Carnegie Mellon
School of Computer Science

Deep Reinforcement Learning and Control

Markov Decision Processes

Lecture 2, CMU 10703

Katerina Fragkiadaki

Supervision for learning goal-seeking behaviors

1. Learning from rewards while interacting with the environment

   Evaluative feedback: the environment provides signal whether actions are good or bad. E.g., your advisor tells you if your research ideas are worth pursuing

2. Learning from expert demonstrations

   Instructive feedback: the expert directly suggests correct actions, i.e., your (oracle) advisor directly suggests to you ideas that are worth pursuing

Note: Evaluative feedback depends on the current policy the agent has: if you never suggest good ideas, you will never have the chance to know they are worthwhile. Instructive feedback is independent of the agent's policy.

3. Learning from specifications of optimal behavior in natural language (we are not going to look into this for a while in the course)

Supervision for learning goal-seeking behaviors

1. **Learning from rewards while interacting with the environment**

   Evaluative feedback: the environment provides signal whether actions are good or bad. E.g., your advisor tells you if your research ideas are worth pursuing

2. Learning from expert demonstrations

   Instructive feedback: the expert directly suggests correct actions, i.e., your (oracle) advisor directly suggests to you ideas that are worth pursuing

Note: Evaluative feedback depends on the current policy the agent (you) have: if you never suggest good ideas, you will never have the chance to know they are worthwhile. Instructive feedback is independent of the agent's current policy

3. Learning from specifications of optimal behavior in natural language (we are not going to look into this for a while in the course)

Reinforcement

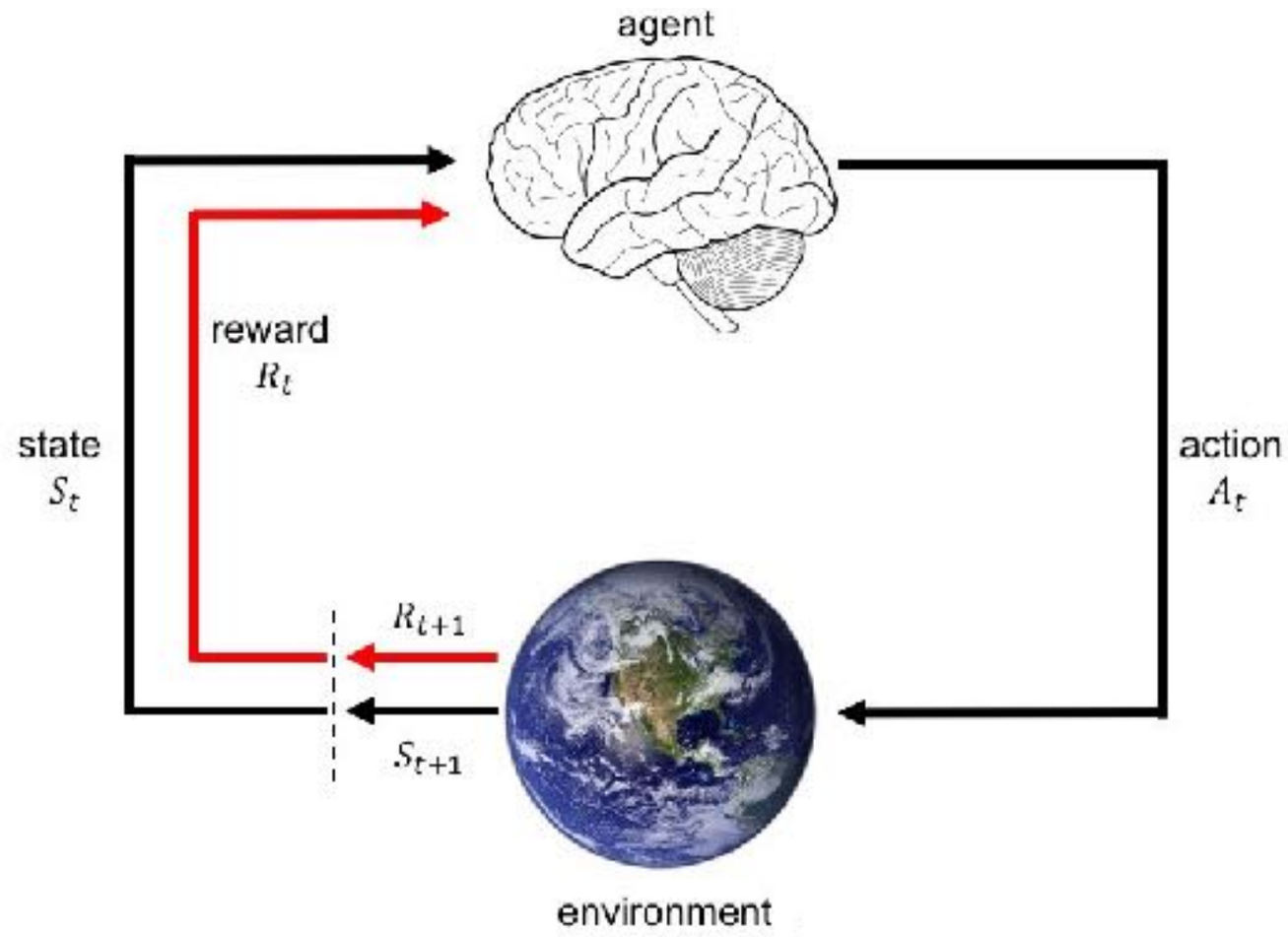
Behavior is primarily shaped by reinforcement rather than free-will.

Positive reinforcement is the strengthening of behavior by the occurrence of some event (e.g., praise after some behavior is performed), whereas negative reinforcement is the strengthening of behavior by the removal or avoidance of some aversive event (e.g., opening and raising an umbrella over your head on a rainy day is reinforced by the cessation of rain falling on you).

B.F. Skinner 1
904-1990
Harvard psychology

Behaviors that result in praise/pleasure tend to repeat, behaviors that result in punishment/pain tend to become extinct.

Reinforcement learning



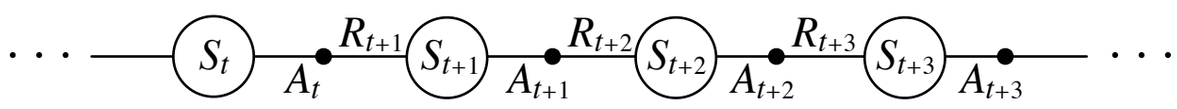
Agent and environment interact at discrete time steps: $t = 0, 1, 2, 3, \ldots$

Agent observes state at step t: $S_t \in \mathcal{S}$

produces action at step t : $A_t \in \mathcal{A}(S_t)$

gets resulting reward: $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$

and resulting next state: $S_{t+1} \in \mathcal{S}^+$

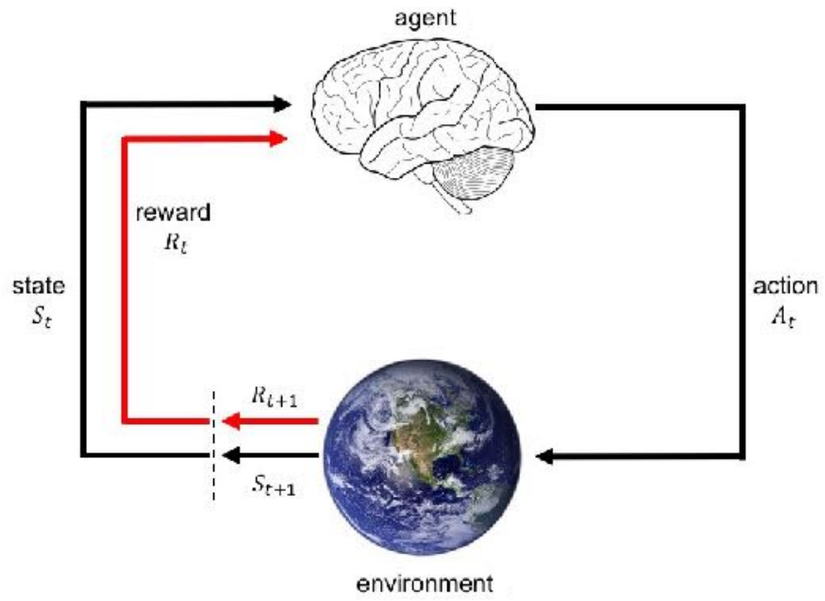


Reinforcement learning-and why we like it

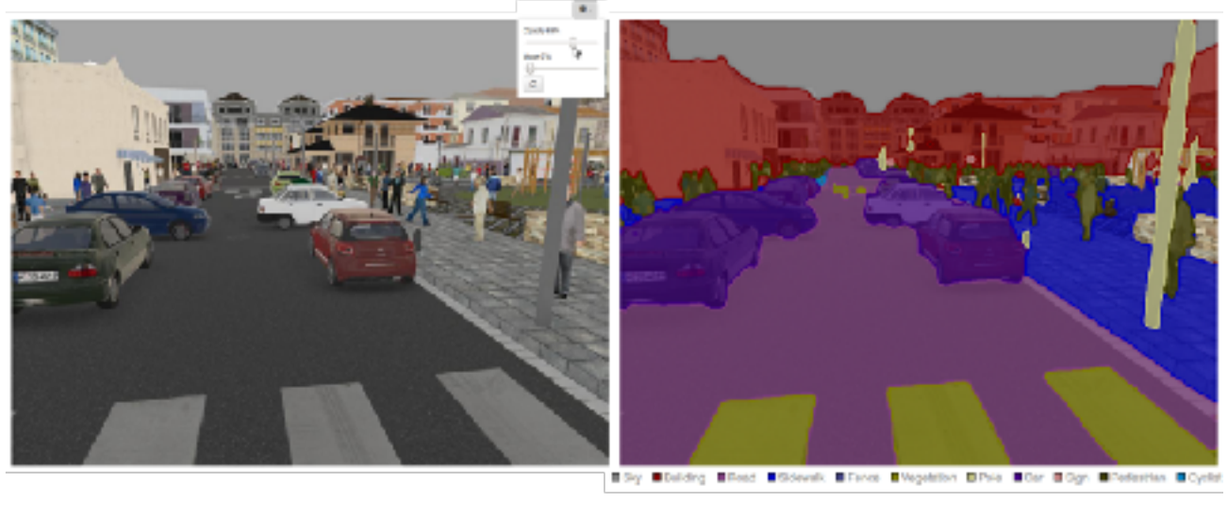
Learning policies that maximize a reward function by interacting with the world

