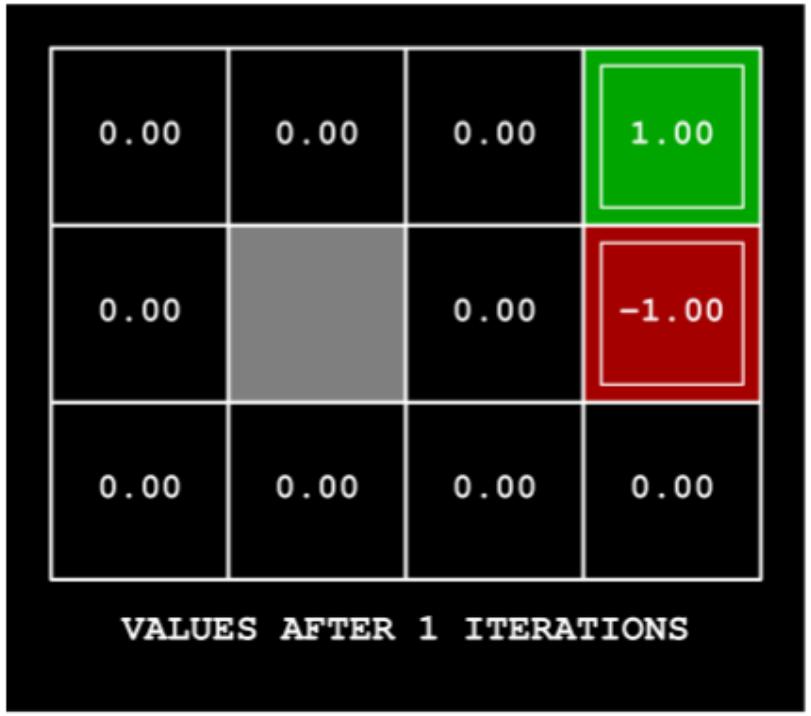
Value Iteration: Example

k = 0

$$V(s) \leftarrow 0$$

Value Iteration: Example

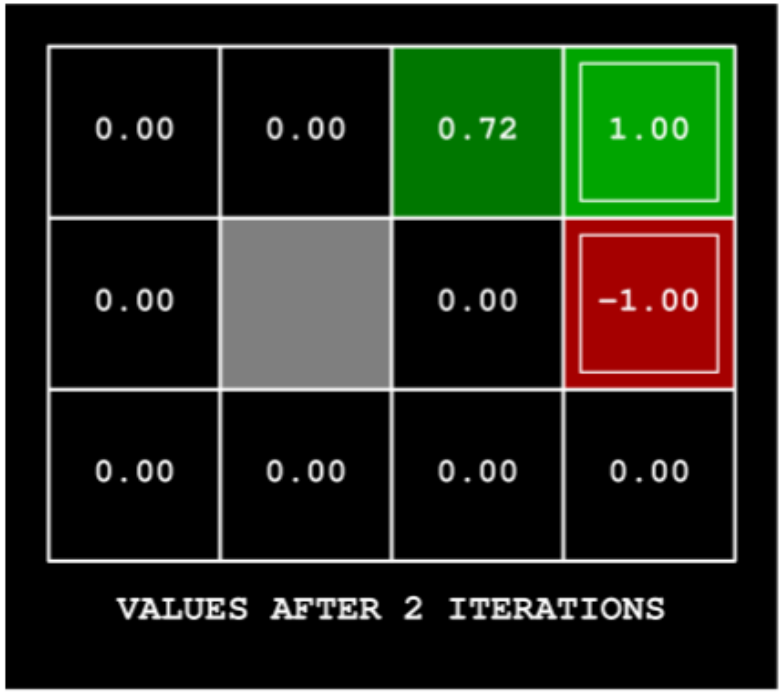
k = 1



$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

Value Iteration: Example

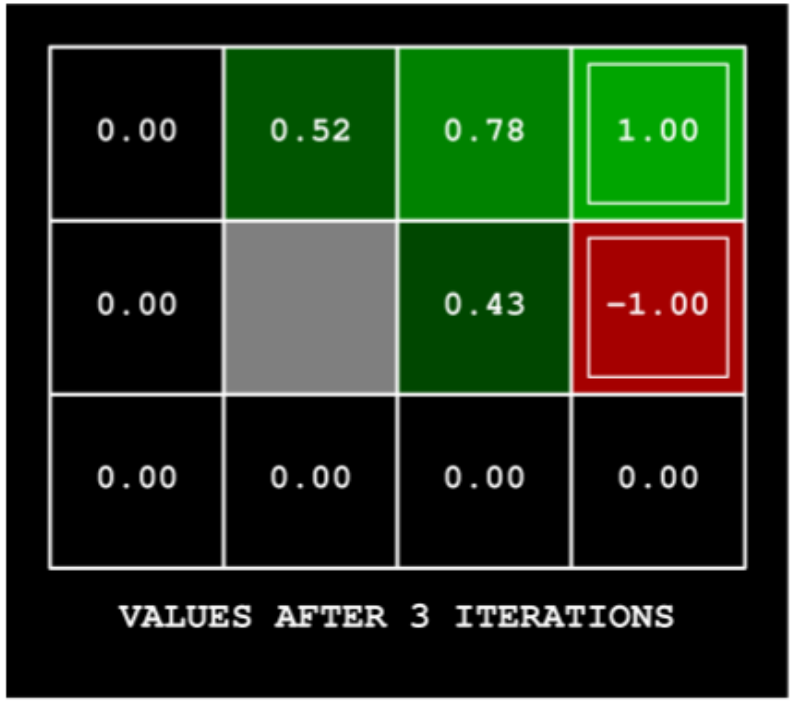
k = 2



$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

Value Iteration: Example

k = 3



$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

Value Iteration: Example

k = 4

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

Value Iteration: Example

k = 5

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

Value Iteration: Example

k = 6

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

Value Iteration: Example

k = 7

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

Value Iteration: Example

k = 8

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

Value Iteration: Example

k = 9

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

Value Iteration: Example

k = 10

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

Value Iteration: Example

k = 11

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

Value Iteration: Example

k = 12

$$V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

Value Iteration: Example

k = 100

$$V(s) = \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

Value Iteration: Example

All we have to do to derive optimal policy:

$$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$$

Policy Evaluation

Idea 1. Evaluate current policy – modify Bellman updates

$$V(s) \leftarrow \max_a \sum_{s',r} p(s',r|s,a)\left[r+\gamma V(s')\right]$$

$$V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\left[r+\gamma V(s')\right]$$

