10703 Deep Reinforcement Learning and Control

Russ Salakhutdinov

Slides borrowed from
Katerina Fragkiadaki

Markov Decision Processes

Logistics

- Prerequisites: Strong knowledge of Linear Algebra, Optimization, Machine Learning, Deep learning, Algorithms
- Three assignments and a final project, 60%/40%
- TAs, collaboration policy, late policy, office hours are or will be announced on the website this week
- People can audit the course, unless there are no seats left in class

Project

- The idea of the final project is to give you some experience trying to do a piece of original research in machine learning and coherently writing up your result.
- What is expected: A simple but original idea that you describe clearly, relate to existing methods, implement and test on some real-world problem.
- To do this you will need to write some basic code, run it on some data, make some figures, read a few background papers, collect some references, and write an 8-page report describing your model, algorithm, and results.
- You are welcome to work in groups of up to 3 people.

Textbooks

- The Sutton & Barto, Reinforcement Learning: An Introduction
- Ian Goodfellow, Yoshua Bengio, Aaron Courville (2016) Deep Learning Book (available online)
- Plus papers

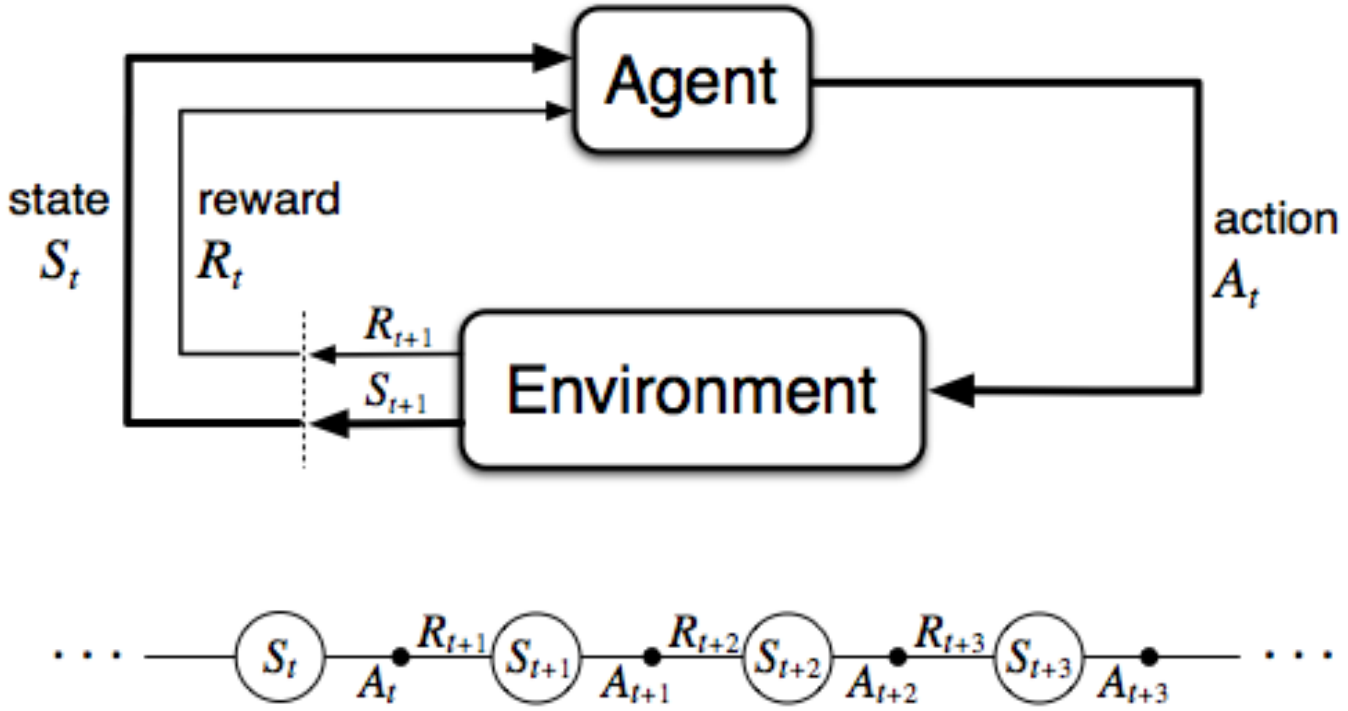
Online courses

- Rich Sutton's class: Reinforcement Learning for Artificial Intelligence, Fall 2016
- John Schulman's and Pieter Abeel's class: Deep Reinforcement Learning, Fall 2015
- Sergey Levine's, Chelsea Finn's and John Schulman's class: Deep Reinforcement Learning, Spring 2017
- Abdeslam Boularias's class: Robot Learning Seminar
- Pieter Abeel's class: Advanced Robotics, Fall 2015
- Emo Todorov's class: Intelligent control through learning and optimization, Spring 2015
- David Silver's class: Reinforcement learning

