Reinforcement Learning

CS 59300: RL1

September 4, 2025

Joseph Campbell

Department of Computer Science

PURDUE UNIVERSITY

Today's lecture

1. Evaluating Policy Quality

2. Known Models: Planning with Dynamic Programming

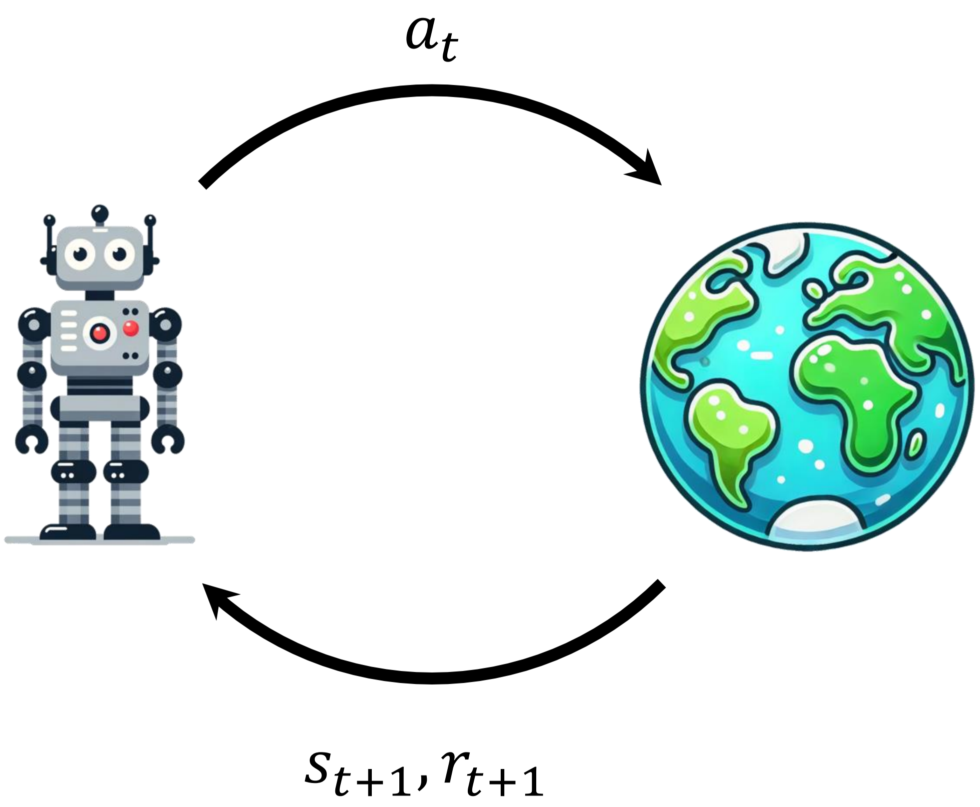
3. Unknown Models: Monte Carlo Learning

Some content inspired by David Silver's UCL RL course and Katerina Fragkiadaki's CMU 10-403

Recap

The agent and environment operate at discrete timesteps $t = 0,1,2, \ldots$

- The agent observes state s_t at time t
- The agent takes action a_t
- The agent gets the resulting reward r_{t+1} and the subsequent state s_{t+1}

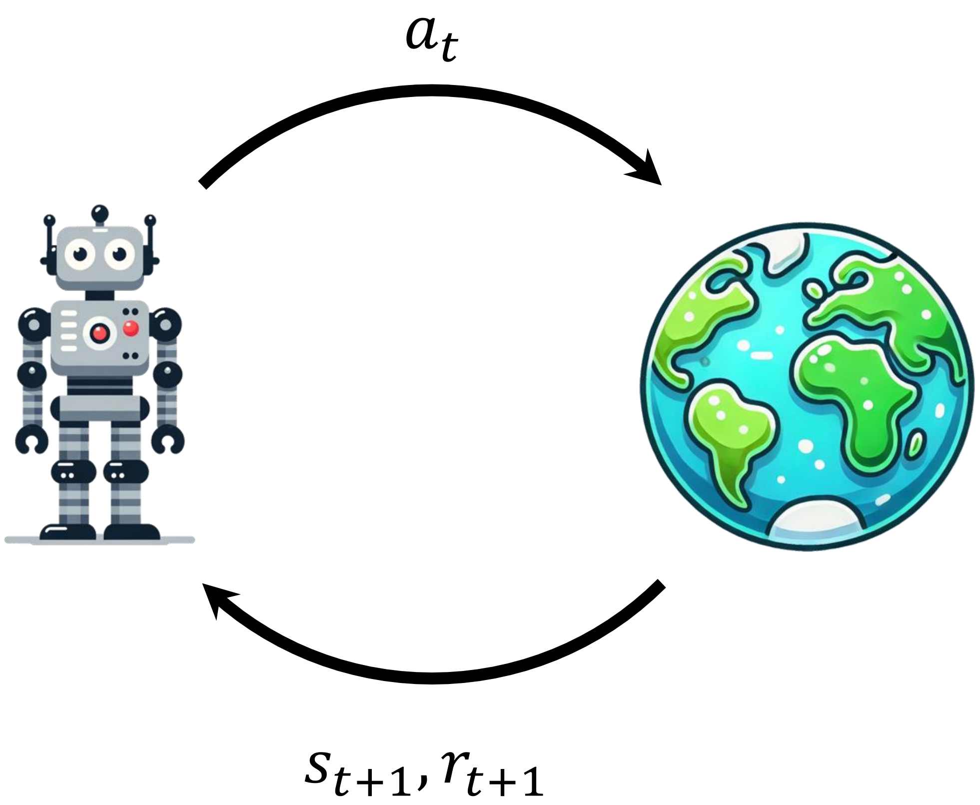


Recap

Action a_t is chosen by sampling actions from a probability distribution

$$a_t \sim \pi(a|s)$$

The probability distribution π is referred to as a policy.