Idea 2. It's just linear system – solve iteratively or with numpy

for all s ϵ *S*

$$V(s) = \sum_{s',r} p(s',r|s,\pi(s))\left[r+\gamma V(s')\right]$$

Policy Iteration

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 $s \in \mathcal{S}$:
     - $v \leftarrow V(s)$
     - $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
     - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

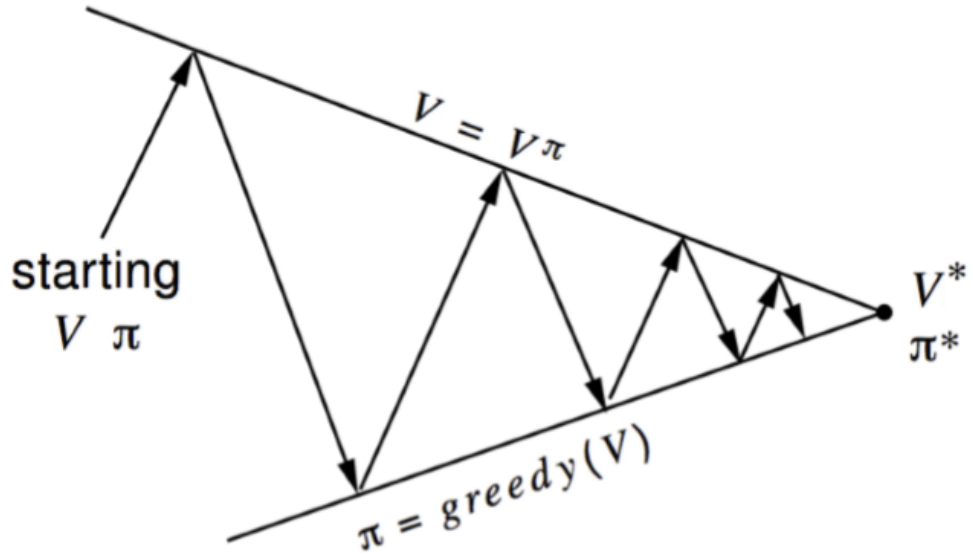
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

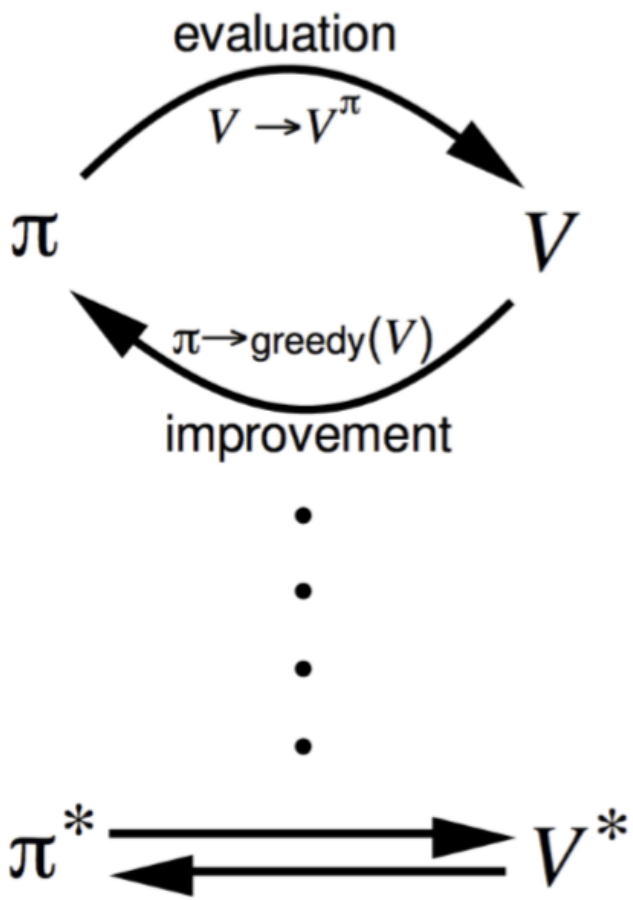
3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   - *old-action* $\leftarrow \pi(s)$
   - $\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
   - If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*

   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Source: Sutton & Barto, Introduction to Reinforcement Learning, 2nd Edition

Policy Iteration





Policy evaluation Estimate v_π

Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement

Q-value Iteration

k = 100

$$Q(s,a) \leftarrow \sum_{s',r} \hat{p}(s',r\,|\,s,a)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

Problems?

Yes!

Problems?

Yes!

Update equations require access to dynamics model of the environment

Solution: Next Lecture

Let’s estimate via sampling transitions from interactions with environment

Still problems?

Yes!