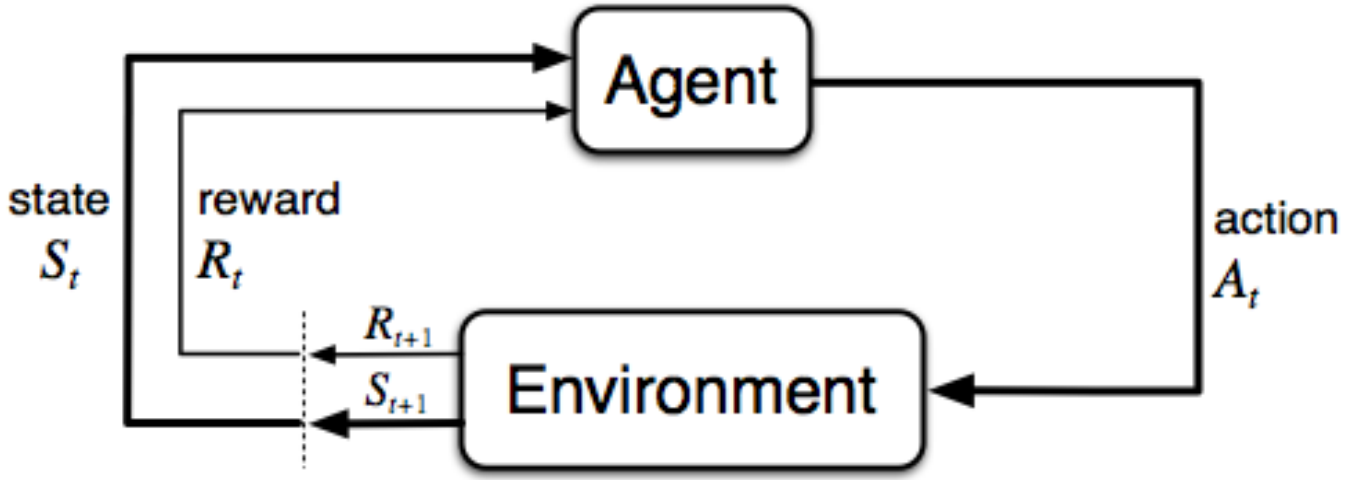
Outline

- Agents, Actions, Rewards
- Markov Decision Processes
- Value functions
- Optimal value functions

The Agent-Environment Interface



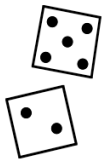
The Agent-Environment Interface

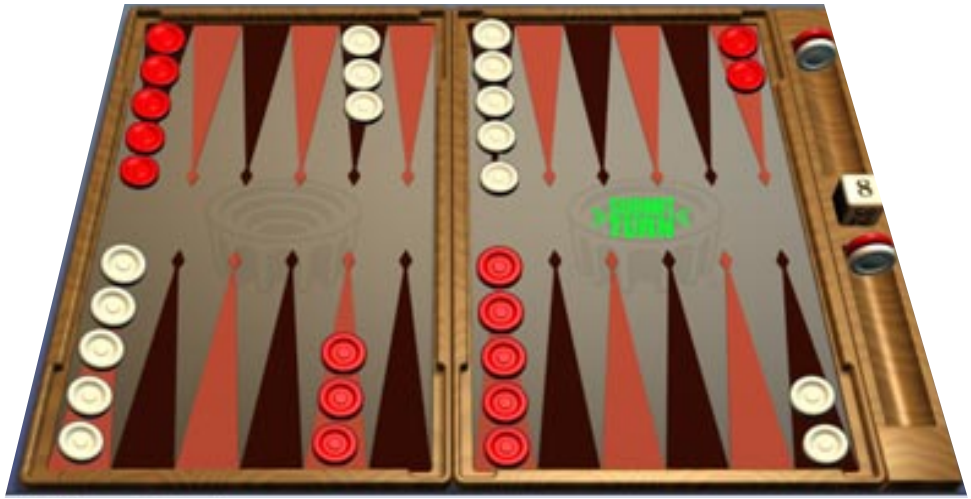


- Rewards specify what the agent needs to achieve, not how to achieve it.
- The simplest and cheapest form of supervision

Backgammon

- States: Configurations of the playing board (≈1020)
- Actions: Moves
- Rewards:
  - win: +1
  - lose: –1
  - else: 0





Visual Attention

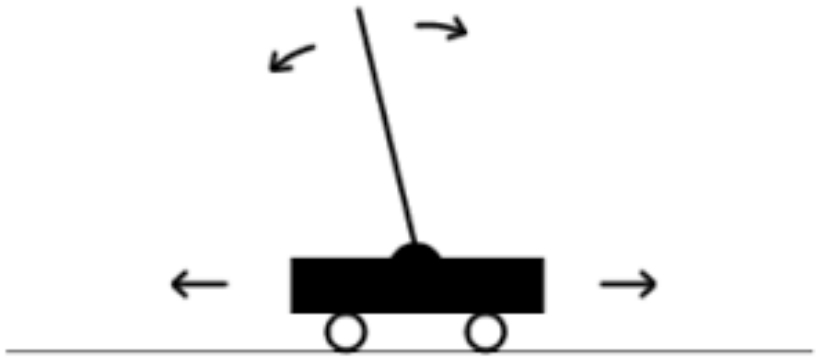
- States: Road traffic, weather, time of day
- Actions: Visual glimpses from mirrors/cameras/front
- Rewards:
  - +1 safe driving, not over-tired
  - -1: honking from surrounding drivers

Figure-Skating

conservative exploration strategy

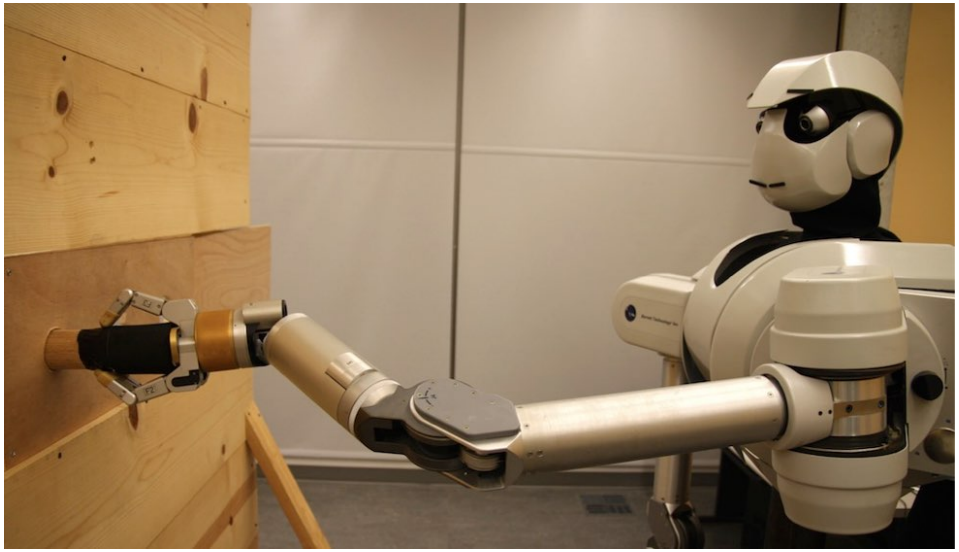
Cart Pole

- States: Pole angle and angular velocity
- Actions: Move left, right
- Rewards:
  - 0 while balancing
  - -1 for imbalance