Recap: Markov Decision Process

We formulate sequential decision-making problems as MDPs

Definition: a tuple consisting of $(\mathcal{S}, \mathcal{A}, R, T, \gamma)$:

- $\mathcal{S}$ is the set of states in our environment
- $\mathcal{A}$ is a set of actions that can be taken by an agent
- R is the reward function. $\mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$
- T is the state transition probability. $\mathcal{S} \times \mathcal{S} \times \mathcal{A} \mapsto [0,1]$
- γ is a discount factor. $\gamma \in [0,1]$

Value functions

Definition: the action-value function of an MDP is the expected return starting from state s, taking action a, and following policy π for all subsequent states

$$Q_\pi(s, a) = \mathbb{E}[G_t | s_t = s, a_t = a]$$

Definition: the state-value function of an MDP is the expected return starting from state s and following policy π

$$V_\pi(s) = \mathbb{E}[G_t | s_t = s]$$

Recap: Finding an optimal policy

If we know the optimal action-value function $Q^*(s, a)$…

$$\pi^*(a|s) = \begin{cases} 1, & \text{if } a = \operatorname{argmax}_{a \in \mathcal{A}} Q^*(s, a) \\ 0, & \text{otherwise} \end{cases}$$

…we immediately have the optimal policy. We simply follow it.

Recap: Finding an optimal policy

If we know the optimal state-value function $V^*(s)$...

$$\pi^*(a|s) = \begin{cases} 1, & \text{if } a = \text{argmax}_{a \in \mathcal{A}} [\sum_{s',r} p(s',r|s,a)(r + \gamma V^*(s'))] \\ 0, & \text{otherwise} \end{cases}$$

...we need access to the state transition function (defined in the MDP earlier as T) to do one-step ahead lookup

- Take the action which leads us to the state with highest value

Evaluating Policy Quality

The reward hypothesis

Let’s take a step back: **why do we care about rewards and values?**

The reward hypothesis

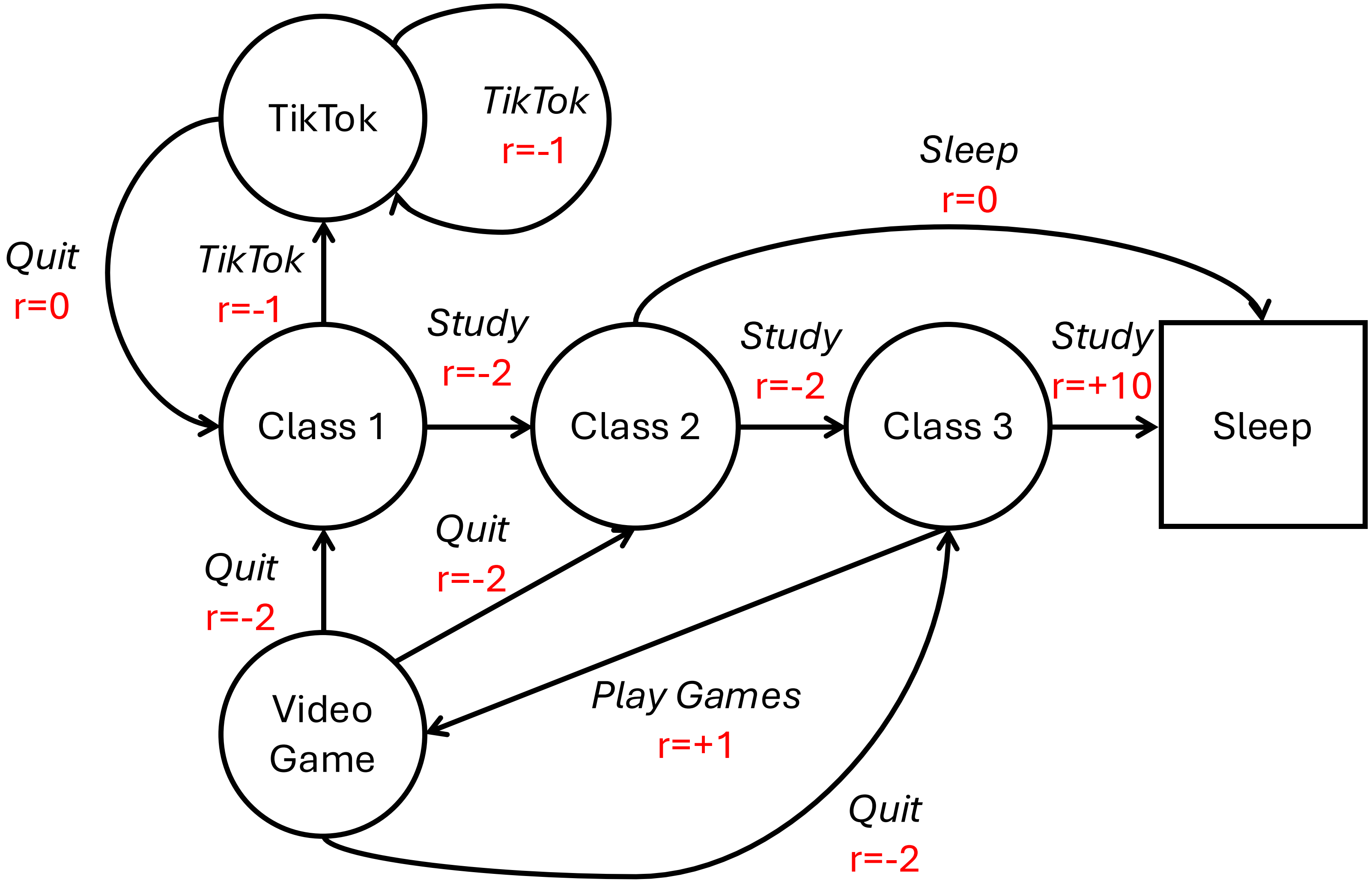
Let's take a step back: **why do we care about rewards and values?**

"*That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).*"

- Sutton and Barto, Chapter 3.2

Do you agree? Why or why not?