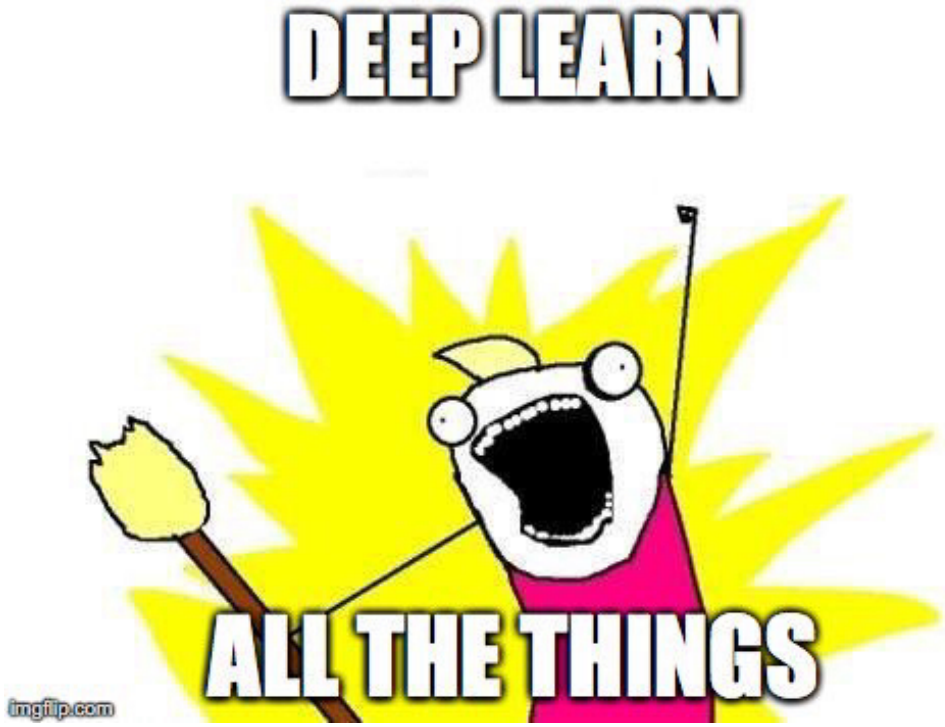
Still problems?

Yes!

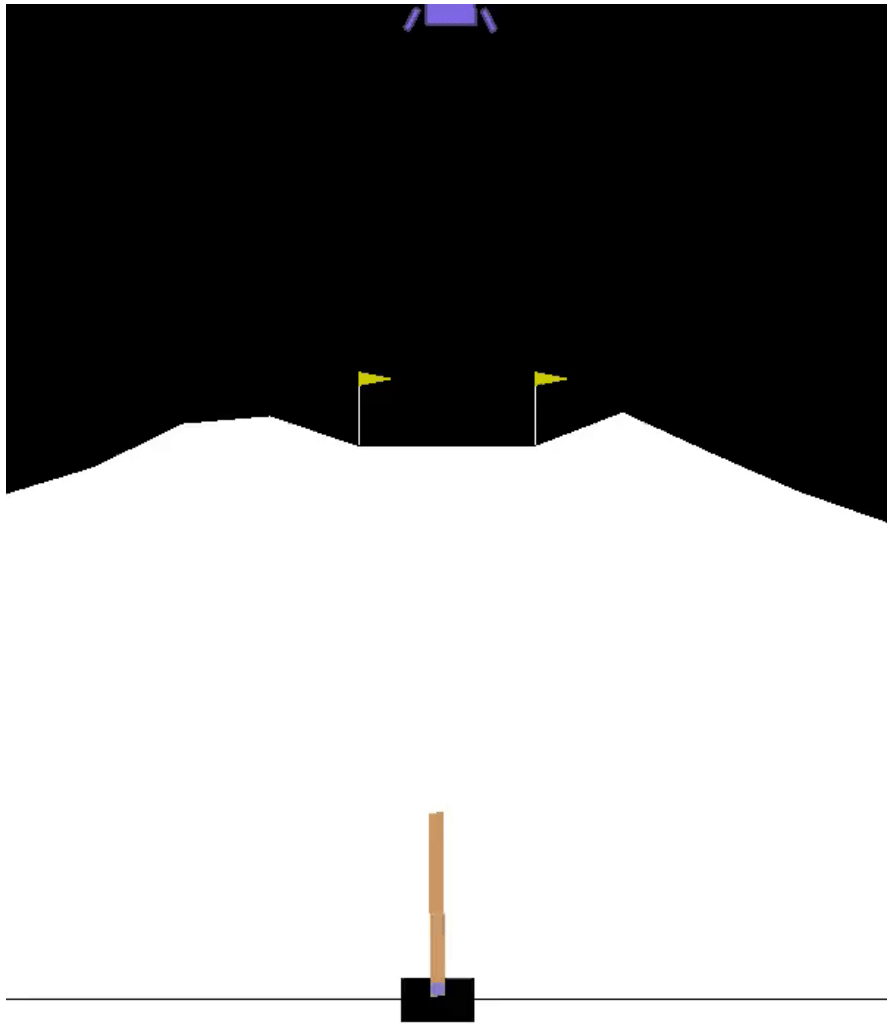
Iteration over/storage for all states and actions requires small, discrete state-action space