Peg in Hole Insertion Task

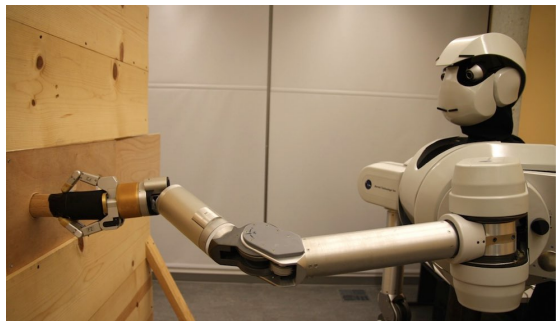
- States: Joint configurations
- Actions: Torques on joints
- Rewards: Penalize jerky motions, inversely proportional to distance from target pose

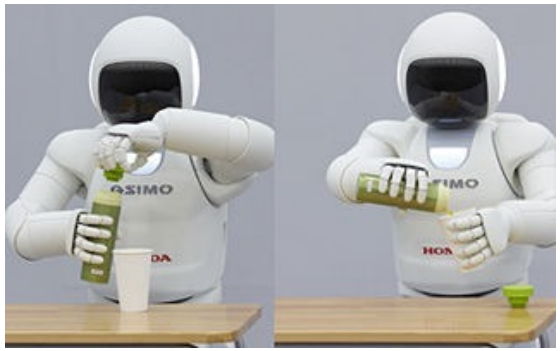


Detecting Success

- The agent should be able to measure its success explicitly.

- We often times cannot automatically detect whether the task has been achieved.





Limitations

- Can we think of goal directed behavior learning problems that cannot be modeled or are not meaningful using the MDP framework and a trial-and-error Reinforcement learning framework?
- The agent should have the chance to try (and fail) enough times
- This is impossible if episode takes too long, e.g., reward="obtain a great Ph.D."
- This is impossible when safety is a concern: we can't learn to drive via reinforcement learning in the real world, failure cannot be tolerated

Markov Decision Process

A **Markov Decision Process** is a tuple $(\mathcal{S}, \mathcal{A}, T, r, \gamma)$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- T is a state transition probability function

$$T(s'|s,a) = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

- r is a reward function

$$r(s,a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$$

- γ is a discount factor $\gamma \in [0, 1]$

Actions

- For now we assume discrete actions.
- Actions can have many different temporal granularities.

States

- A state captures whatever information is available to the agent at time step t about its environment.
- The state can include immediate “sensations", highly processed sensations, and structures built up over time from sequences of sensations, memories, etc.
- A state should summarize past sensations so as to retain all “essential” information, i.e., it should have the **Markov Property**:

$$\mathbb{P}[R_{t+1} = r, S_{t+1} = s' | S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t] = \mathbb{P}[R_{t+1} = r, S_{t+1} = s' | S_t, A_t]$$

for all $s' \in \mathcal{S}, r \in \mathcal{R}$

- We should be able to throw away the history once state is known.

States

- A state captures whatever information is available to the agent at time step t about its environment.
- The state can include immediate “sensations,” highly processed sensations, and structures built up over time from sequences of sensations, memories, etc.
- An agent cannot be blamed for missing information that is unknown, but for forgetting relevant information.

States

- A state captures whatever information is available to the agent at time step t about its environment.
- The state can include immediate “sensations,” highly processed sensations, and structures built up over time from sequences of sensations, memories, etc.
- What would you expect to be the state information of a vacuum-cleaner robot?

Dynamics

- How the state changes given the actions of the agent
- Model based: dynamics are known or are estimated
- Model free: we do not know the dynamics of the MDP

Since in practice the dynamics are unknown, the state representation should be such that it is easily predictable from neighboring states