- It is considered the most biologically plausible form of learning

- It addresses the full problem of making artificial agents that act in the world end-to-end, so it is driven by the right loss function

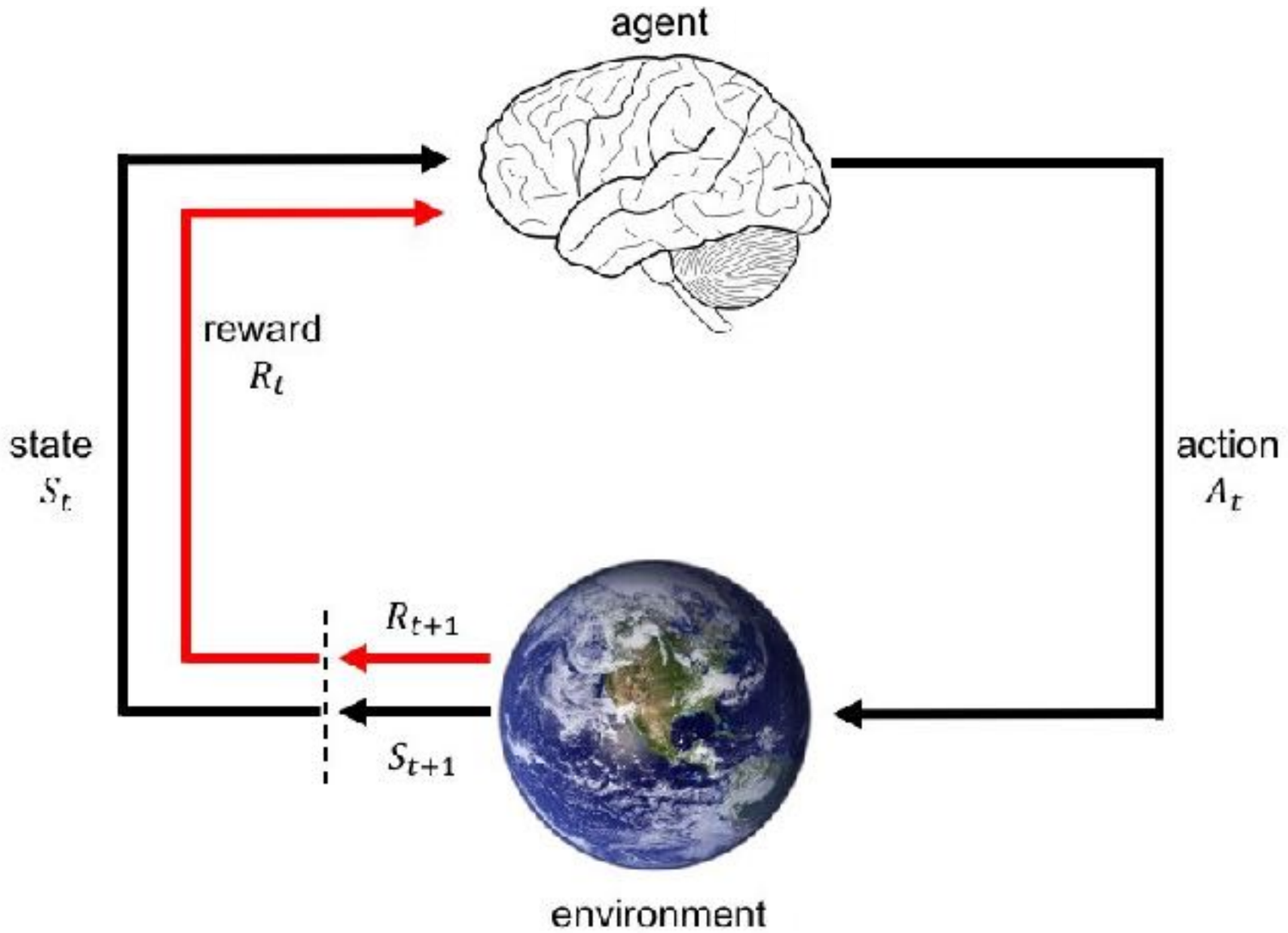


…in contrast to, for example, pixel labelling



Reinforcement learning

Learning policies that maximize a reward function by interacting with the world



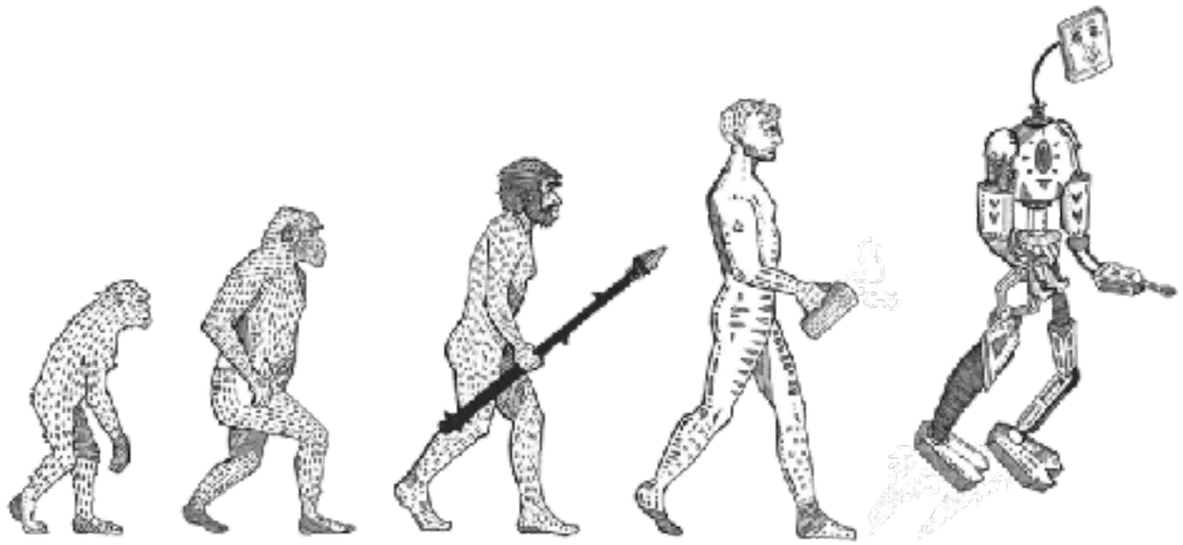
State estimation: from observations to states

- Observations a.k.a. sensations: the (raw) input of the agent's sensors, images, tactile singal, waveforms, etc.

- A state captures whatever information is available to the agent at 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.

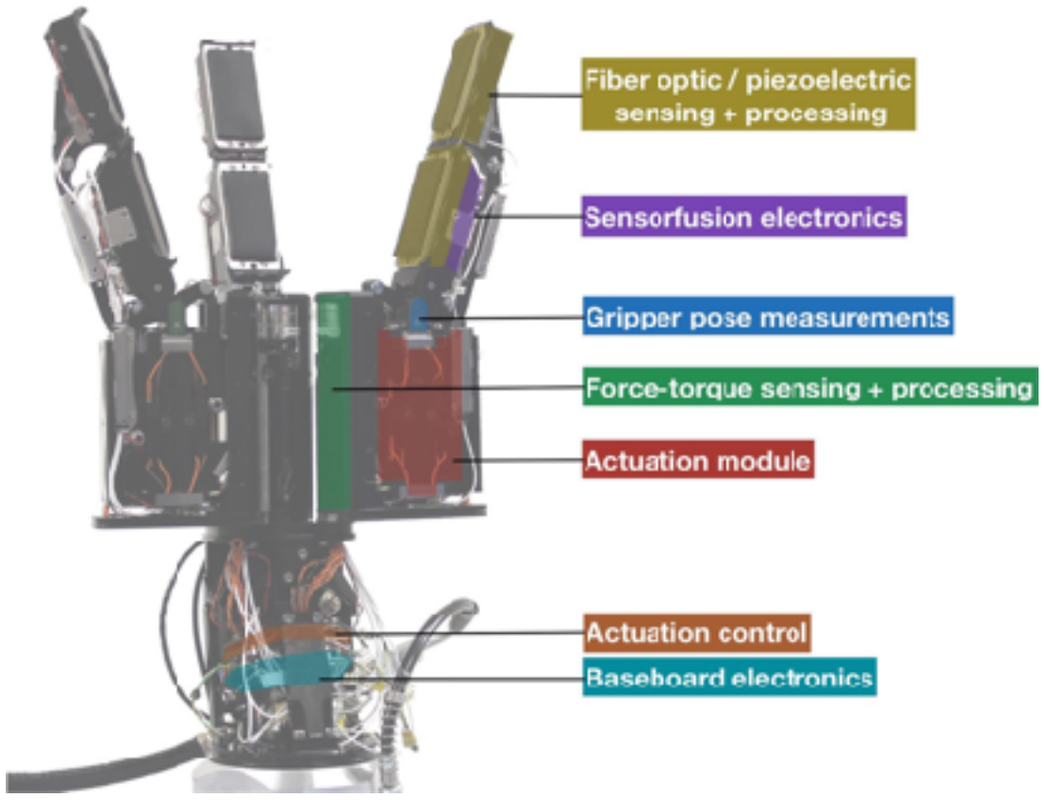
Agent

An entity that is equipped with **sensors**, in order to sense the environment, **end effectors** in order to act in the environment, and **goals** that she wants to achieve



Actions A_t

They are used by the agent to interact with the world. They can have many different temporal granularities and abstractions.