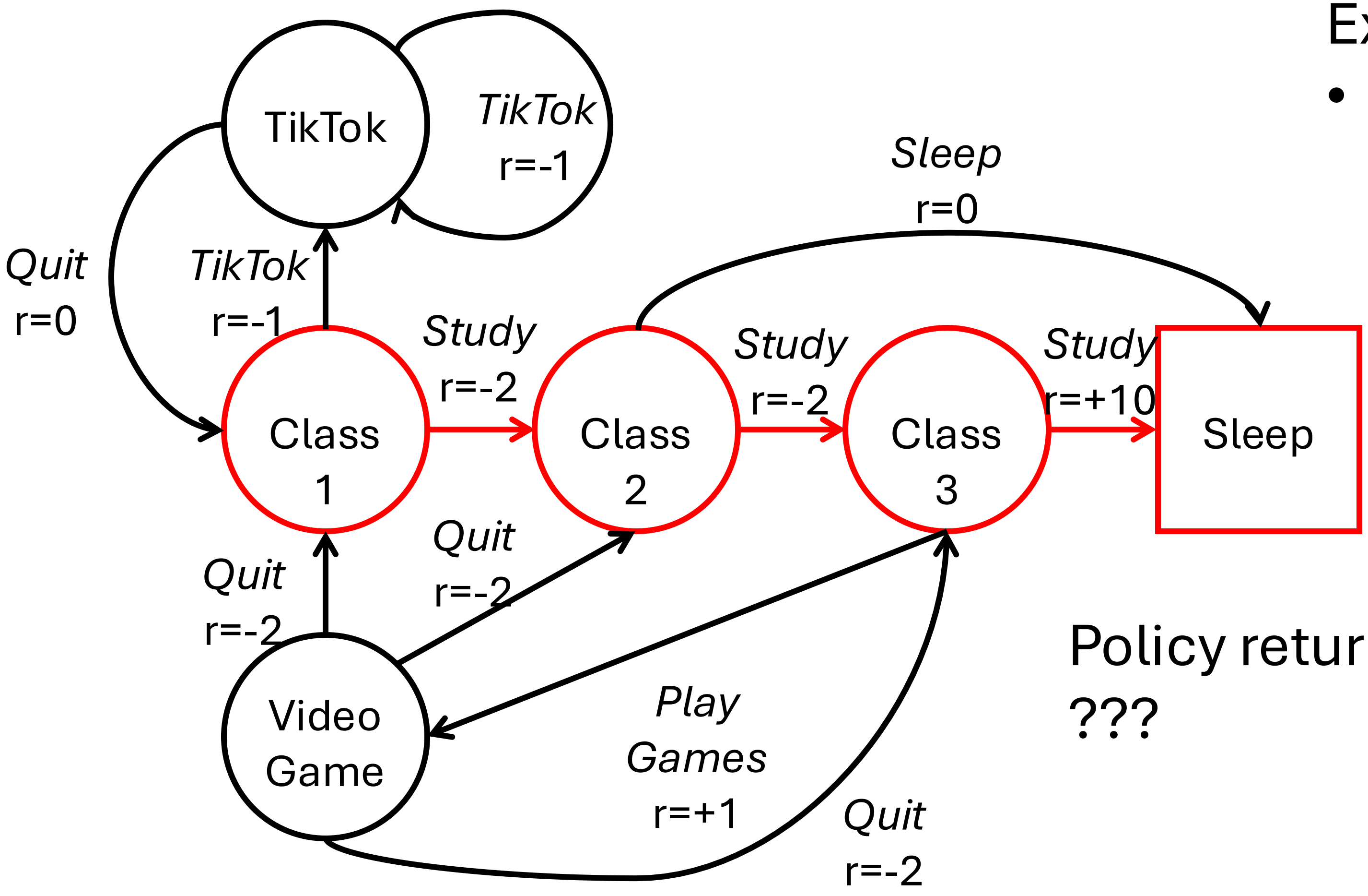
Example: student life as an MDP



Recall: Returns for a policy

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots$$

Example: student life as an MDP