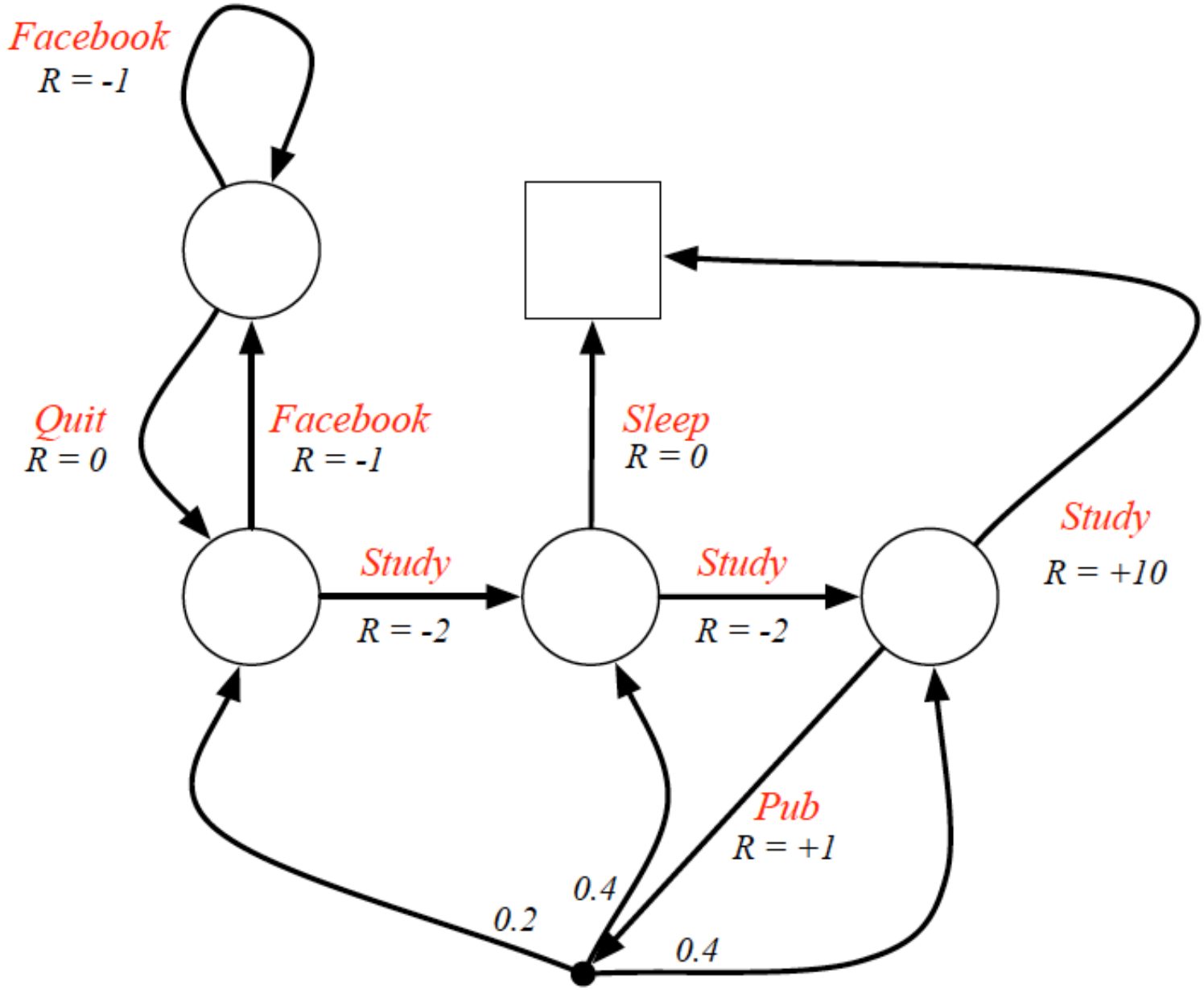
Rewards

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

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

- The objective in RL is to maximize long-term future reward
- That is, to choose A_t so as to maximize $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$
- Episodic tasks - finite horizon vs. continuous tasks - infinite horizon
- In episodic tasks we can consider undiscounted future rewards

The Student MDP



Agent Learns a Policy

Definition: A policy π is a distribution over actions given states,

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

- A policy fully defines the behavior of an agent
- MDP policies depend on the current state (not the history)
- i.e. policies are stationary (time-independent)

$$A_t \sim \pi(\cdot|S_T), \forall t > 0$$

Solving Markov Decision Processes

- Find the optimal policy
- Prediction: For a given policy, estimate value functions of states and states/action pairs
- Control: Estimate the value function of states and state/action pairs for the optimal policy.

Value Functions

	state values	action values
prediction	v_π	q_π
control	v_*	q_*

- Value functions measure the goodness of a particular state or state/action pair: how good is for the agent to be in a particular state or execute a particular action at a particular state. Of course that depends on the policy.
- Optimal value functions measure the best possible goodness of states or state/action pairs under any policy.

Value Functions are Cumulative Expected Rewards

Definition: The *state-value function* $\mathrm{v}_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π

$$\mathrm{v}_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

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

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

Optimal Value Functions are Best Achievable Cumulative Expected Rewards

- **Definition**: The *optimal state-value function* $\mathrm{v}_*(s)$ is the maximum value function over all policies

$$\mathrm{v}_*(s) = \max_\pi \mathrm{v}_\pi(s)$$

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

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

Bellman Expectation Equation

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

The value function can be decomposed into two parts:

- Immediate reward R_{t+1}
- Discounted value of successor state $\gamma \mathbf{v}(S_{t+1})$

Bellman Expectation Equation

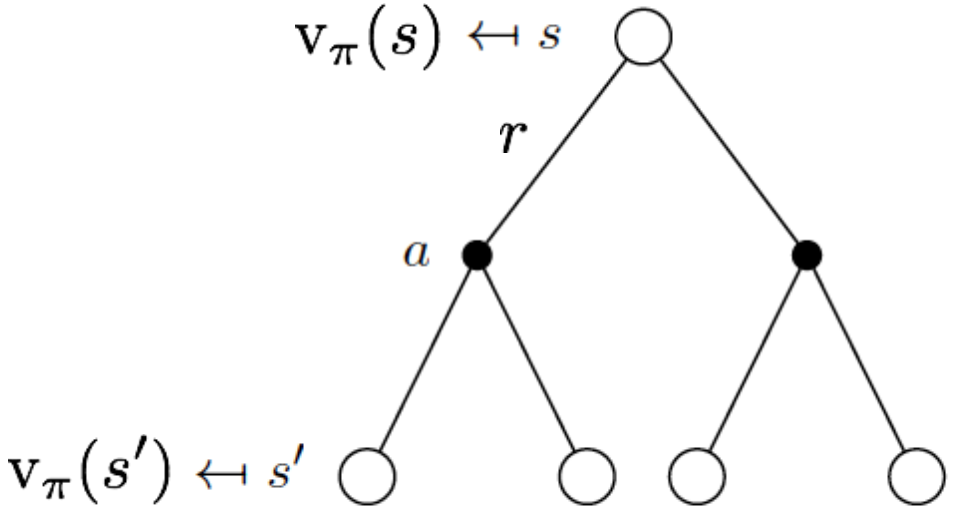
The state-value function can be decomposed into immediate reward plus discounted value of successor state,