Actions can be defined to be

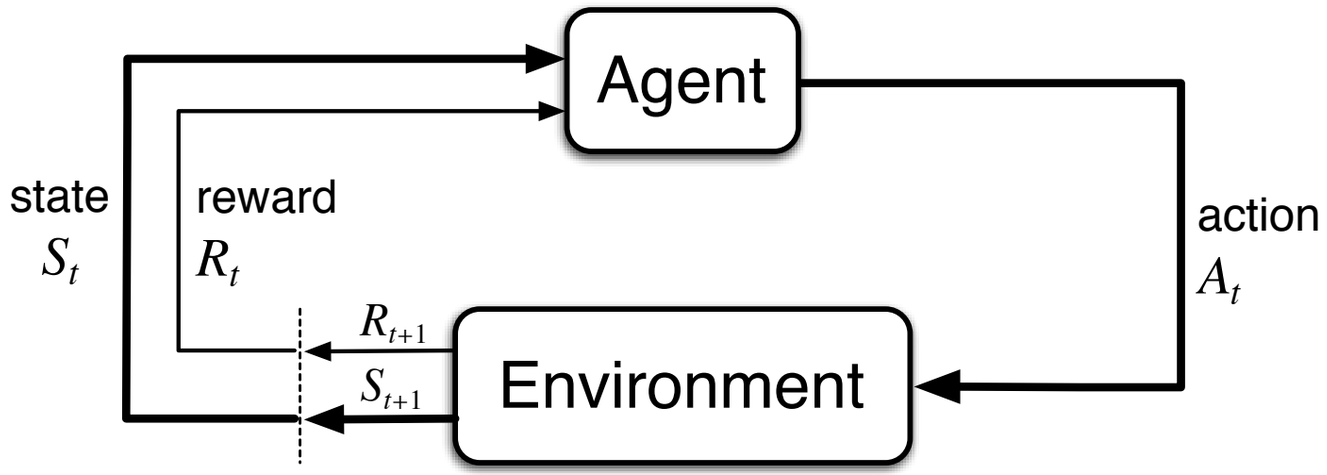
- The instantaneous torques applied on the gripper
- The instantaneous gripper translation, rotation, opening
- Instantaneous forces applied to the objects
- Short sequences of the above

What limits the temporal extent of actions?

Closed loop sensing and acting

Imagine an agent that wants to pick up an object and has a policy that predicts what the actions should be for the next 2 secs ahead. This means, for the next 2 secs we switch off the sensors, and just execute the predicted actions. In the next second, due to imperfect sensing, the object is about to fall over!

Sensing is always imperfect. Our excellent motor skills are due to continuous sensing and updating of the actions. So this loop is in fact extremely short in time.



Rewards $\mathbf{R}_t$

They are scalar values provided by the environment to the agent that indicate whether goals have been achieved, e.g., ***1 if goal is achieved, 0 otherwise, or -1 for overtime step the goal is not achieved***

- Rewards specify what the agent needs to achieve, not how to achieve it.

- The simplest and cheapest form of supervision, and surprisingly general: All of what we mean by goals and purposes can be well thought of as the maximization of the cumulative sum of a received scalar signal (reward)

Returns $\mathbf{G}_t$

Goal-seeking behavior of an agent can be formalized as the behavior that seeks maximization of the expected value of the ***cumulative sum of (potentially time discounted) rewards, we call it return. We want to maximize returns.***

Returns G_t - Episodic tasks

Episodic tasks: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze

In episodic tasks, we almost always use simple *total reward*:

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_T$$

where T is a final time step at which a **terminal state** is reached, ending an episode.

Returns G_t - Continuing tasks

Continuing tasks: interaction does not have natural episodes, but just goes on and on...

In episodic tasks, we almost always use simple *total reward*:

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

Why temporal discounting? A sequence of interactions based on which the reward will be judged at the end is called episode. Episodes can have finite or infinite length. For infinite length, the undercounted sum blows up, thus we add discounting $\gamma < 1$ to prevent this, and treat both cases in a similar manner.

Dynamics p a.k.a. the Model

- How the states and rewards change given the actions of the agent

$$\mathrm{p}(s', r \,|\, s, a) = \mathbb{P}\{S_t = s', \mathrm{R}_\mathrm{t} = r \,|\, \mathrm{S}_{t-1} = s, \mathrm{A}_{t-1} = a\}$$

- Transition function or next step function:

$$\mathrm{T}(s' \,|\, s, a) = \mathrm{p}(s' \,|\, s, a) = \mathbb{P}\{S_t = s' \,|\, \mathrm{S}_{t-1} = s, \mathrm{A}_{t-1} = a\} = \sum_{r \in \mathbb{R}} \mathrm{p}(s', r \,|\, s, a)$$

Model-free VS model-based RL

- Model-based RL: dynamics are known or are estimated, and are used for learning the policy
- Model-free: we do not know the dynamics, and we do not attempt to estimate them
- Q:Is AlphaGoZero model-based or model-free?
- What about TD-gammon?

Model-free VS model-based RL

- An estimated (learned) model is never perfect.

  ``All models are wrong but some models are useful”

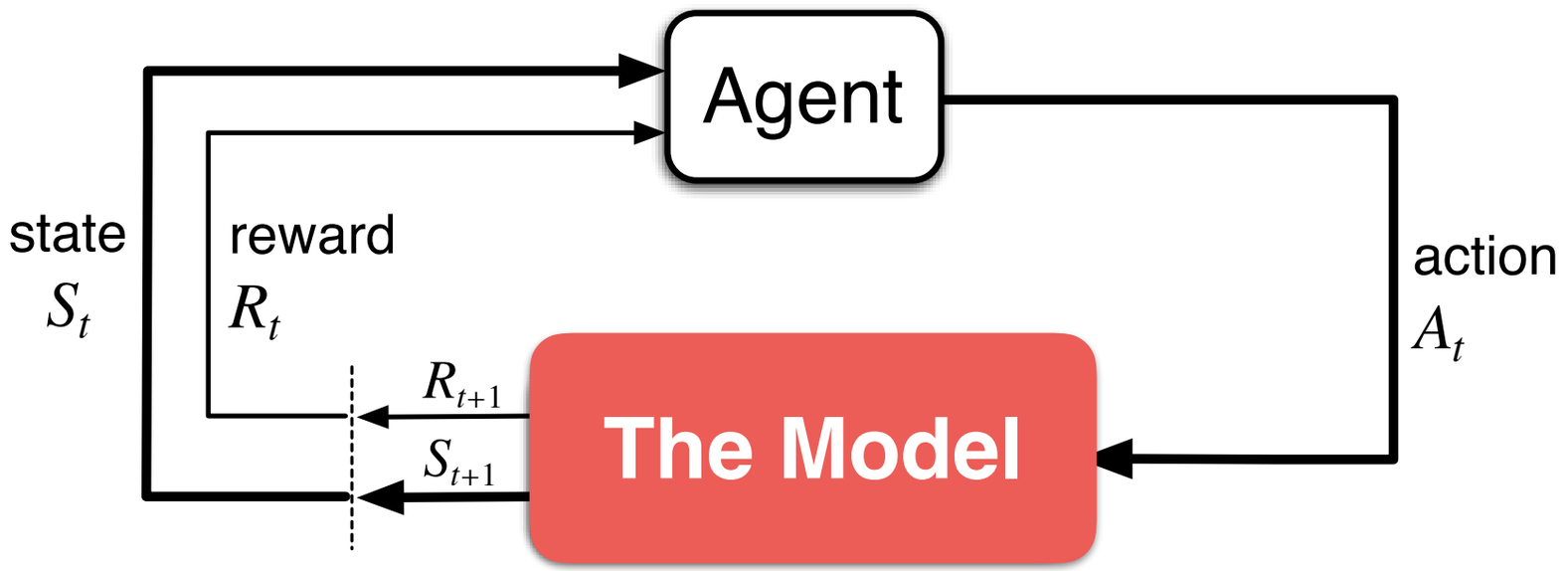
George Box

- Due to model error model-free methods often achieve better policies though are more time consuming. Later in the course, we will examine use of (inaccurate) learned models and ways not to hinder the final policy while still accelerating learning

Planning

Planning: unrolling (querying) a model forward in time and selecting the best action sequence that satisfies a specific goal

Plan: a sequence of actions

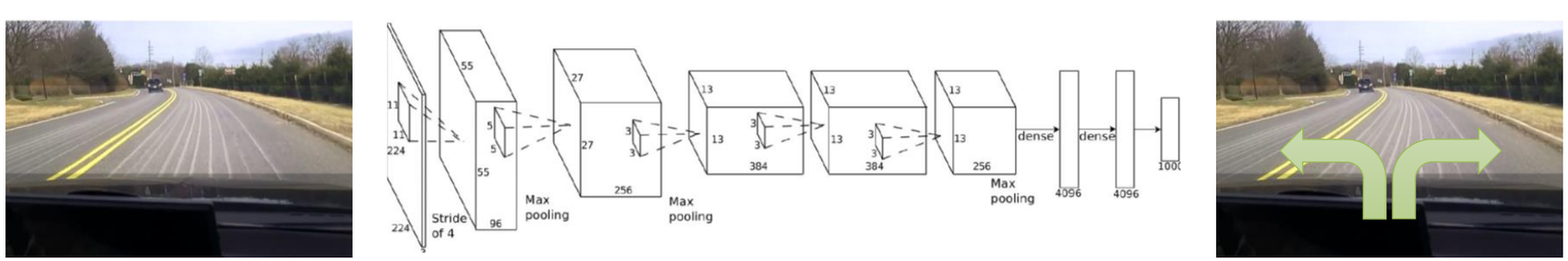


Policy π

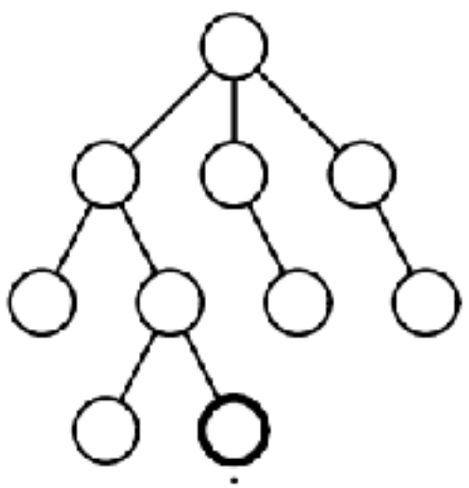
A mapping function from states to actions of the end effectors.