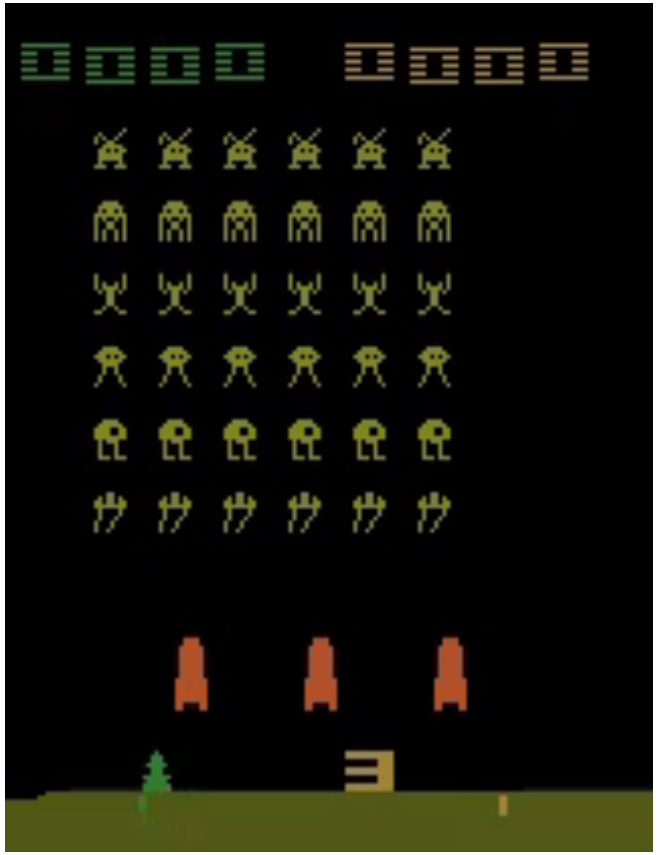
Solution: Next Lecture

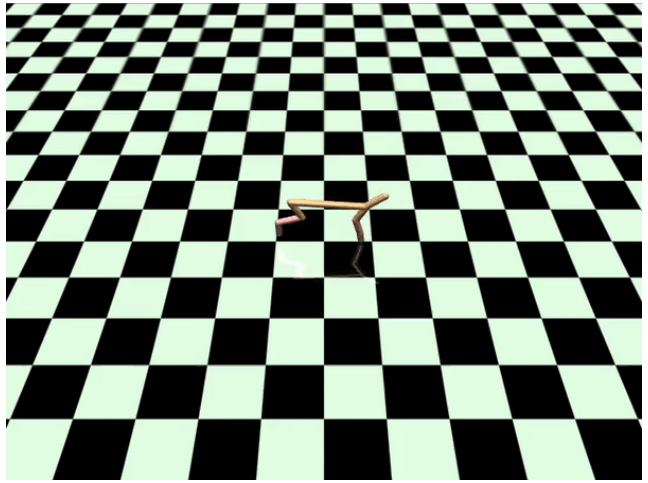
- From tables to function approximations
- DQN (Deep Q-network)
- DQN tips and tricks



OpenAI Gym: Environments







OpenAI Gym: Simple API

```
import gym
env = gym.make("Taxi-v1")
observation = env.reset()
for _ in range(1000):
  env.render()
  action = env.action_space.sample() # your agent here (this takes random actions)
  observation, reward, done, info = env.step(action)
```

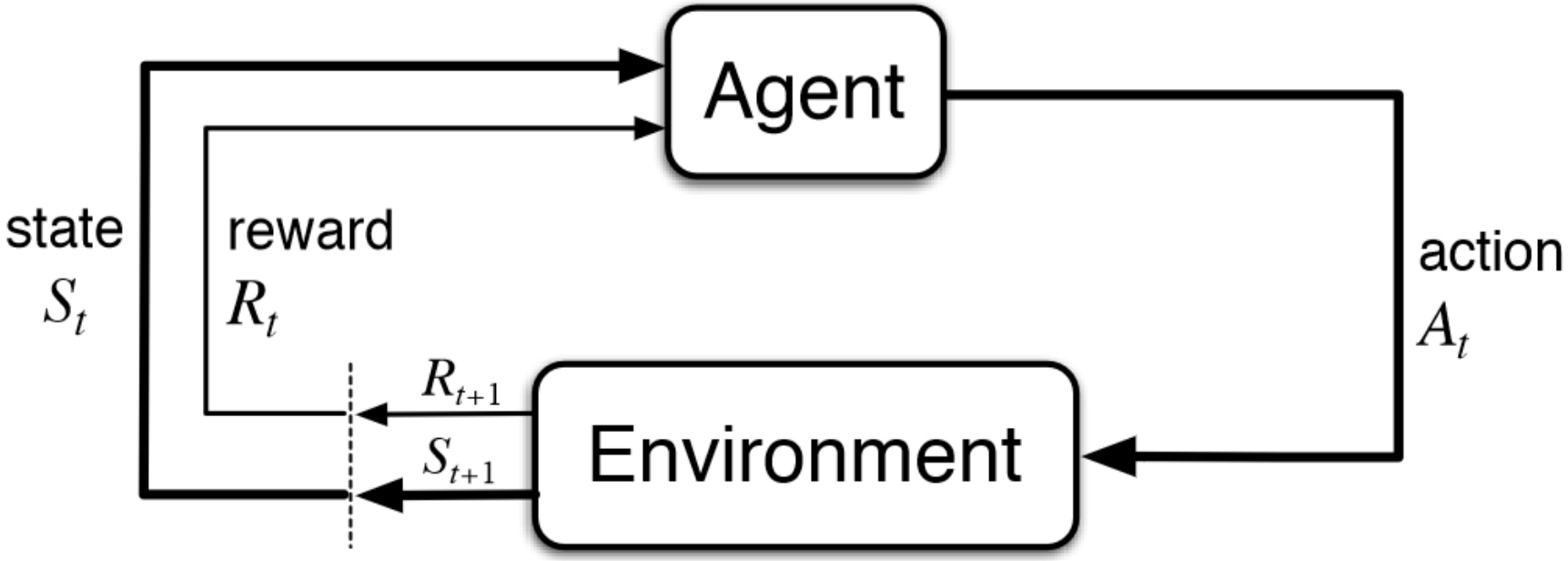
Useful links

- R. Sutton, A. Barto. Reinforcement Learning: An Introduction. Second Edition (Chapter 3 & 4)

  http://incompleteideas.net/book/the-book-2nd.html
- David Silver. Reinforcement Learning (Lecture 2 & 3)

  http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

Recap: Reinforcement Learning Problem



Source: Sutton & Barto, Introduction to Reinforcement Learning, 2nd Edition

Recap: Return

Definition

The *return* G_t is the total discounted reward from time-step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Recap: Value Function

Definition

The *state-value function* $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

Definition

The *action-value function* $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right]$$

Recap: Policy

Definition

A *policy* π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}\left[A_t = a \mid S_t = s\right]$$

- Policy completely describes agent's behavior
- MDP policies depend on the current state
- Policies are stationary (time-independent)

Recap: Optimal Value Function

Definition

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

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

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

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

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 π_* 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 Optimal Policy

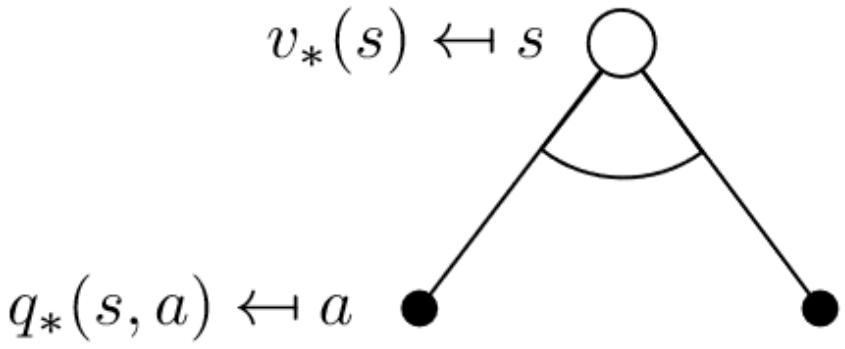
An optimal policy can be found by maximising over $q_*(s, a)$,

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

There is always a deterministic optimal policy for any MDP

If we know $q_*(s, a)$, we immediately have the optimal policy

Bellman Optimality Equation for V^*



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

Problems?