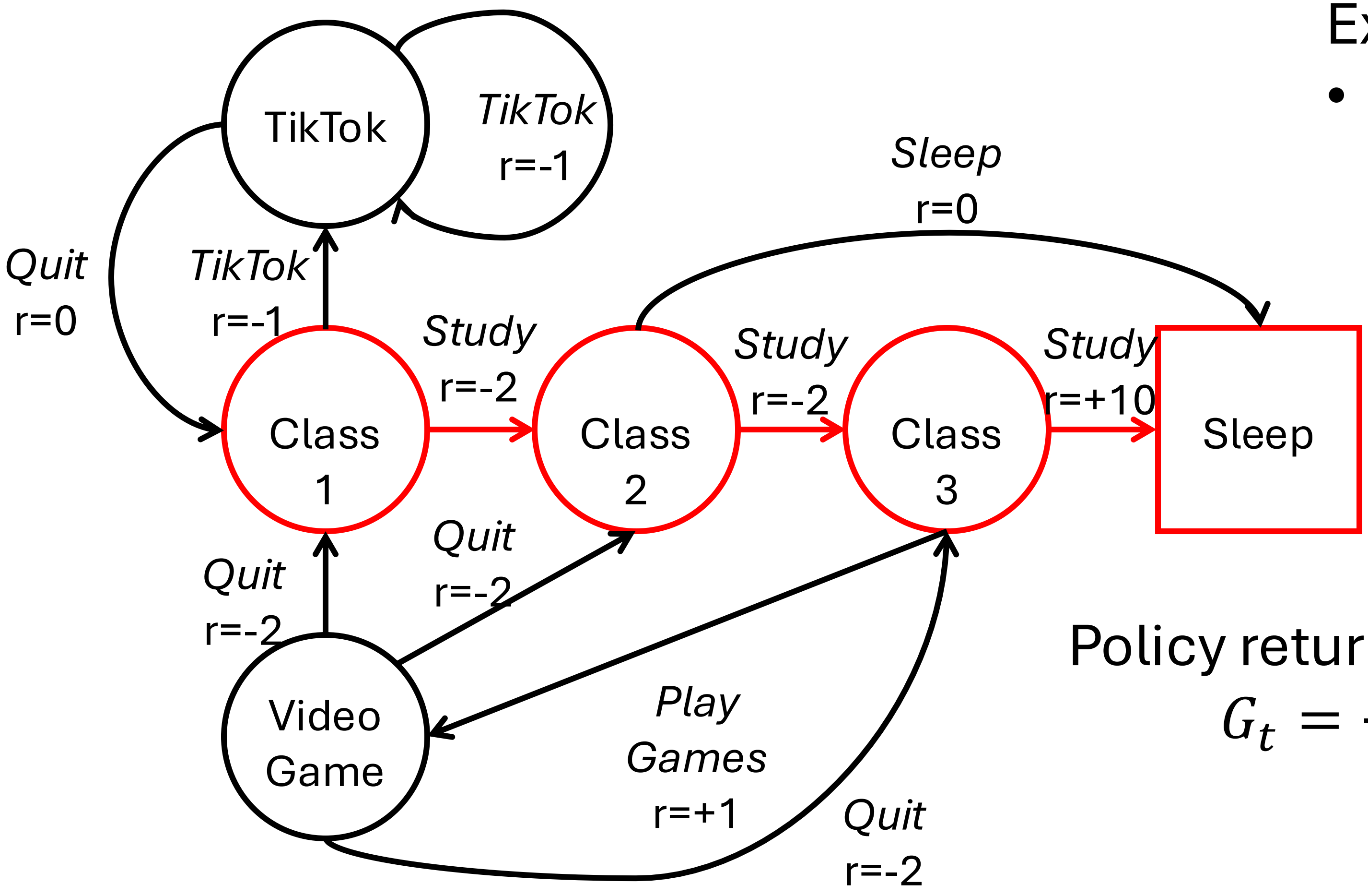


Example policy rollout:

- Class 1, Class 2, Class 3, Sleep

Policy return:
???

Example: student life as an MDP



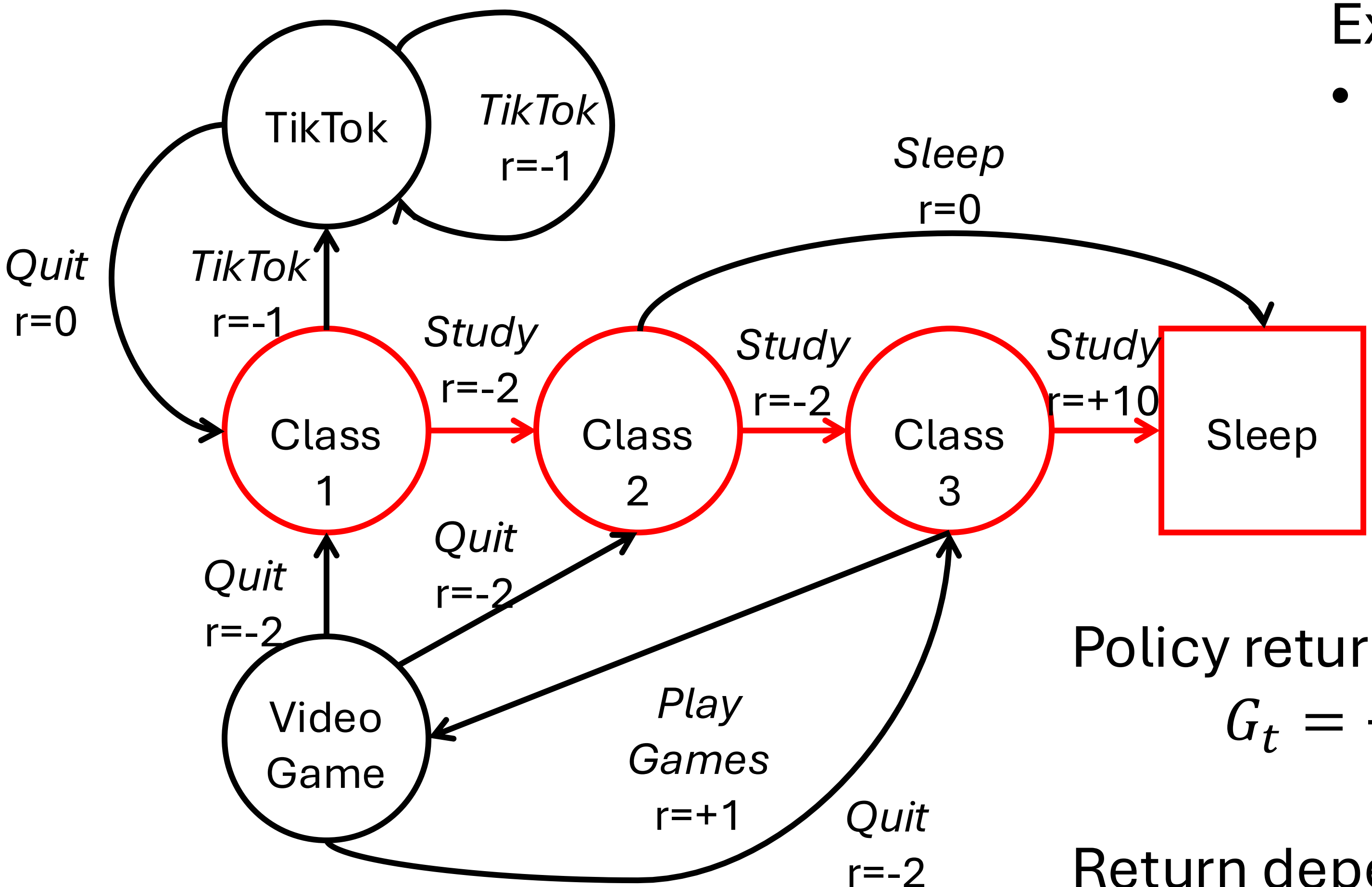
Example policy rollout:

- Class 1, Class 2, Class 3, Sleep

Policy return:

$$G_t = -2 + \gamma(-2) + \gamma^2(+10)$$

Example: student life as an MDP



Example policy rollout:

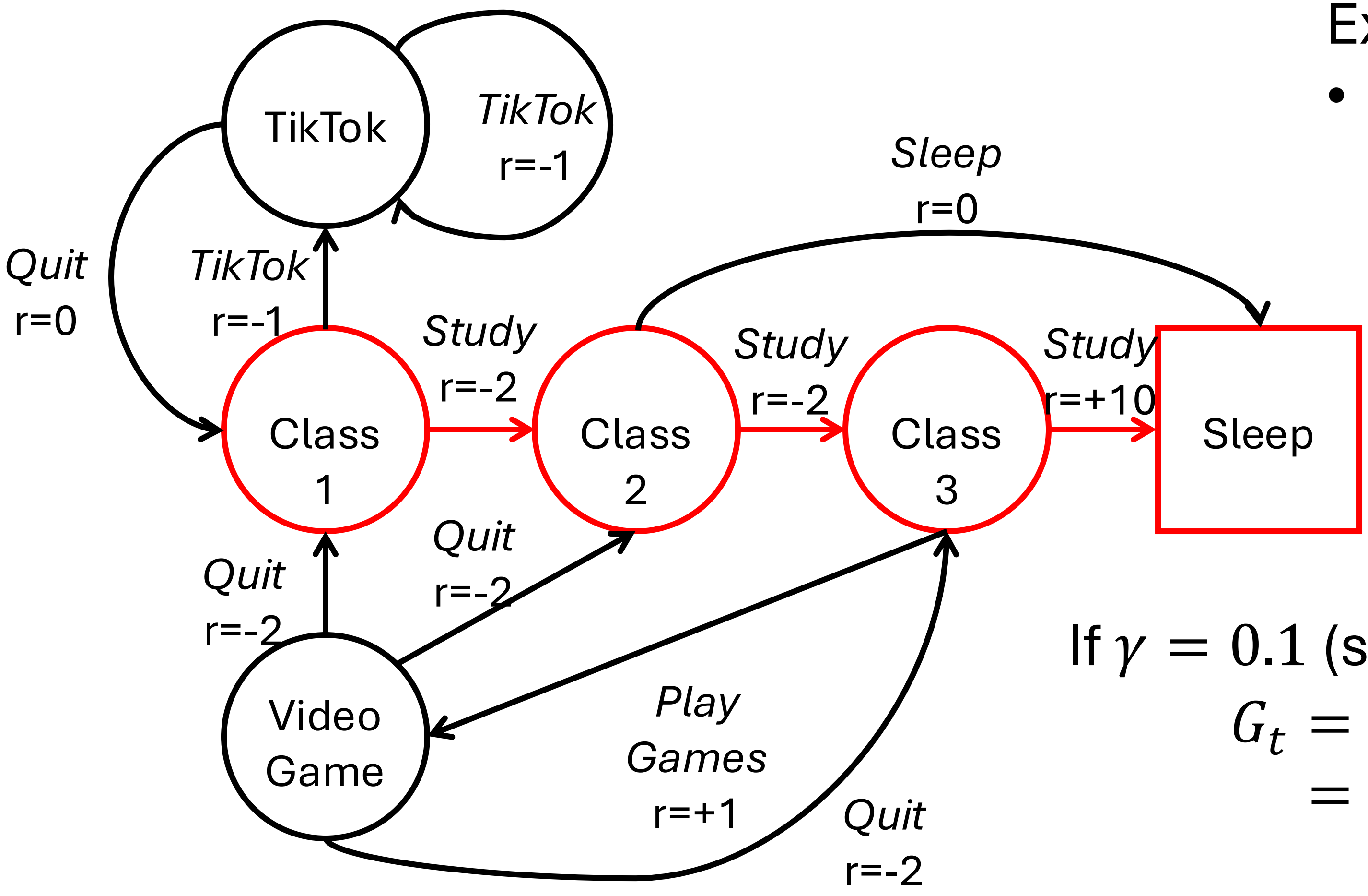
- Class 1, Class 2, Class 3, Sleep

Policy return:

$$G_t = -2 + \gamma(-2) + \gamma^2(+10)$$

Return depends on γ!