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

It can be a shallow or deep function mapping,



or it can be as complicated as involving a tree look-ahead search



Definitions

Agent: an entity that is equipped with **sensors**, in order to sense the environment, and **end effectors** in order to act in the environment, and goals that he wants to achieve

Policy: a mapping function from observations (sensations, inputs of the sensors) to actions of the end effectors.

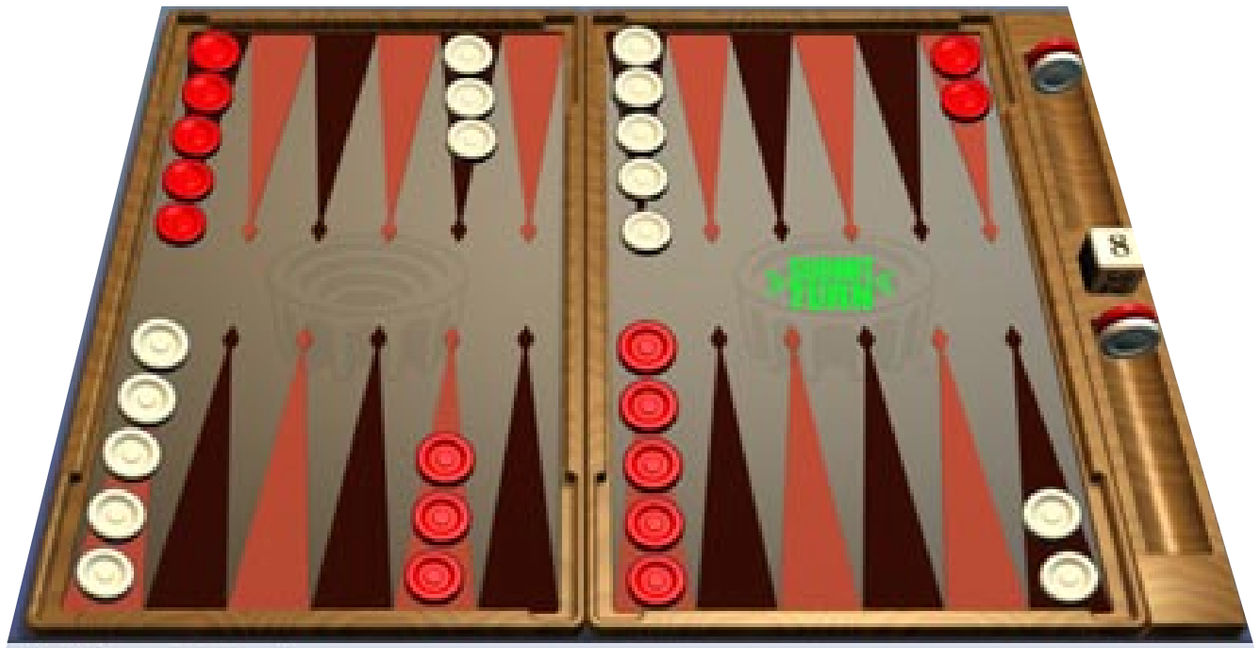
Model: the mapping function from states/observations and actions to future states/observations

Planning: unrolling a model forward in time and selecting the best action sequence that satisfies a specific goal

Plan: a sequence of actions

Backgammon

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

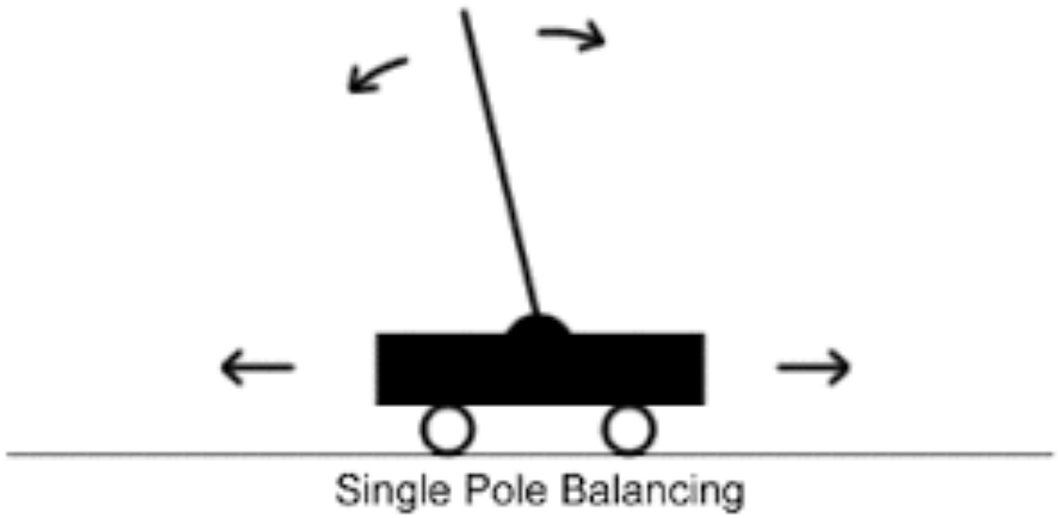


Learning to Drive

- States: Road traffic, weather, time of day
- Actions: steering wheel, break
- Rewards:
  - +1 reaching goal not over-tired
  - -1: honking from surrounding drivers
  - -100: collision

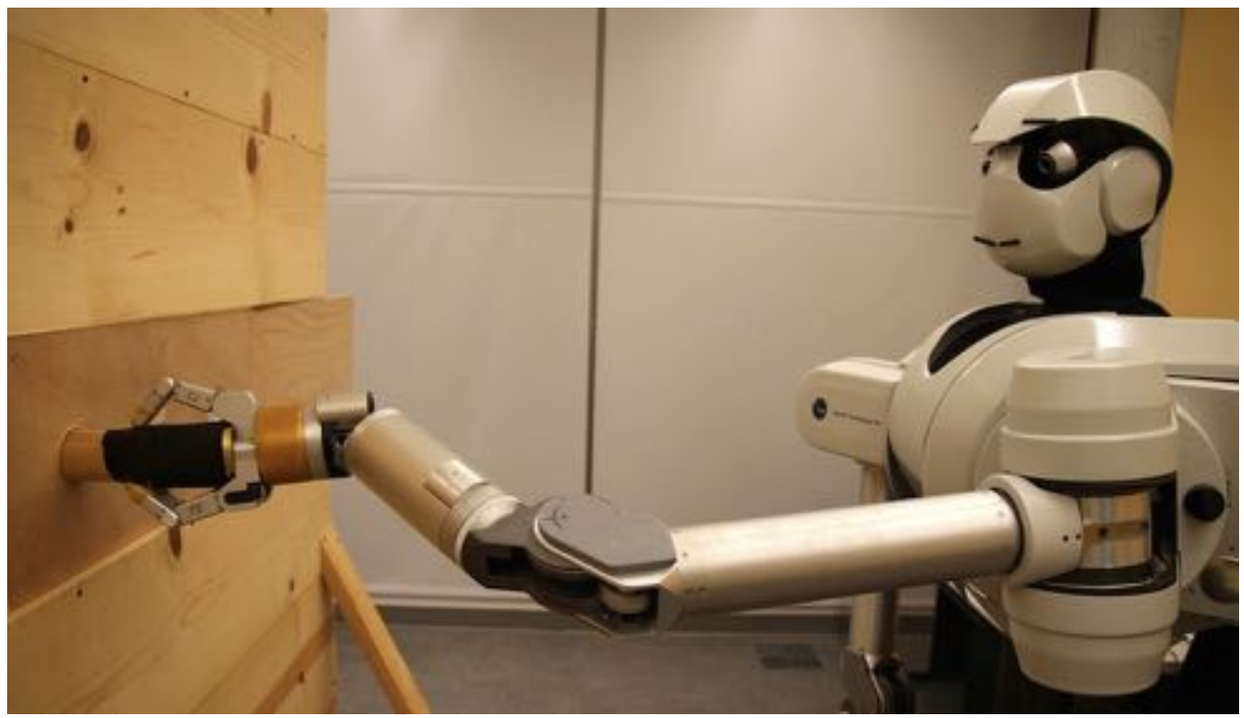
Cart Pole

- 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 (7DOF)
- Actions: Torques on joints
- Rewards: Penalize jerky motions, inversely proportional to distance from target pose



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

For now we will assume that the world is not diverse at all: It has a finite number of states and in fact so few that we can easily enumerate them and store their values in a table. That is why the method we will discuss in the next few lectures are also known as tabular methods.

Finite Markov Decision Process

A **Finite 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 $\gamma \in [0, 1]$

The recycling robot MDP

- At each step, robot has to decide whether it should (1) actively search for a can, (2) wait for someone to bring it a can, or (3) go to home base and recharge.
- Searching is better but runs down the battery; if runs out of power while searching, has to be rescued (which is bad).
- Decisions made on basis of current energy level: `high`, `low`.
- Reward = number of cans collected