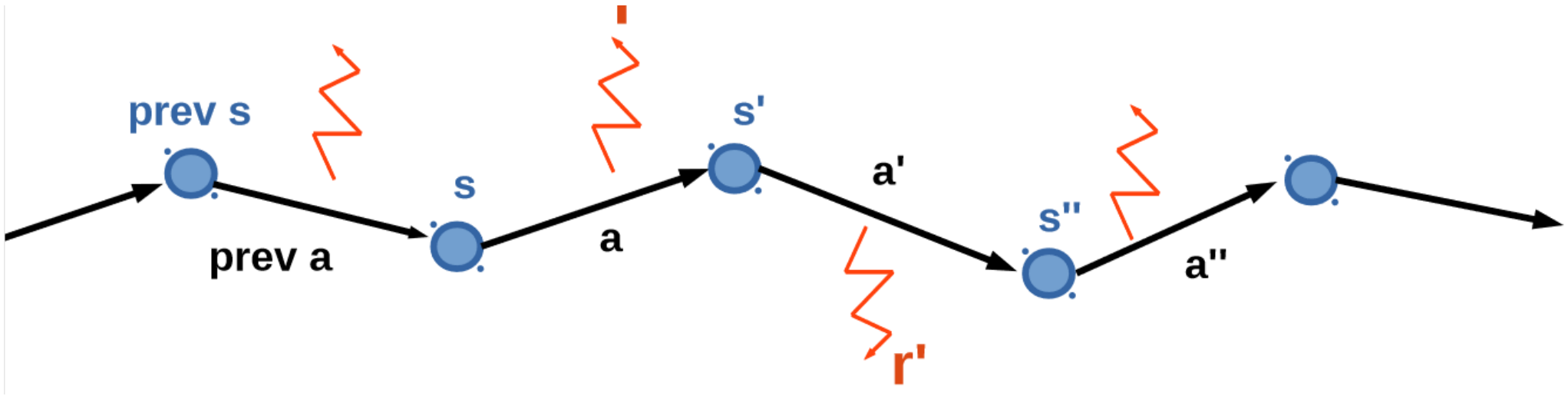
Yes!

Update equations require access to dynamics model of the environment

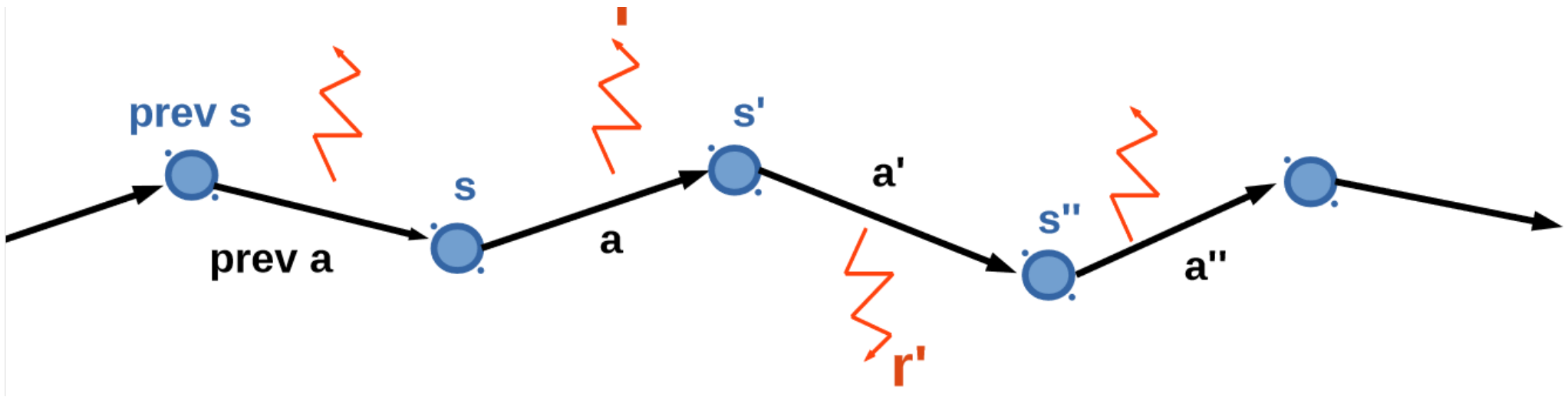
What can we do about it?

Sample trajectories from interactions with the environment

Learning from trajectories (I)