Example: student life as an MDP



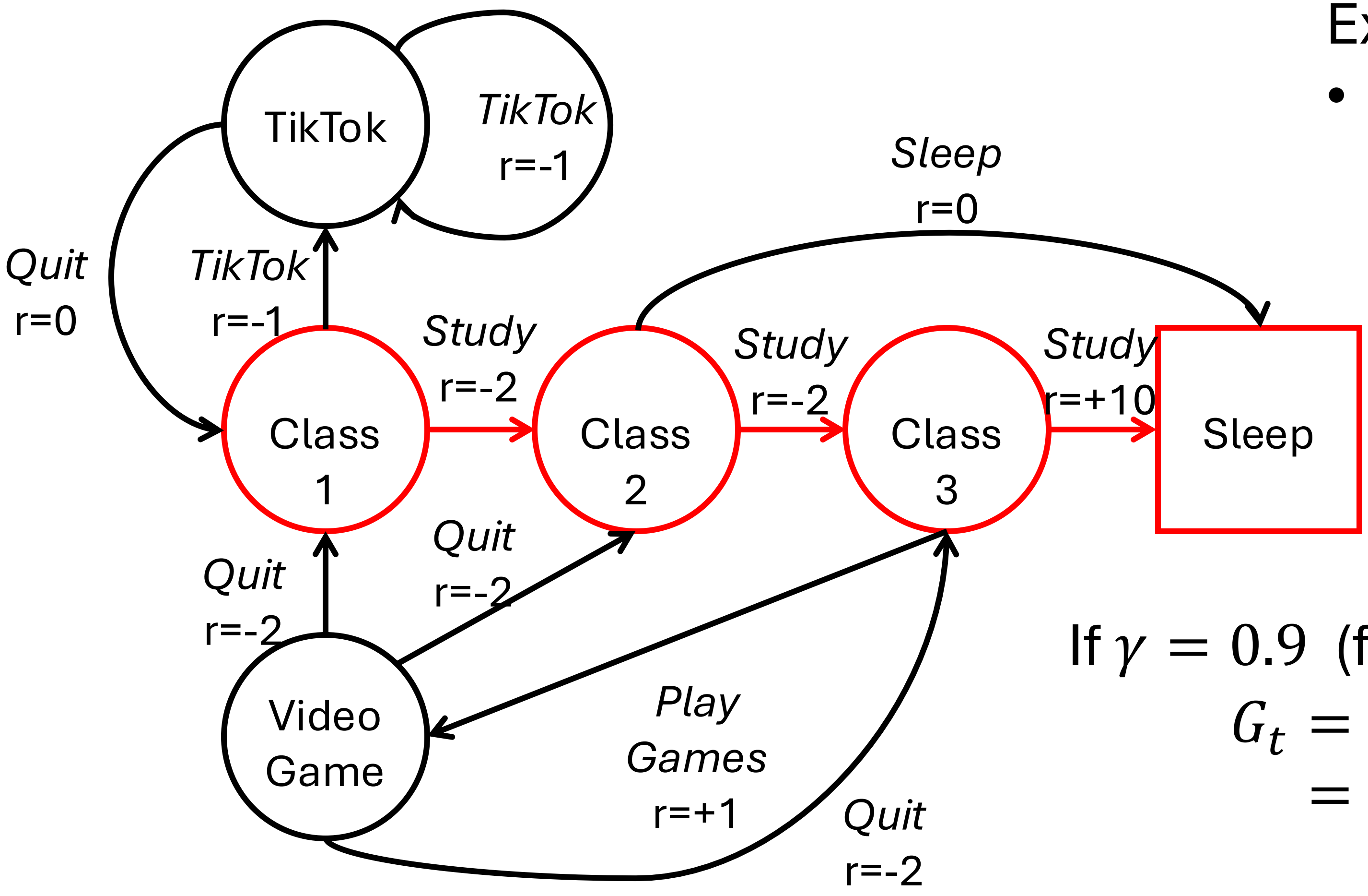
Example policy rollout:

- Class 1, Class 2, Class 3, Sleep

If $\gamma = 0.1$ (short-sighted):