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

The action-value function can similarly be decomposed,

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

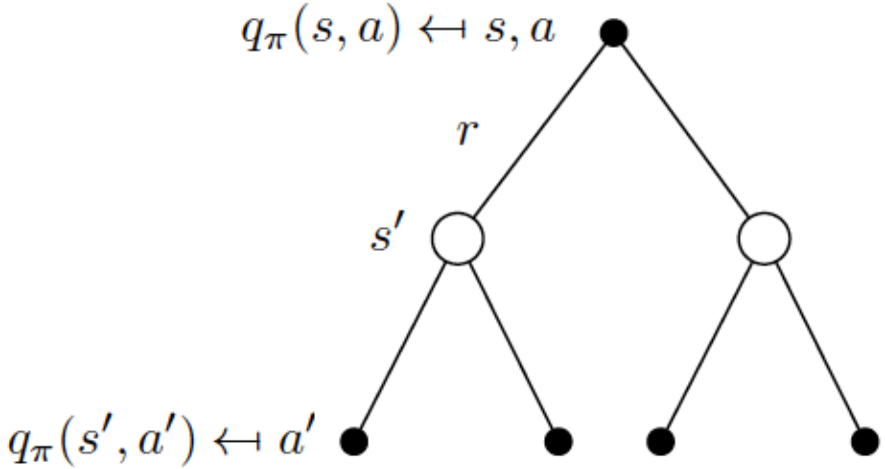
Looking Inside the Expectations



$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) v_\pi(s') \right)$$

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

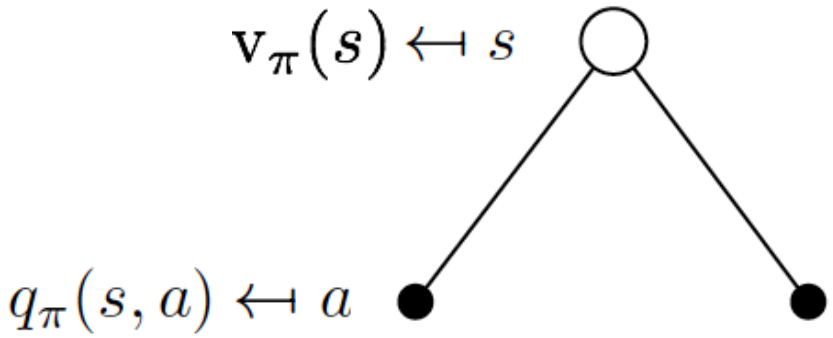
Looking Inside the Expectations



$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

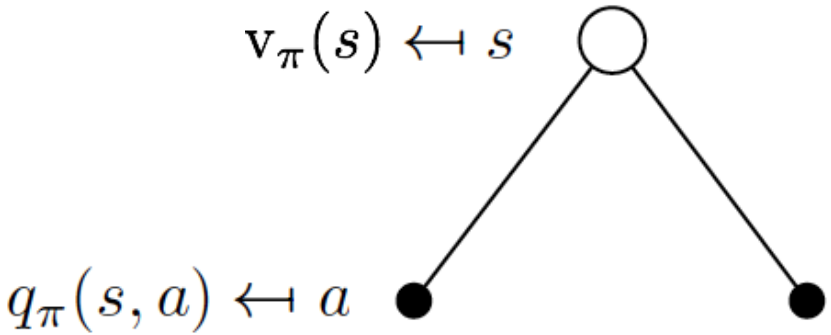
$$\mathrm{q}_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