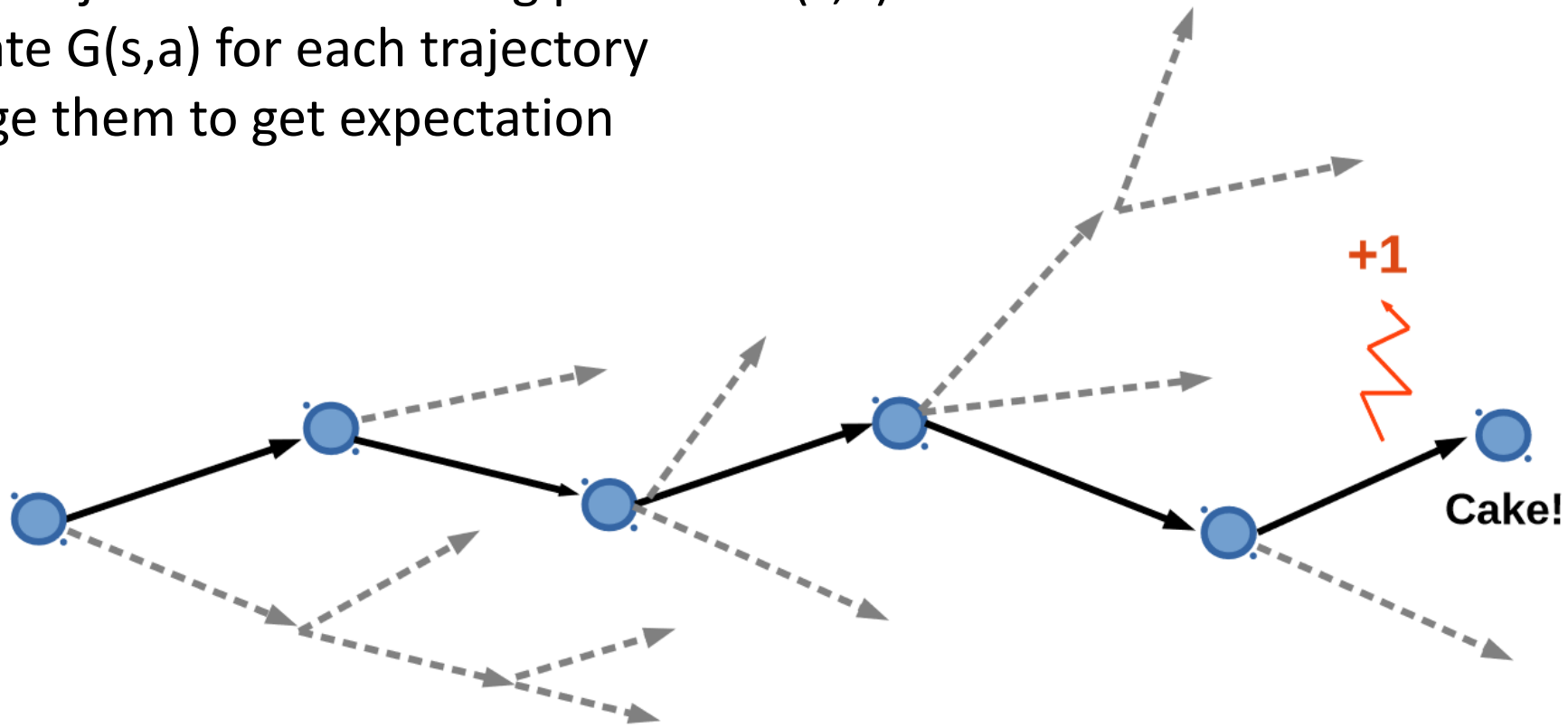
Learning from trajectories (II)



- Trajectory is a sequence of states – actions – rewards (r)
- We can only sample trajectories

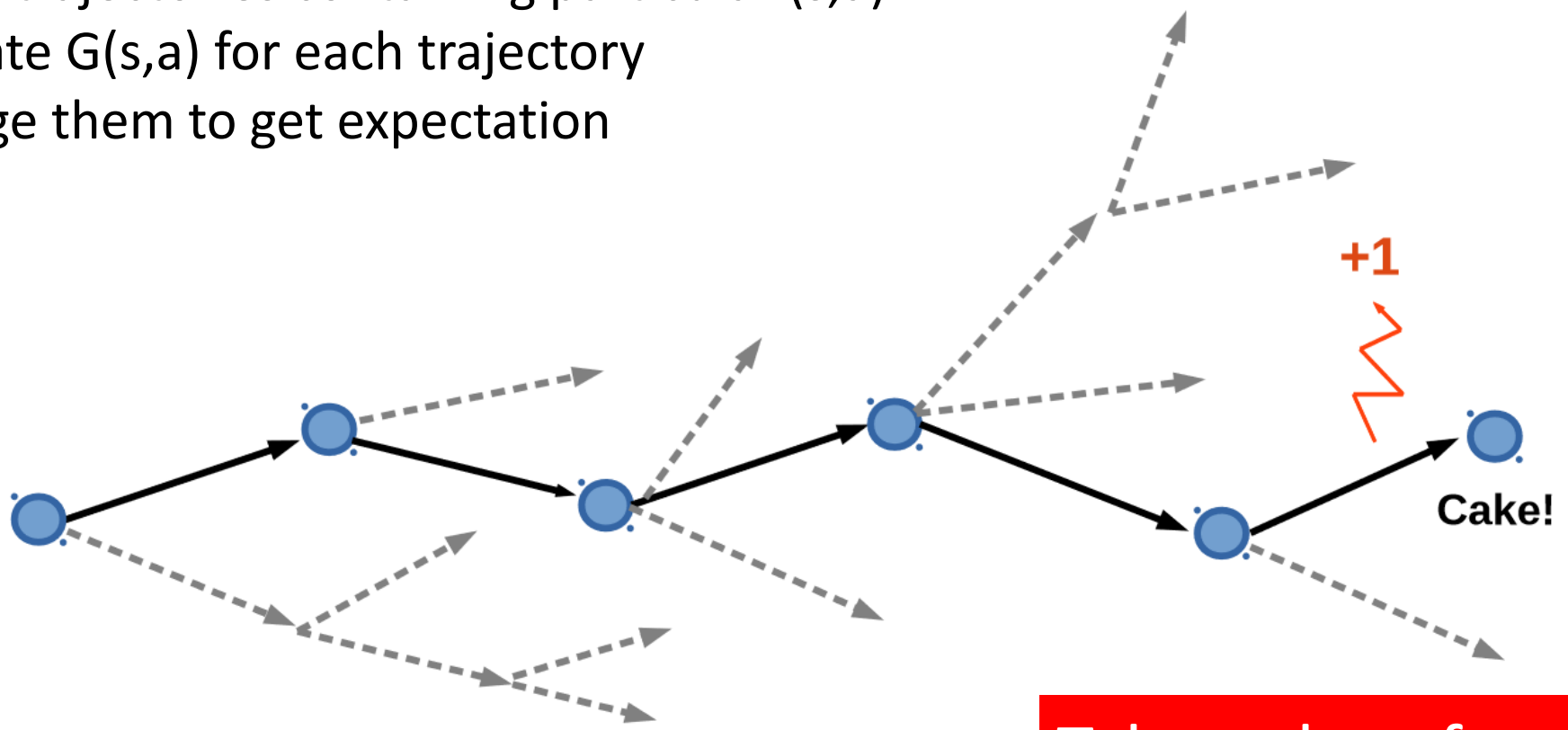
Метод Монте-Карло (I)