$$G_t = -2 + 0.1(-2) + 0.1^2(+10)$$
$$= \mathbf{-2.1}$$

Example: student life as an MDP



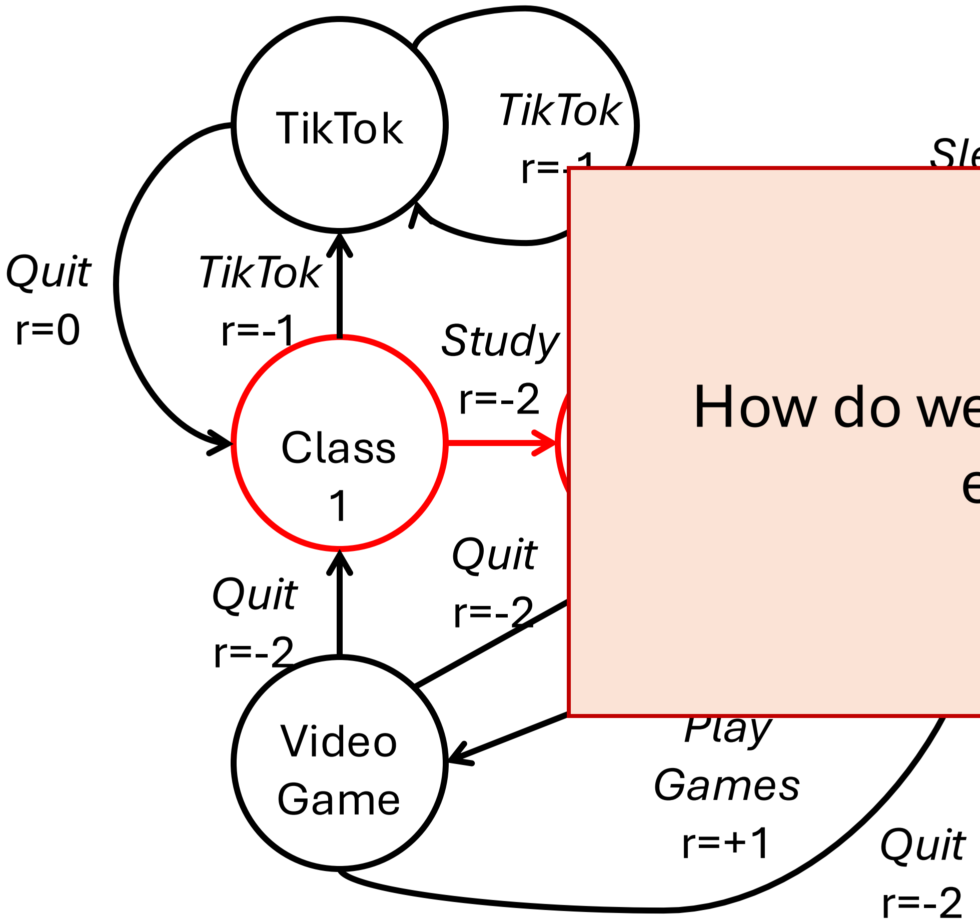
Example policy rollout:

- Class 1, Class 2, Class 3, Sleep

If $\gamma = 0.9$ (far-sighted):

$$G_t = -2 + 0.9(-2) + 0.9^2(+10)$$
$$= \mathbf{+4.3}$$

Example: student life as an MDP



Example policy rollout:

- Class 1, Class 2, Class 3, Sleep

How do we choose actions so as to maximize the expected return for a given γ?

$$G_t = -2 + 0.9(-2) + 0.9^2(-2) + 0.9^3(+10) = +1.87$$

Recursive form of policy returns

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots$$

Recursive form of policy returns

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots$$

$$= \ r_{t+1} + \gamma (r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \cdots)$$

Recursive form of policy returns

$$
\begin{aligned}
G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots \\
&= r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \cdots) \\
&= r_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

Recursive form of policy returns

$$
\begin{aligned}
G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots \\
&= r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \cdots) \\
&= r_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

This relationship allows us to decompose value functions

Bellman expectation equation

We can decompose value functions into two parts:

- The immediate reward
- The expected future returns

$$\mathbb{E}[G_t | s_t = s] = \mathbb{E}[r_{t+1} + \gamma G_{t+1} | s_t = s]$$

Bellman expectation equation

We can decompose value functions into two parts:

- The immediate reward
- The expected future returns

$$\mathbb{E}[G_t | s_t = s] = \mathbb{E}[r_{t+1} + \gamma G_{t+1} | s_t = s]$$

State-value: $V_\pi(s) = \mathbb{E}[r_{t+1} + \gamma V_\pi(s_{t+1}) | s_t = s]$

Bellman expectation equation

We can decompose value functions into two parts:

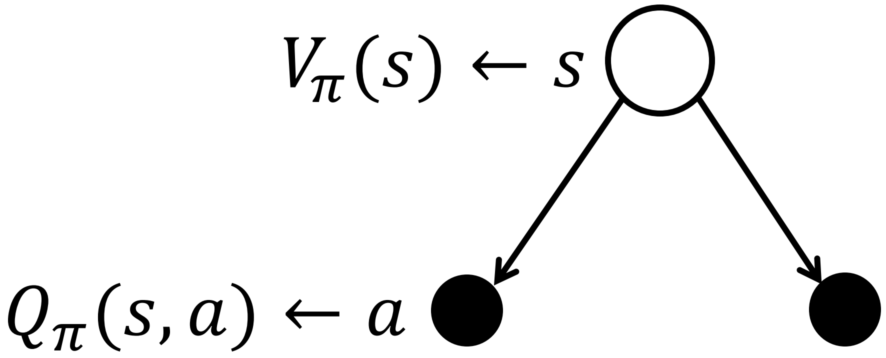
- The immediate reward
- The expected future returns

$$\mathbb{E}[G_t | s_t = s] = \mathbb{E}[r_{t+1} + \gamma G_{t+1} | s_t = s]$$

State-value: $V_\pi(s) = \mathbb{E}[r_{t+1} + \gamma V_\pi(s_{t+1}) | s_t = s]$

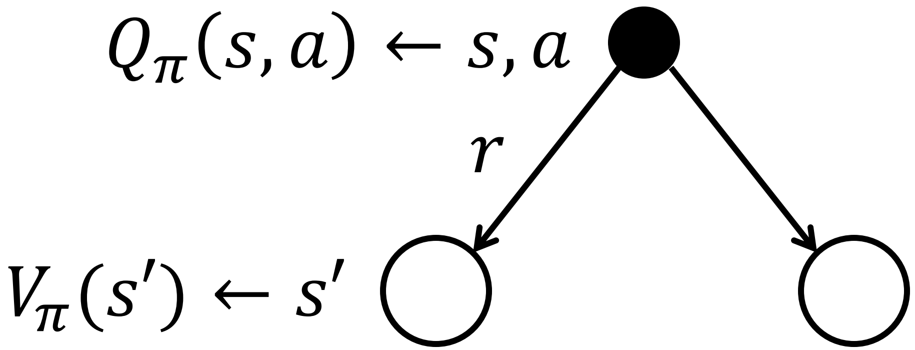
Action-value: $Q_\pi(s, a) = \mathbb{E}[r_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1}) | s_t = s, a_t = a]$