State and State/Action Value Functions



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

State and State/Action Value Functions

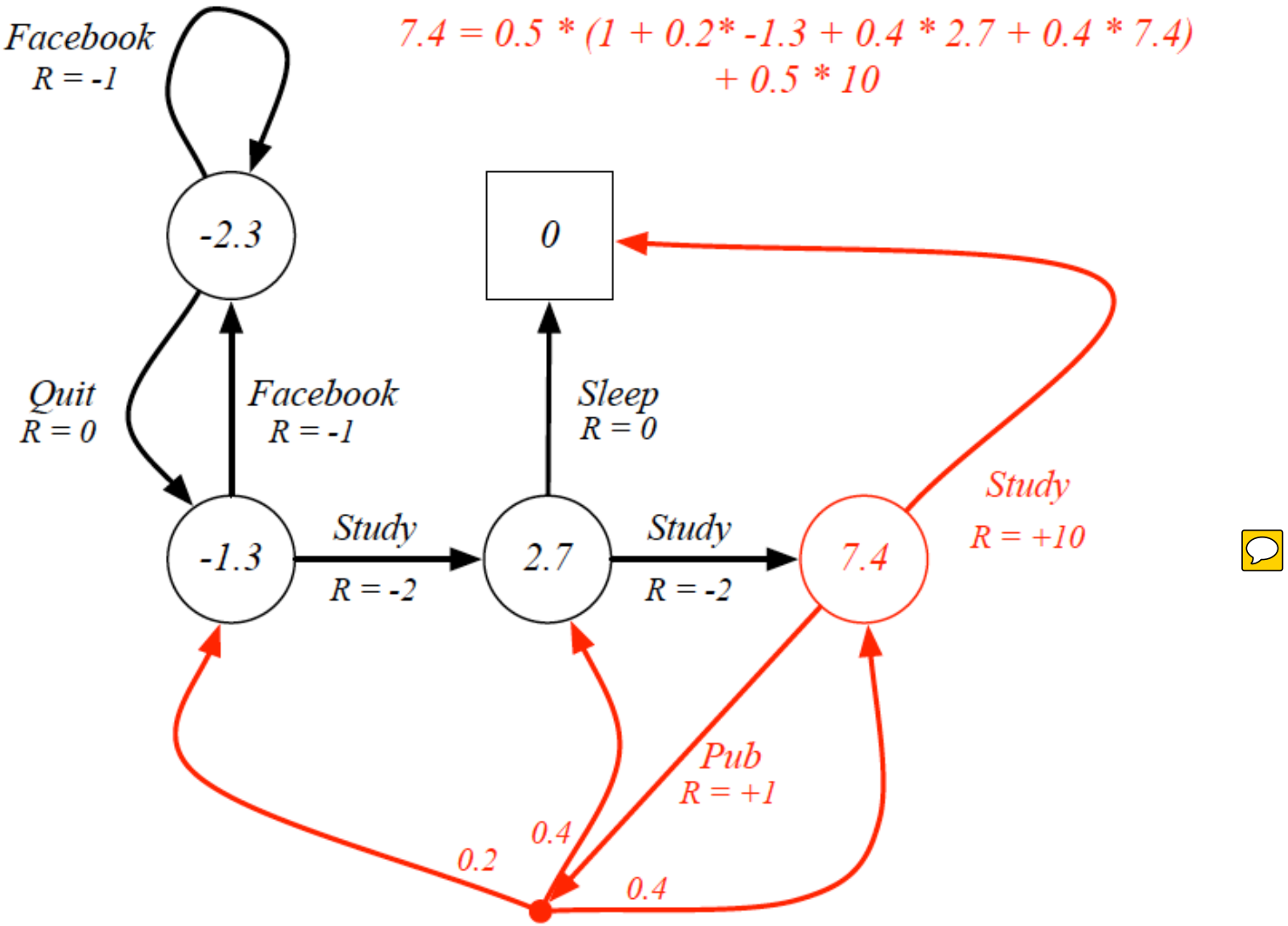


$$\mathrm{v}_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

$$q_\pi(s,a) = r(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) \mathrm{v}_\pi(s')$$

Value Function for the Student MDP



Optimal Value Functions

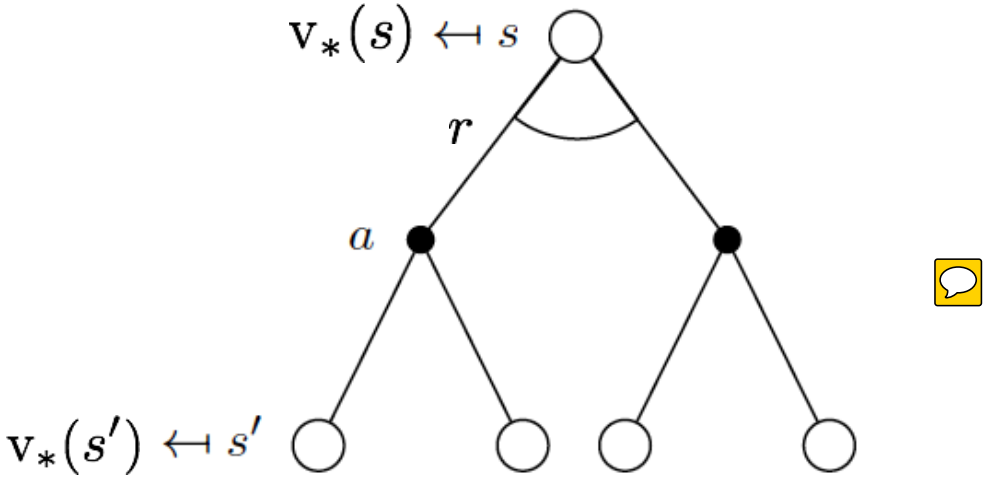
- **Definition**: The *optimal state-value function* $\mathrm{v}_*(s)$ is the maximum value function over all policies

$$\mathrm{v}_*(s) = \max_\pi \mathrm{v}_\pi(s)$$

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

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

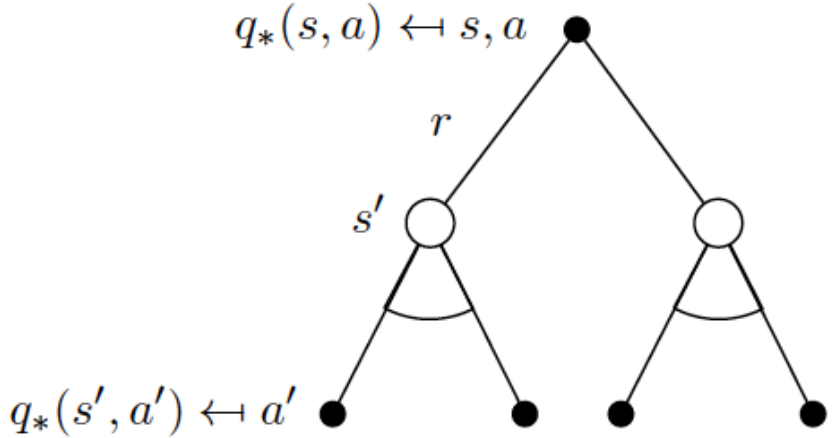
Bellman Optimality Equations for State Value Functions



$$v_*(s) \leftarrow \max_{a \in \mathcal{A}} r(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s'|s,a) v_*(s')$$