- Get all trajectories containing particular (s,a)
- Estimate G(s,a) for each trajectory
- Average them to get expectation



Метод Монте-Карло (II)

- Get all trajectories containing particular (s,a)
- Estimate G(s,a) for each trajectory
- Average them to get expectation



Takes a lot of samples

Monte-Carlo vs Q-learning vs SARSA

Great slides from Practical RL Course:

https://github.com/yandexdataschool/Practical_RL/tree/master/week3_model_free

From Q-value iteration to Q-learning

$Q^*(s, a)$ = expected utility starting in s, taking action a, and (thereafter) acting optimally

Bellman Equation:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q^*(s',a'))$$

Q-Value Iteration:

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$$

From Q-value iteration to Q-learning

- Q-value iteration: $Q_{k+1}(s,a) \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$
- Rewrite as expectation: $Q_{k+1} \leftarrow \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a') \right]$
- (Tabular) Q-Learning: replace expectation by samples
  - For an state-action pair (s,a), receive: $s' \sim P(s'|s,a)$
  - Consider your old estimate: $Q_k(s,a)$
  - Consider your new sample estimate:

    $$\text{target}(s') = R(s,a,s') + \gamma \max_{a'} Q_k(s',a')$$
  - Incorporate the new estimate into a running average:

    $$Q_{k+1}(s,a) \leftarrow (1-\alpha) Q_k(s,a) + \alpha \left[\text{target}(s') \right]$$

Q-learning

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

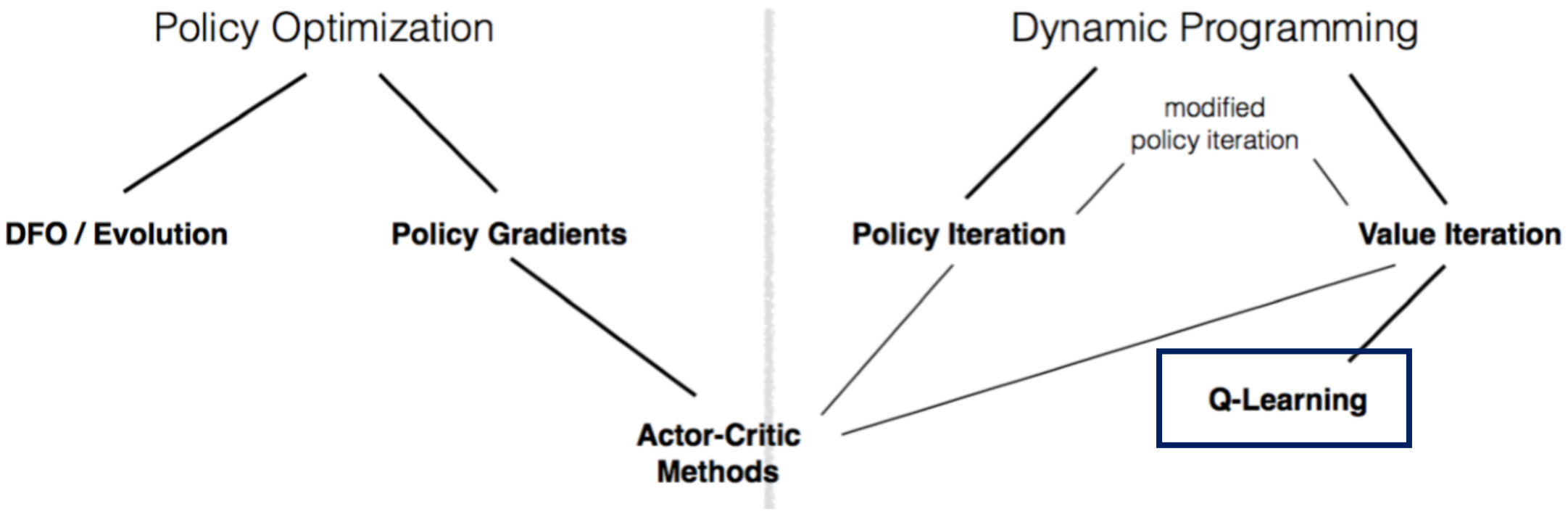
SARSA (Homework)

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

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

Recap: Types of RL Algorithms



Source: J. Schulman, OpenAI, Lectures

Still problems?

Yes!

Still problems?

Yes!

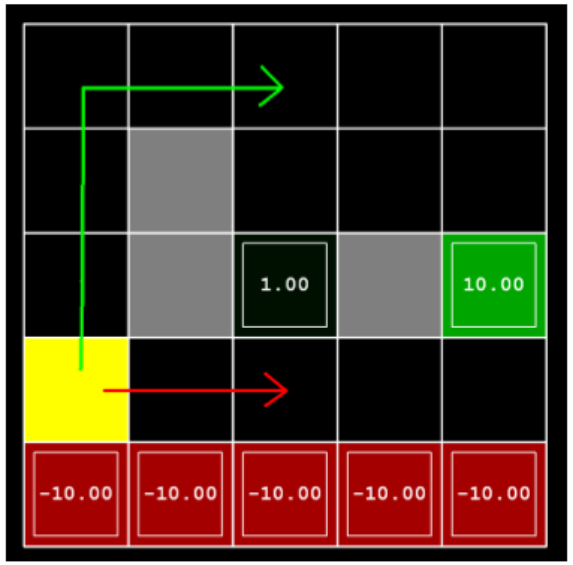
Iteration over/storage for all states and actions requires small, discrete state-action space

What can we do about it?