The recycling robot MDP

$$\mathcal{S} = \{\texttt{high}, \texttt{low}\}$$

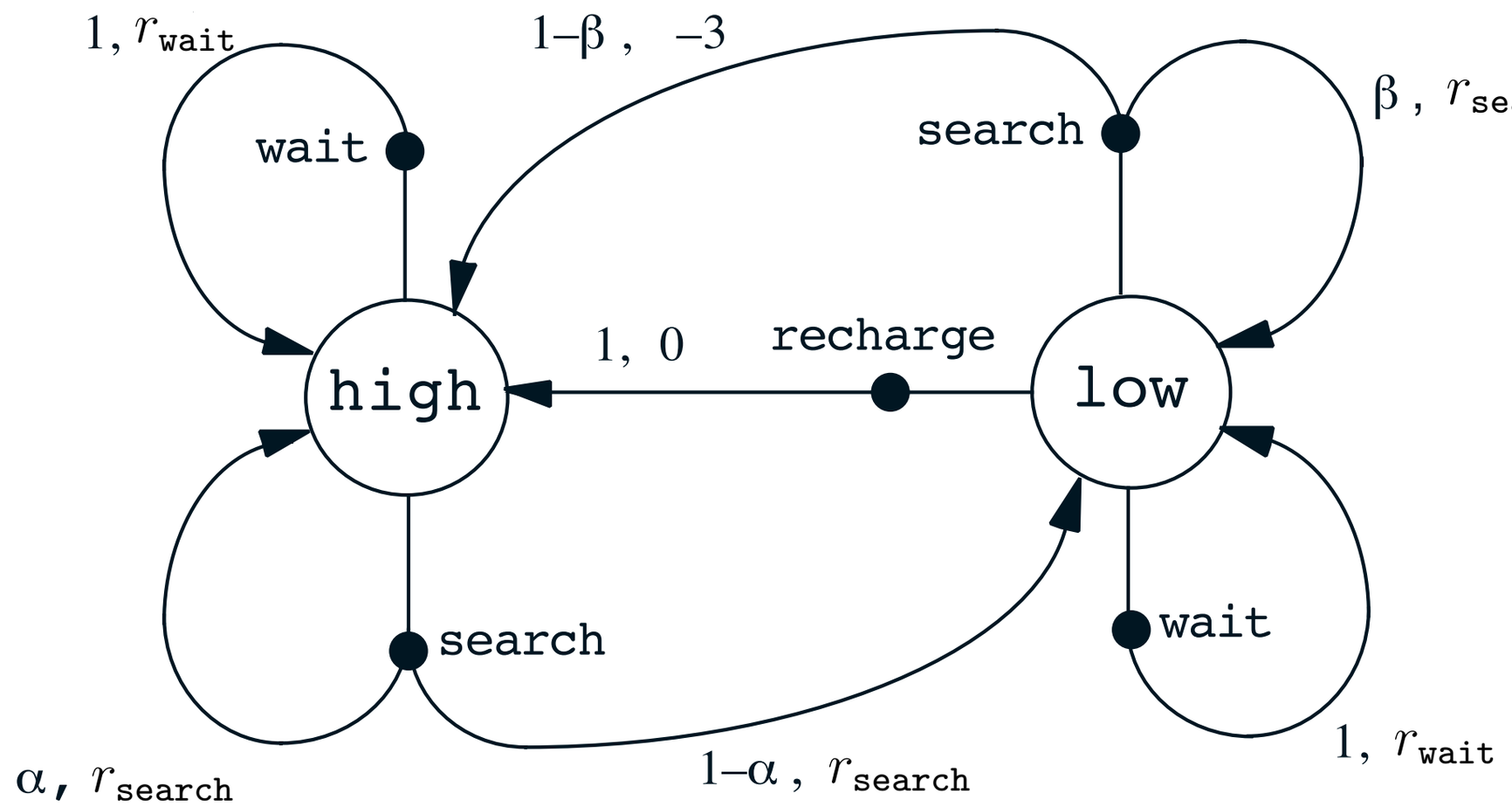
$$\mathcal{A}(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$$

$$\mathcal{A}(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$$

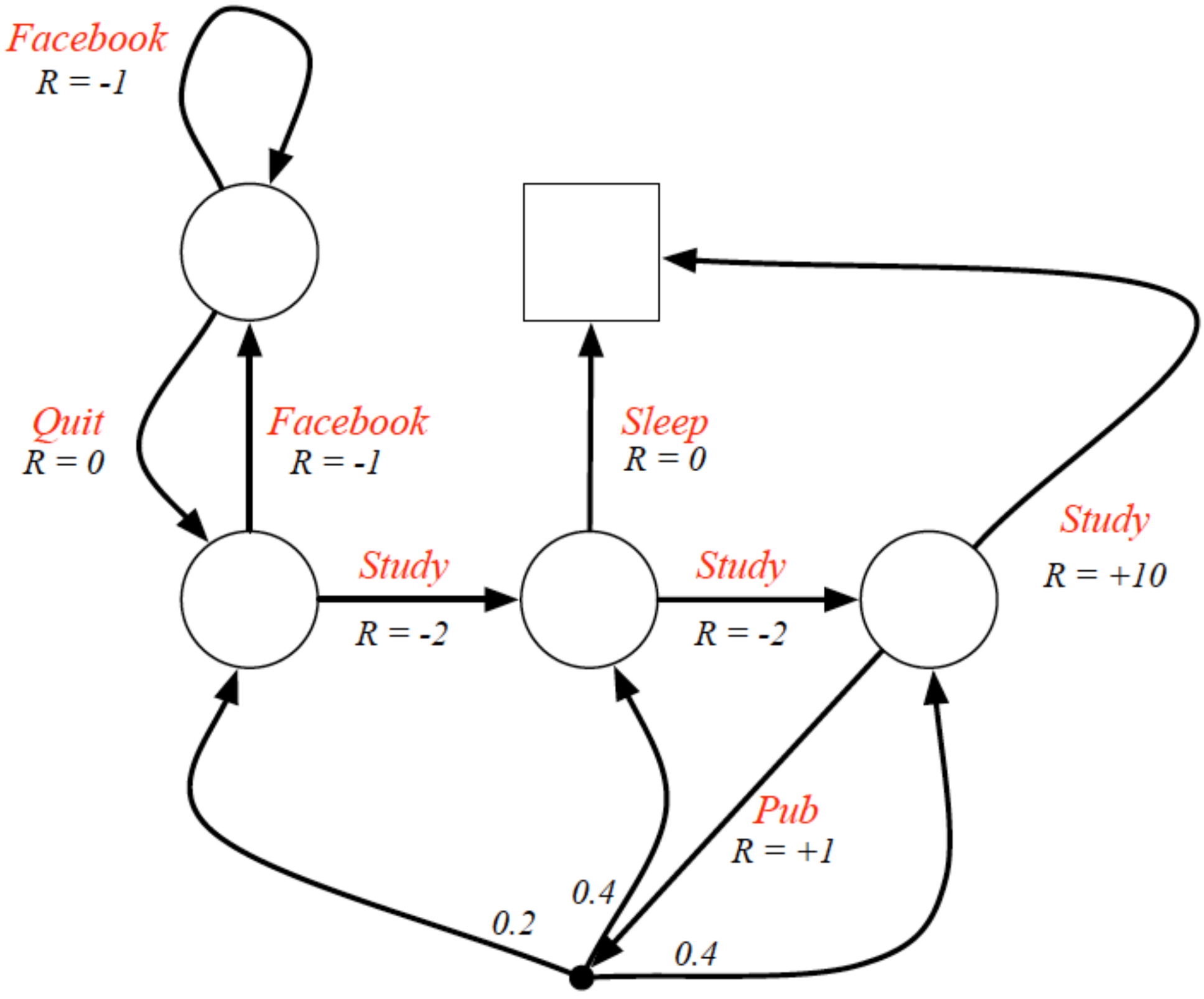
$r_{\texttt{search}}$ = expected no. of cans while searching

$r_{\texttt{wait}}$ = expected no. of cans while waiting

$$r_{\texttt{search}} > r_{\texttt{wait}}$$



The Student MDP



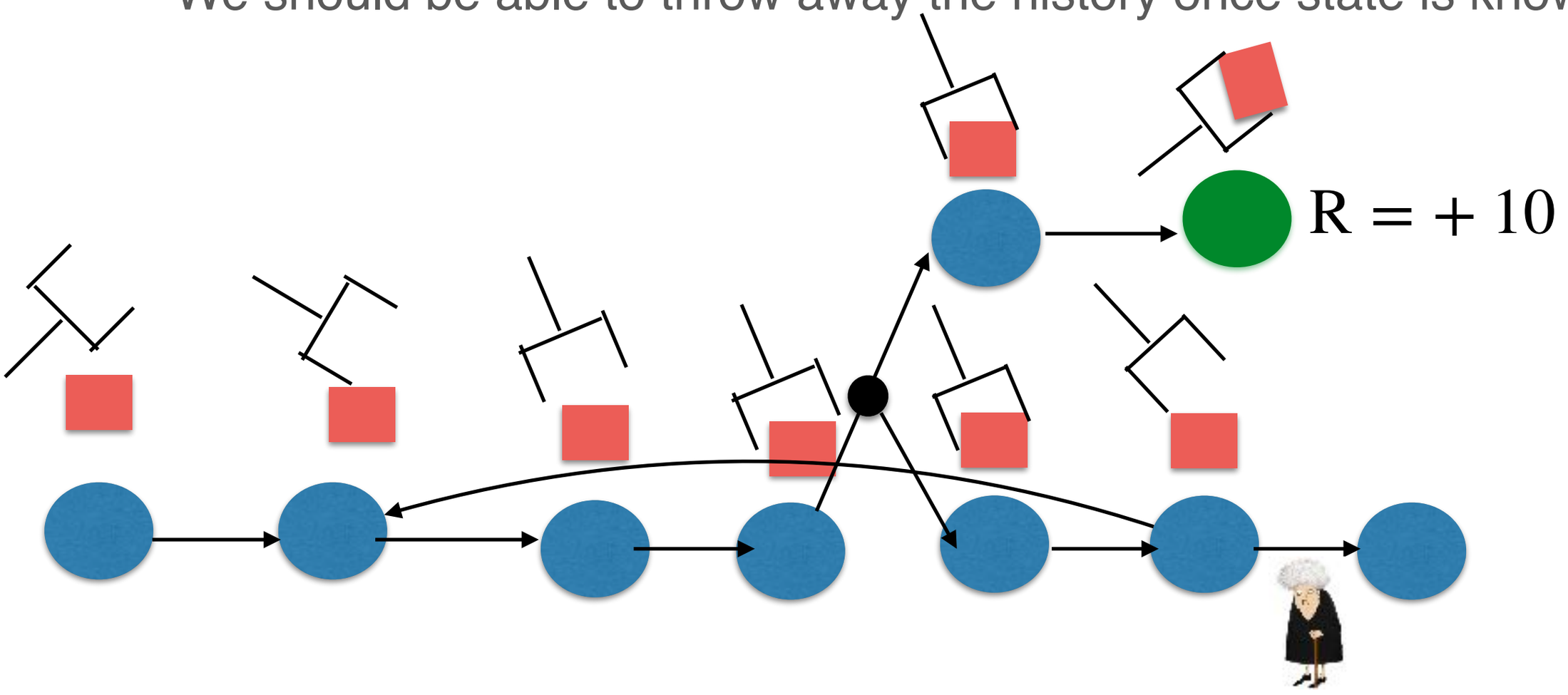
Markov property and should we worry about it?