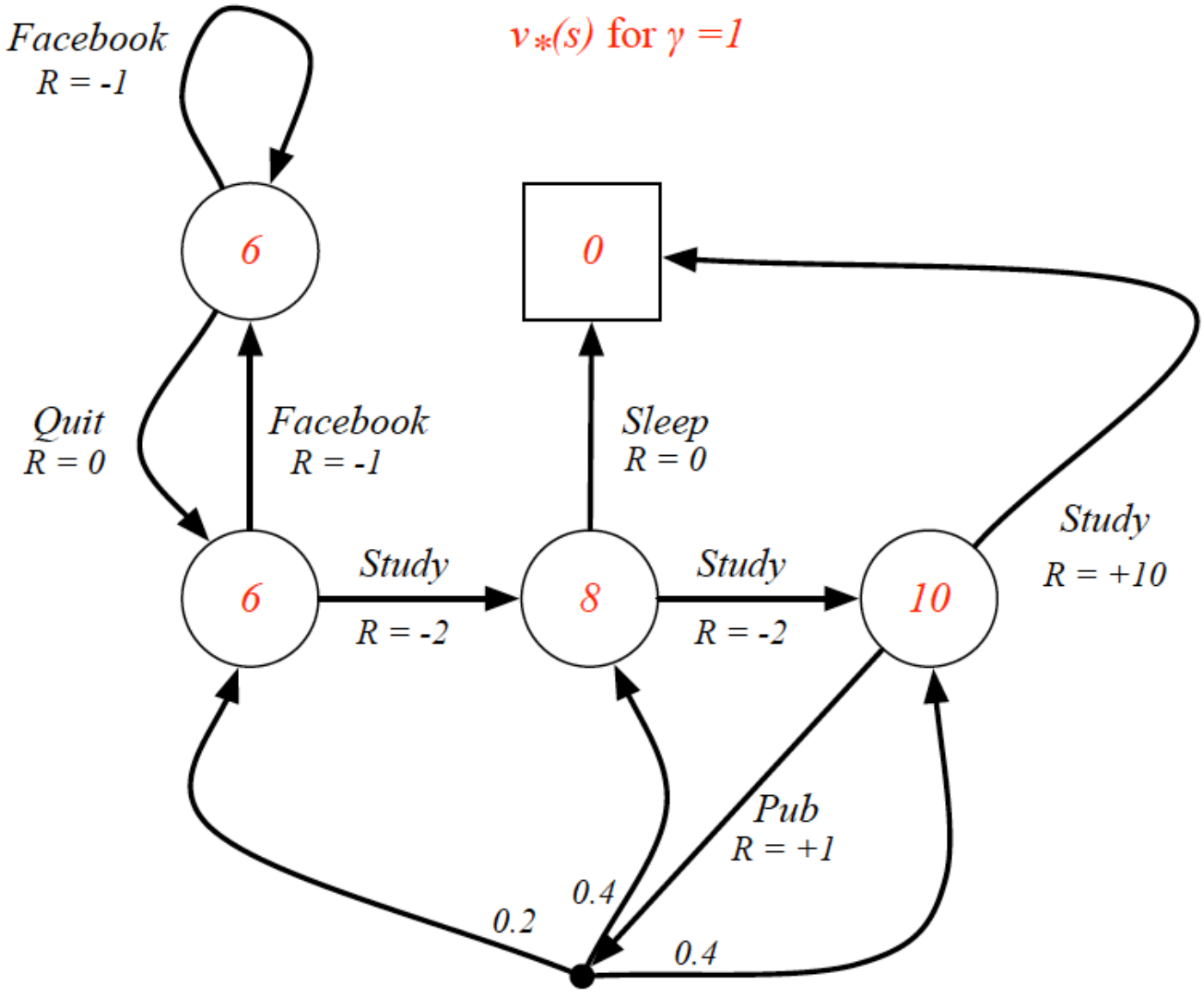
Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. (See Bellman, 1957, Chap. III.3).