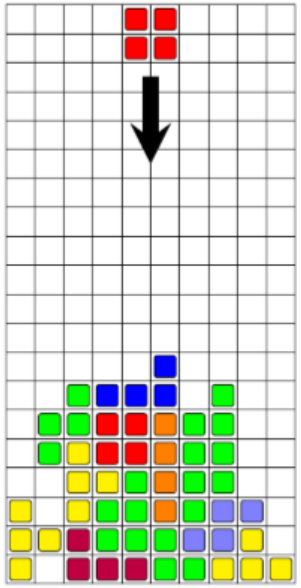
- Discretize action / state space
- Use function approximation

State Spaces (I)

- Discrete environments



Gridworld
10^1

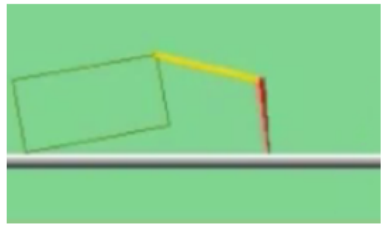


Tetris
10^60

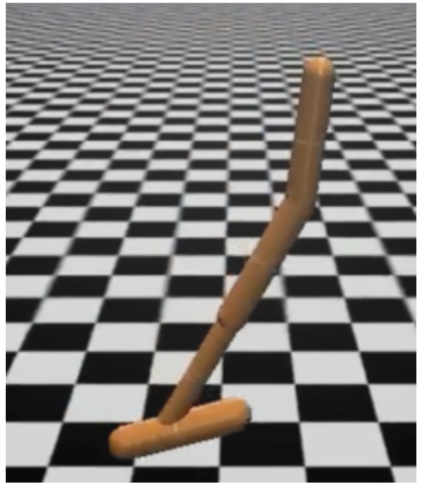
Atari
10^308 (ram) 10^16992 (pixels)

State Spaces (II)

- Continuous environments (by crude discretization)



Crawler
10^2

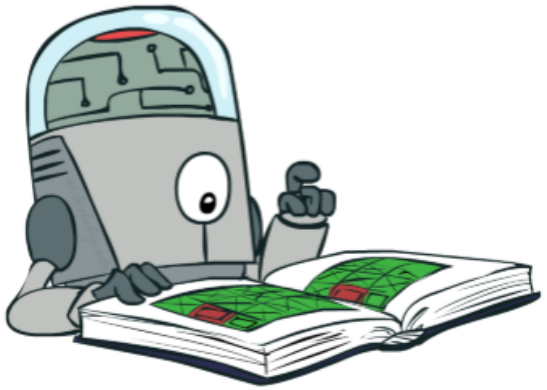


Hopper
10^10

Humanoid
10^100

Generalization

- Basic Q-Learning keeps a table of all q-values



- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory

- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - This is a fundamental idea in machine learning, and we'll see it over and over again

Approximate Q-learning

- Instead of a table, we have a parametrized Q function: $Q_\theta(s,a)$
  - Can be a linear function in features:

$$Q_\theta(s,a) = \theta_0 f_0(s,a) + \theta_1 f_1(s,a) + \cdots + \theta_n f_n(s,a)$$

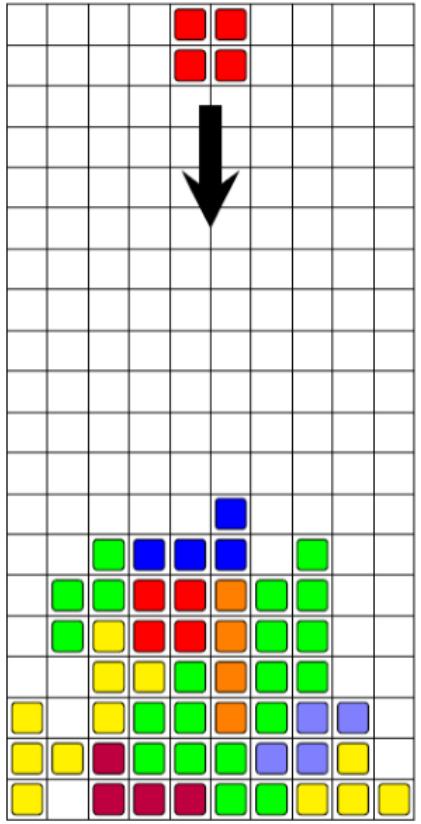
  - Or a complicated neural net
- Learning rule:
  - Remember: $\text{target}(s') = R(s,a,s') + \gamma \max_{a'} Q_{\theta_k}(s',a')$
  - Update:

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \left[\frac{1}{2} (Q_\theta(s,a) - \text{target}(s'))^2 \right] \Bigg|_{\theta=\theta_k}$$

Approximate Q-learning: Example

- state: naïve board configuration + shape of the falling piece ~10^{60} states!
- action: rotation and translation applied to the falling piece
- 22 features aka basis functions ϕ_i
  - Ten basis functions, 0, . . . , *9, mapping the state to the height h[k] of each* column.
  - Nine basis functions, 10, . . . , *18, each mapping the state to the absolute difference* between heights of successive columns: $|h[k+1] - h[k]|$, $k = 1, \ldots, 9$.
  - One basis function, 19, that maps state to the maximum column height: $\max_k h[k]$
  - One basis function, 20, that maps state to the number of 'holes' in the board.
  - One basis function, 21, that is equal to 1 in every state.

$$\hat{V}_\theta(s) = \sum_{i=0}^{21} \theta_i \phi_i(s) = \theta^\top \phi(s)$$



[Bertsekas & Ioffe, 1996 (TD); Bertsekas & Tsitsiklis 1996 (TD); Kakade 2002 (policy gradient); Farias & Van Roy, 2006 (approximate LP)]

Useful links

- R. Sutton, A. Barto. Reinforcement Learning: An Introduction. Second Edition (Chapter 5 & 6)

  http://incompleteideas.net/book/the-book-2nd.html
- David Silver. Reinforcement Learning (Lecture 4 & 5)

  http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
- Practical RL Week 3

  https://github.com/yandexdataschool/Practical_RL/tree/master/week3_model_free