- 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 $s' \in \mathcal{S}, r \in \mathcal{R}$, and all histories

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



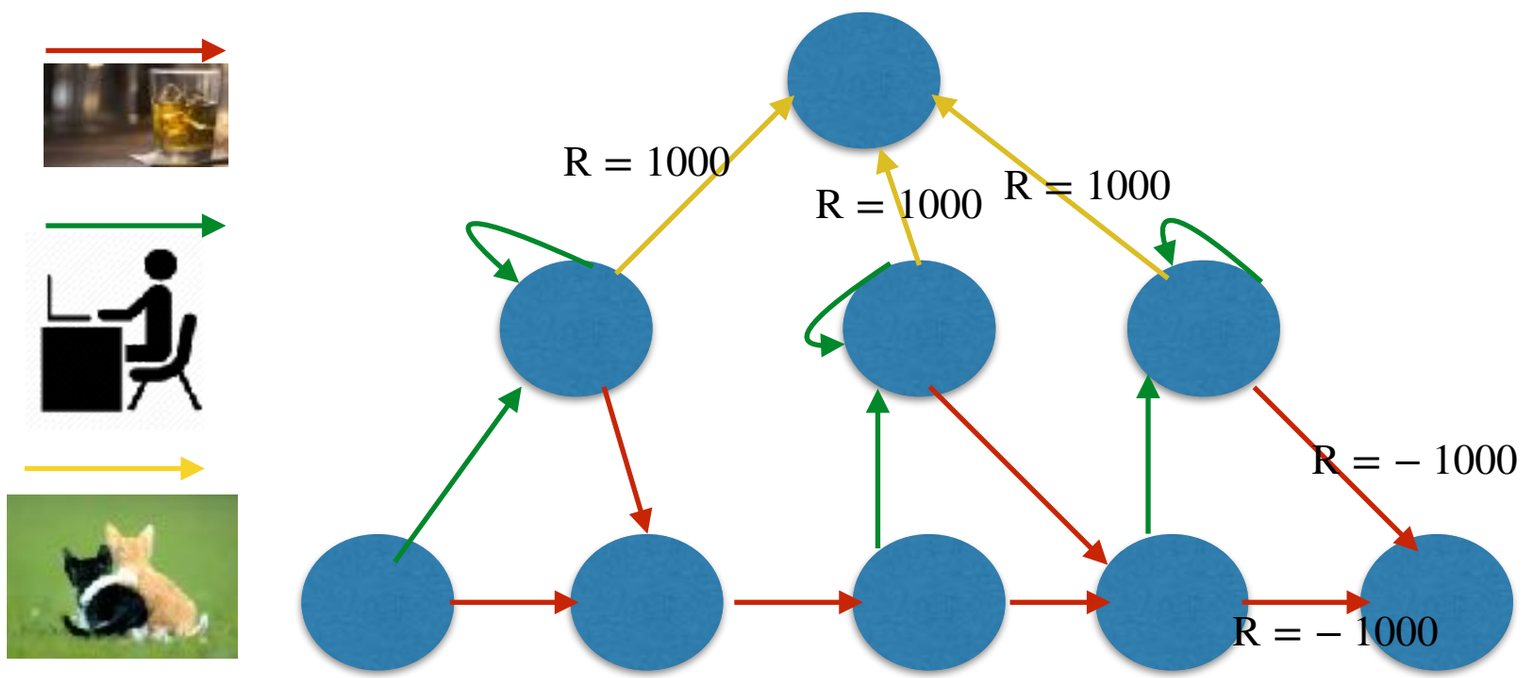
Markov property and should we worry about it?

- 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 $s' \in \mathcal{S}, r \in \mathcal{R}$, and all histories

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



With enough state information most things can be made Markov

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, for a given policy.
- Optimal value functions measure the best possible goodness of states or state/action pairs under all possible policies.

Value Functions

- Prediction: For a given policy, estimate state and state/action value functions
- Control: Estimate the optimal state and state/action value functions

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

Lower case for ideal **expected** values, upper case for their estimates:

$V_t(s)$ $Q_t(s,a)$

What exactly is stochastic here?

Value Functions are Expected Returns

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 Expected Returns

- **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

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

Value Functions are Expected Returns

- The value of a state, given a policy:

  $v_\pi(s) = \mathbb{E}\{G_t \mid S_t = s, A_{t:\infty} \sim \pi\} \qquad v_\pi : \mathcal{S} \to \Re$

- The value of a state-action pair, given a policy:

  $q_\pi(s,a) = \mathbb{E}\{G_t \mid S_t = s, A_t = a, A_{t+1:\infty} \sim \pi\} \qquad q_\pi : \mathcal{S} \times \mathcal{A} \to \Re$

- The optimal value of a state:

  $v_*(s) = \max_\pi v_\pi(s) \qquad v_* : \mathcal{S} \to \Re$

- The optimal value of a state-action pair:

  $q_*(s,a) = \max_\pi q_\pi(s,a) \qquad q_* : \mathcal{S} \times \mathcal{A} \to \Re$

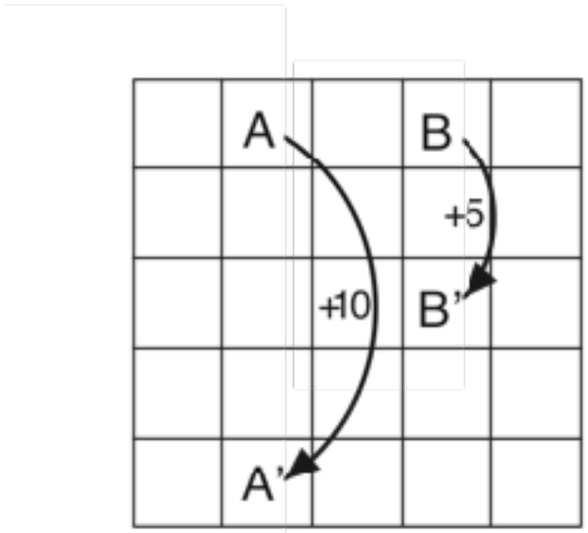
- Optimal policy: π_* is an optimal policy if and only if

  $\pi_*(a|s) > 0 \text{ only where } q_*(s,a) = \max_b q_*(s,b) \qquad \forall s \in \mathcal{S}$

  - in other words, π_* is optimal iff it is *greedy* wrt q_*

Gridworld-value function

- Actions: `north`, `south`, `east`, `west`; deterministic.
- If would take agent off the grid: no move but reward = –1
- Other actions produce reward = 0, except actions that move agent out of special states A and B as shown.



(a)

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

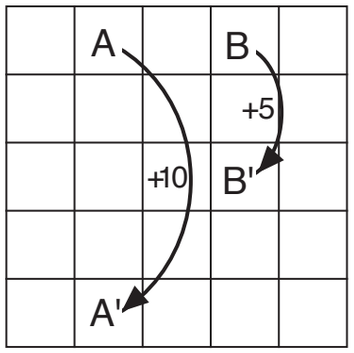
(b)

State-value function for equiprobable random policy;
$\gamma = 0.9$

Gridworld - optimal value function

Any policy that is greedy with respect to v_* is an optimal policy.

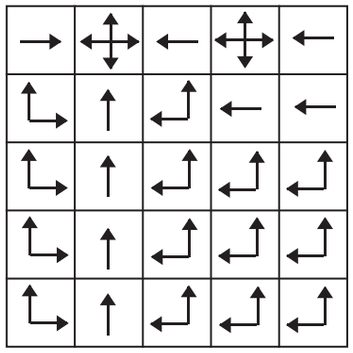
Therefore, given v_*, one-step-ahead search produces the long-term optimal actions.



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) v_*



c) π_*

Optimal action-value functions

Given q_*, the agent does not even have to do a one-step-ahead search:

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, **we do not need the dynamics!**

Bellman Expectation Equation

$$
\begin{aligned}
G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} \mathsf{L} \\
&= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} \mathsf{L} \right) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

Bellman Expectation Equation

$$
\begin{aligned}
G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} \mathrm{L} \\
&= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} \mathrm{L} \right) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

So:
$$
\begin{aligned}
v_\pi(s) &= E_\pi \left\{ G_t \,\middle|\, S_t = s \right\} \\
&= E_\pi \left\{ R_{t+1} + \gamma v_\pi \left(S_{t+1} \right) \middle|\, S_t = s \right\}
\end{aligned}
$$

The value function can be decomposed into two parts:

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

Bellman Expectation Equation

$$
\begin{aligned}
G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} \mathrm{L} \\
&= R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} \mathrm{L} \right) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

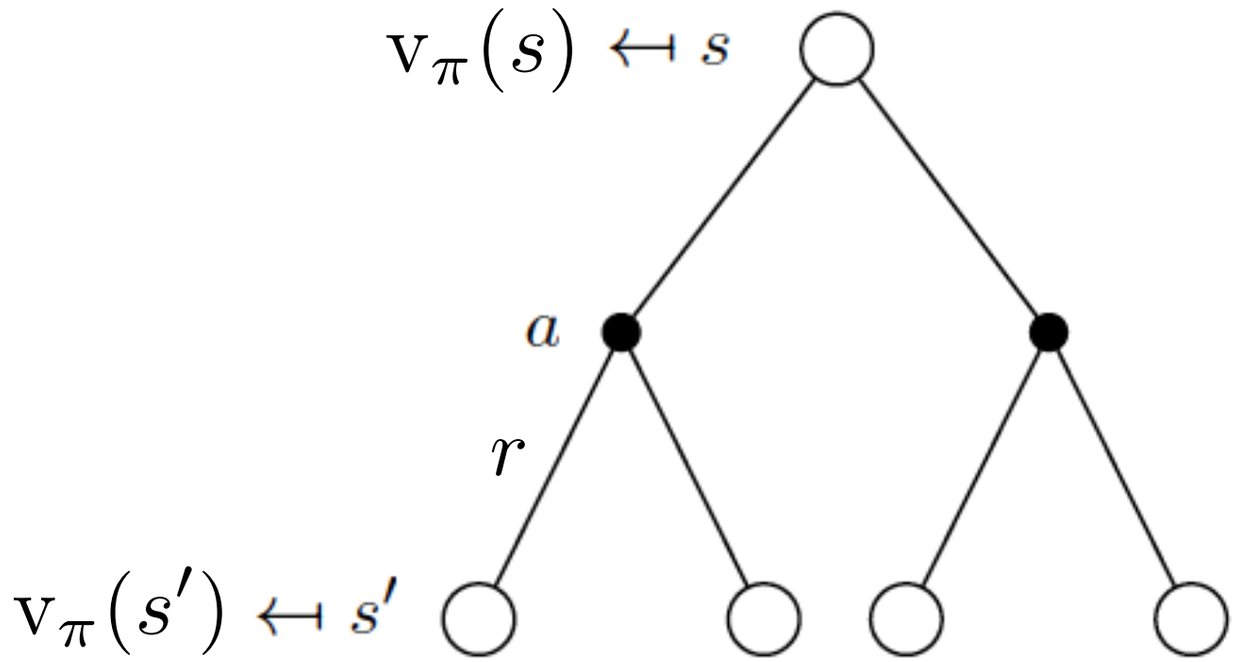
So:
$$
\begin{aligned}
v_\pi(s) &= E_\pi \left\{ G_t \,\middle|\, S_t = s \right\} \\
&= E_\pi \left\{ R_{t+1} + \gamma v_\pi \left(S_{t+1} \right) \middle|\, S_t = s \right\}
\end{aligned}
$$