Bellman Optimality Equations for State/Action Value Functions

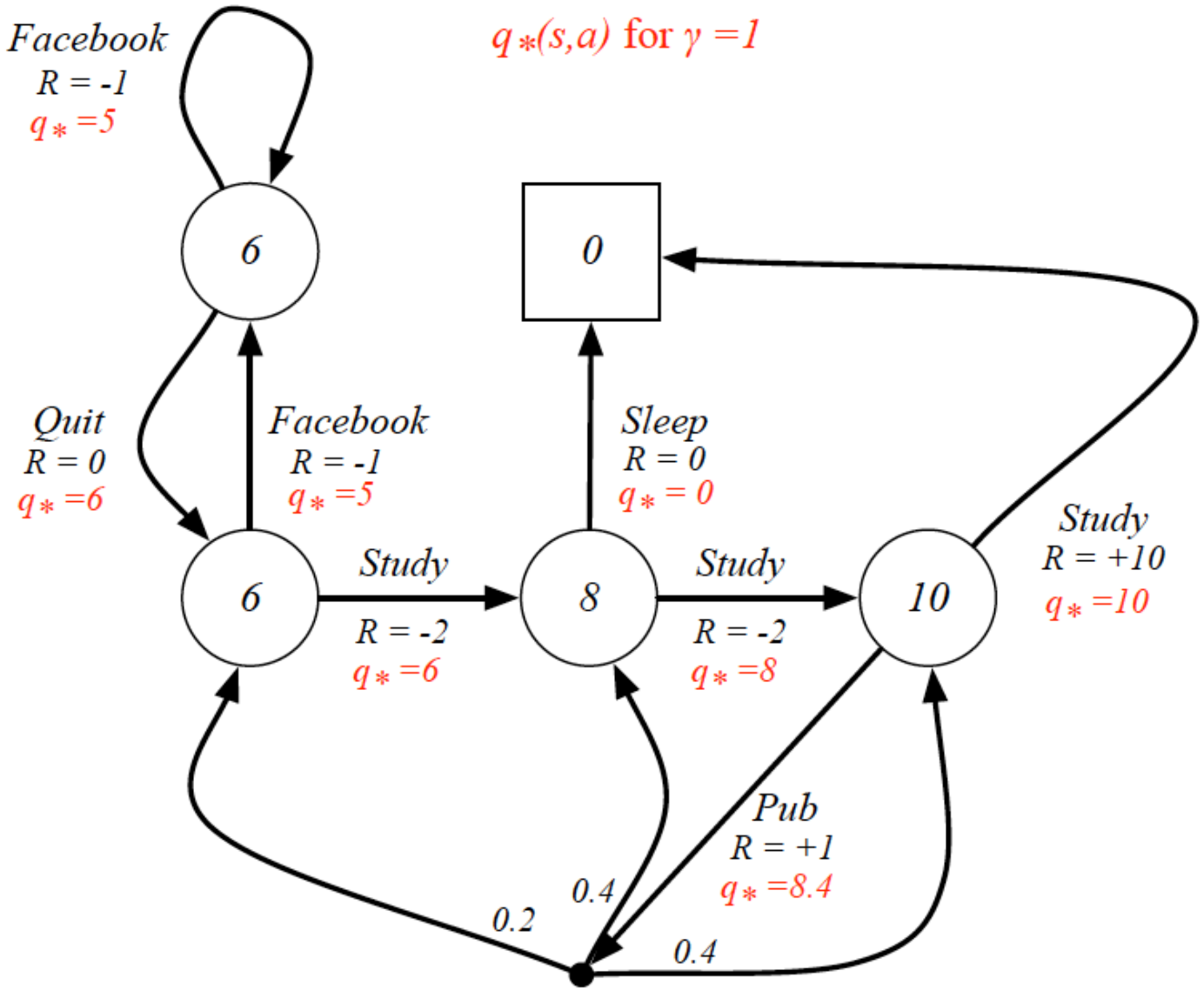


$$q_*(s,a) = r(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) \max_{a'} q_*(s',a')$$

Optimal Value Function for the Student MDP

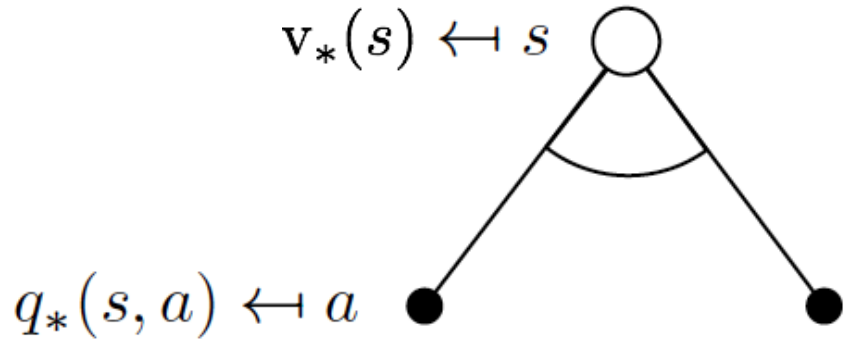


Optimal State/Action Value Function for the Student MDP



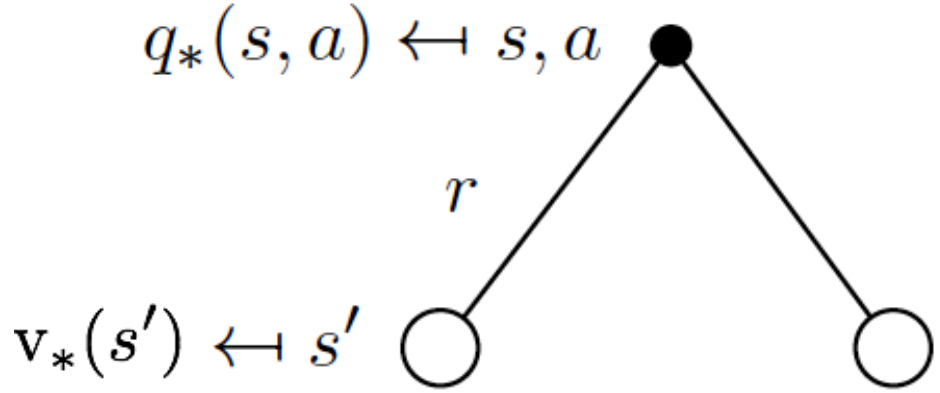
Relating Optimal State and Action Value Functions

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



$$\mathrm{v}_*(s) = \max_a q_*(s,a)$$

Relating Optimal State and Action Value Functions



$$q_*(s,a) = r(s,a) + \gamma \sum_{s' \in S} T(s'|s,a)\mathrm{v}_*(s')$$

Optimal Policy

Define a partial ordering over policies

$$\pi \geq \pi' \text{ if } \mathrm{v}_\pi(s) \geq \mathrm{v}_{\pi'}(s), \forall s$$

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, $\mathrm{v}_{\pi_*}(s) = \mathrm{v}_*(s)$
- All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s, a) = q_*(s, a)$

From Optimal State Value Functions to Optimal Policies

- An optimal policy can be found from $\mathrm{v}_*(s)$ and the model dynamics using one step look ahead, that is, acting greedily w.r.t. $\mathrm{v}_*(s)$

$$\mathrm{v}_*(s) = \max_a r(s,a) + \gamma \sum_{s' \in S} T(s'|s,a)\mathrm{v}_*(s')$$

From Optimal Action Value Functions to Optimal Policies

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

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

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

Solving the Bellman Optimality Equation

- Finding an optimal policy by solving the Bellman Optimality Equation requires the following:
  - accurate knowledge of environment dynamics;
  - we have enough space and time to do the computation;
  - the Markov property.

- How much space and time do we need?
  - polynomial in number of states (tabular methods)
  - BUT, number of states is often huge
  - So exhaustive sweeps of the state space are not possible

Solving the Bellman Optimality Equation

- We usually have to settle for approximations.
- Approximate dynamic programming has been introduced by D. P. Bertsekas and J. N. Tsitsiklis with the use of artificial neural networks for approximating the Bellman function.
- This is an effective mitigation strategy for reducing the impact of dimensionality by replacing the memorization of the complete function mapping for the whole space domain with the memorization of the sole neural network parameters.

Approximation and Reinforcement Learning

- RL methods: Approximating Bellman optimality equations
- Balancing reward accumulation and system identification (model learning) in case of unknown dynamics

- The on-line nature of reinforcement learning makes it possible to approximate optimal policies in ways that put more effort into learning to make good decisions for frequently encountered states, at the expense of less effort for infrequently encountered states.

Summary

- Markov Decision Processes
- Value functions and Optimal Value functions
- Bellman Equations

So far finite MDPs with known dynamics

Next Lecture

- Countably infinite state and/or action spaces
- Continuous state and/or action spaces
  - Closed form for linear quadratic model (LQR)
- Continuous time
  - Requires partial differential equations