Or, without the expectation operator:

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

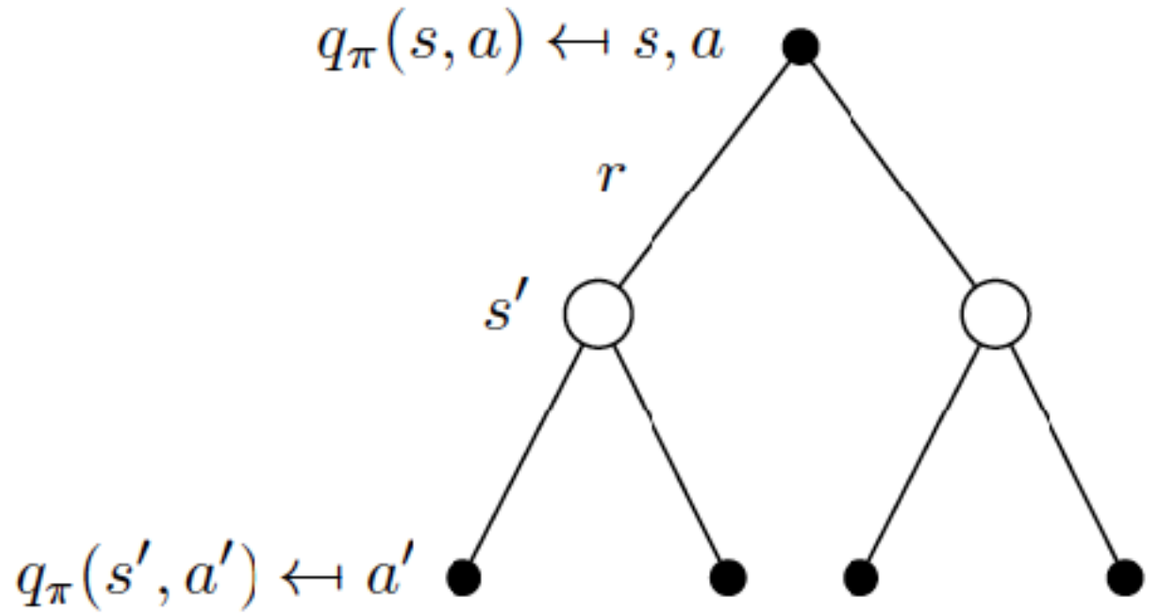
This is a set of linear equations one for each state.
The value function for π is its unique solution.

Bellman Expectation Equation



$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_\pi(s') \Big]$$

Bellman Expectation Equation

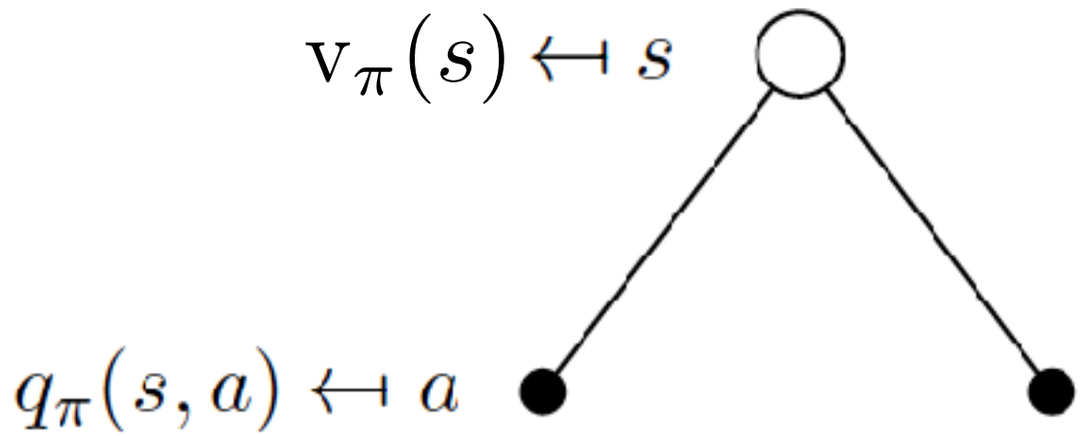


$$\mathrm{q}_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

$$\mathrm{q}_\pi(s,a) = \sum_{r,s'} \mathrm{p}(r,s' \,|\, s,a) \left(r + \gamma \sum_{a'} \pi(a' \,|\, s') q_\pi(s',a') \right)$$

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

State and State/Action Value Functions

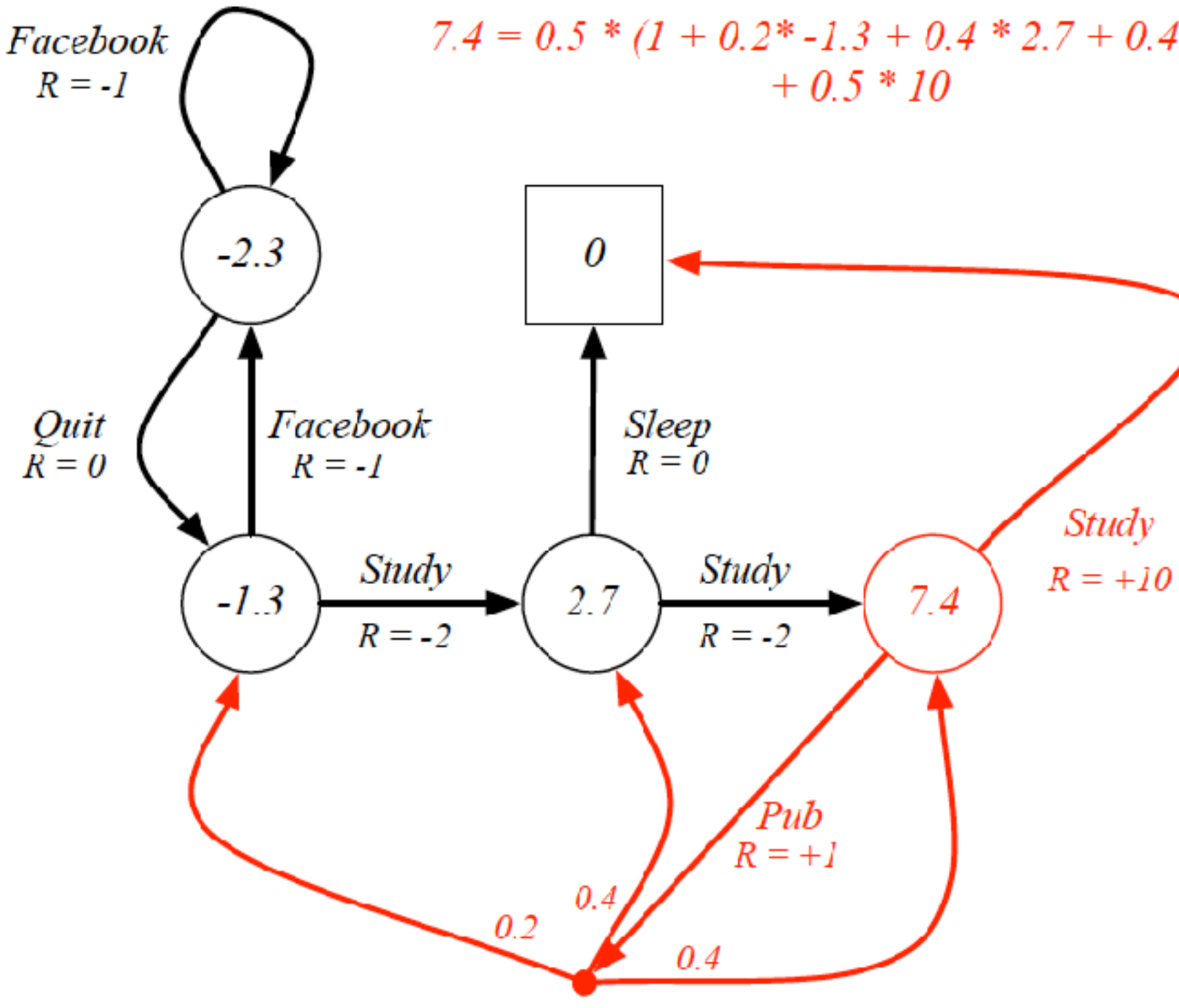


$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

Value Function for the Student MDP

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_\pi(s') \Big]$$

*7.4 = 0.5 * (1 + 0.2* -1.3 + 0.4 * 2.7 + 0.4 * 7.4)*
*+ 0.5 * 10*



Optimal Value Functions

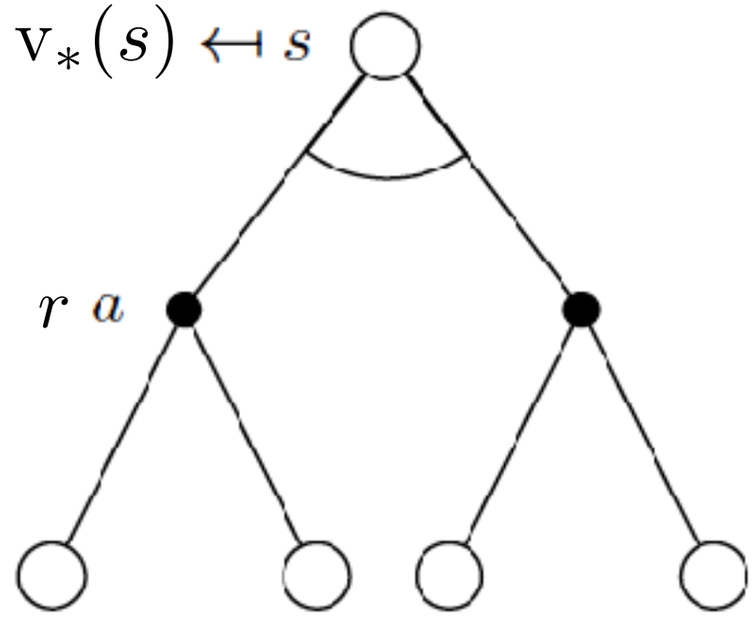
Definition: The *optimal state-value function* $\mathrm{v}_*(s)$ is the maximum value function over all policies

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

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



$$\mathrm{v}_*(s) = \max_a \left(\sum_{s',r} \mathrm{p}(s', r \,|\, s, a)(r + \gamma v_*(s')) \right)$$

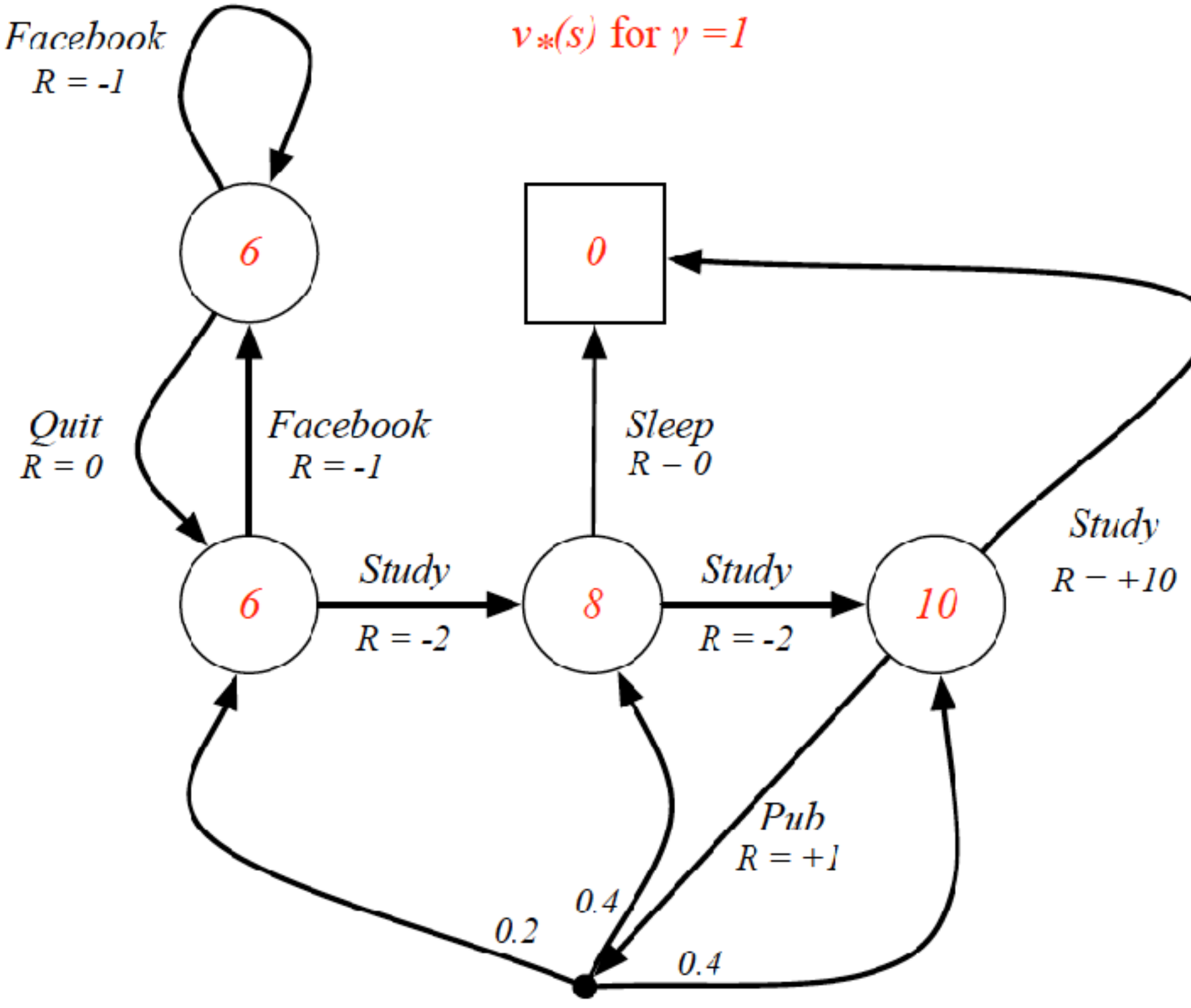
If deterministic rewards: $$\mathrm{v}_*(s) \leftarrow \max_{a \in \mathcal{A}} r(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s'|s, a)\mathrm{v}_*(s')$$

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. $\mathrm{v}_*(s)$

$$\pi_*(a \mid s) = \begin{cases} 1, & \text{if } a = \operatorname{argmax}_a \left(\sum_{s',r} \mathrm{p}(s', r \mid s, a)(r + \gamma v_*(s')) \right) \\ 0, & \text{otherwise} \end{cases}$$

Optimal Value Function for the Student MDP



Optimal action-value functions

Given q_*, the agent does not even have to do a one-step-ahead search:

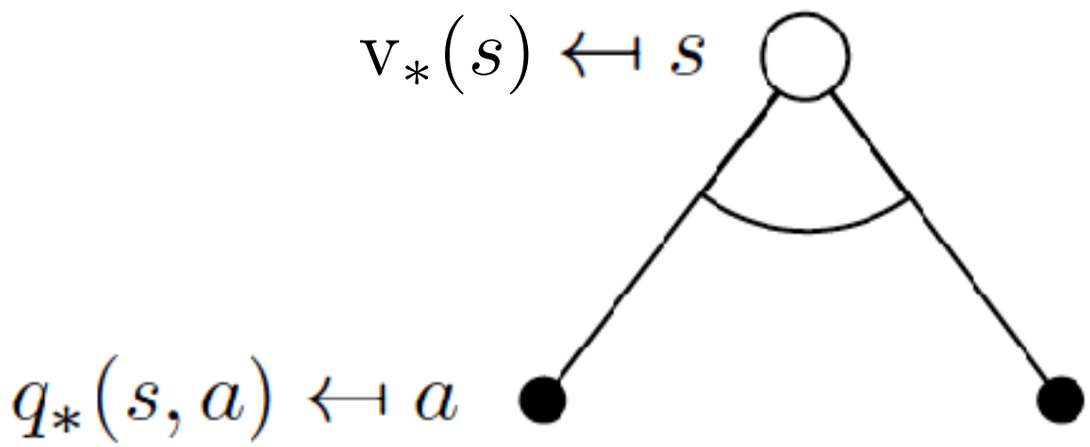
An optimal policy can be found by maximizing over $q_*(s, a)$

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

- There is always a deterministic optimal policy for any MDP
- If we know $q_*(s, a)$ we immediately have the optimal policy, **we do not need the dynamics!**

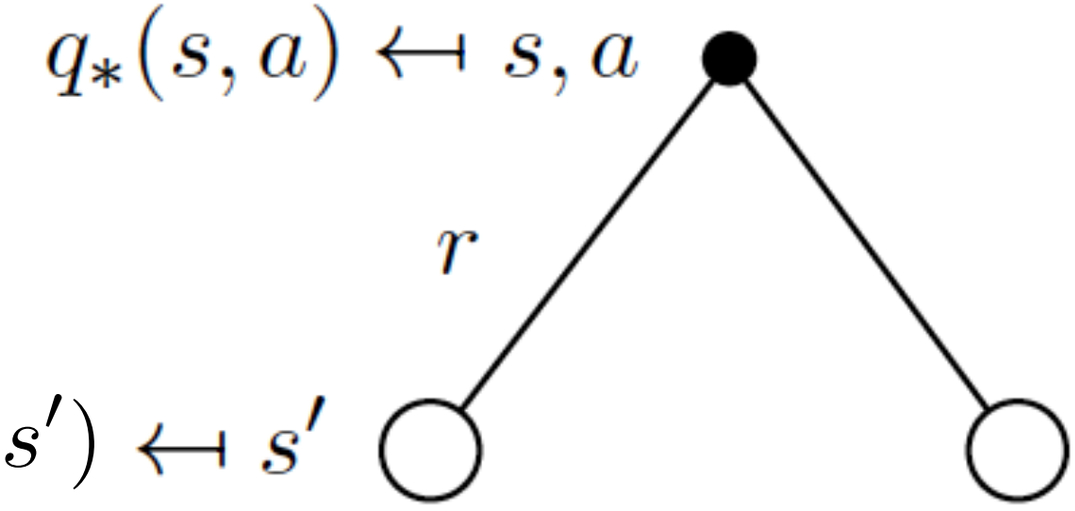
Relating Optimal State and Action Value Functions

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



$$\mathrm{q}_*(s,a) = \sum_{s',r} \mathrm{p}(s', r \,|\, s, a)(r + \gamma v_*(s'))$$

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

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;
- 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

Approximation and Reinforcement Learning

- RL methods: Approximating Bellman optimality equations

- 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

- Finite and small Markov Decision Processes with Known Dynamics
- Value functions and Optimal Value functions
- Bellman Equations

Next Lecture

- Countably infinite state and/or action spaces
- Continuous state and/or action spaces