State-value and action-values are related



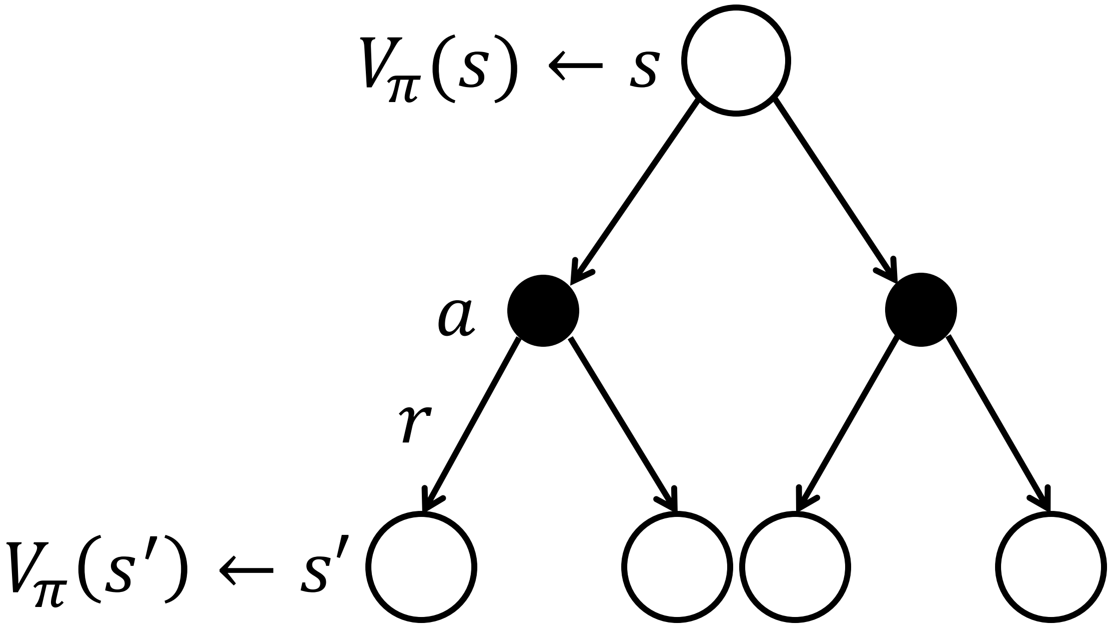
$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s)\, Q_\pi(s,a)$$

State-value and action-values are related



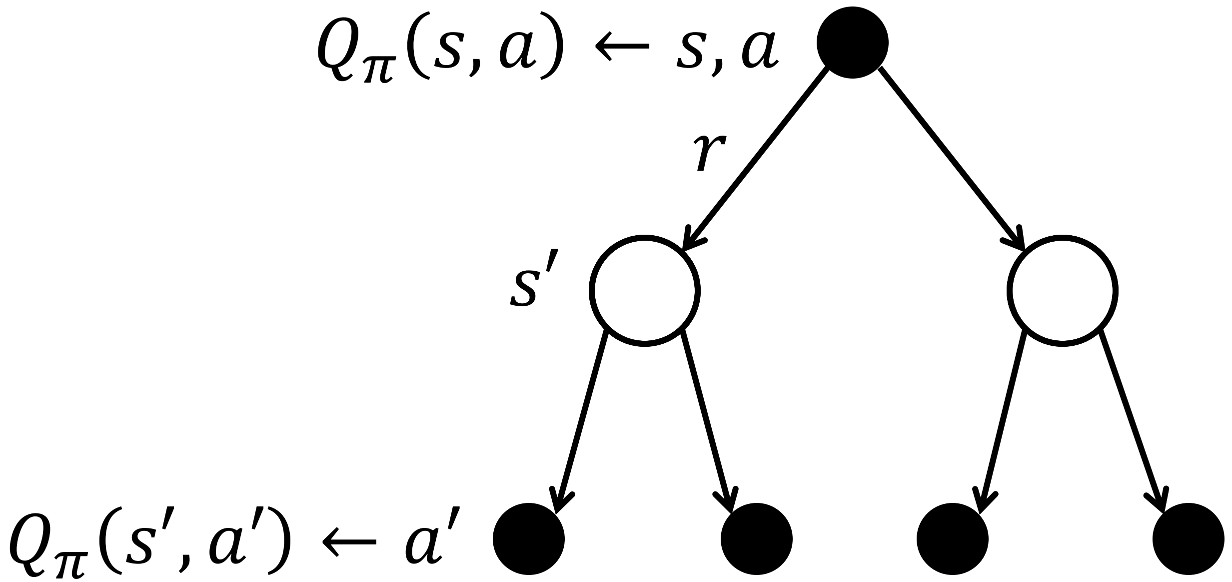
$$Q_\pi(s,a) = r + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)\, V_\pi(s')$$

Bellman expectation equation for state-value



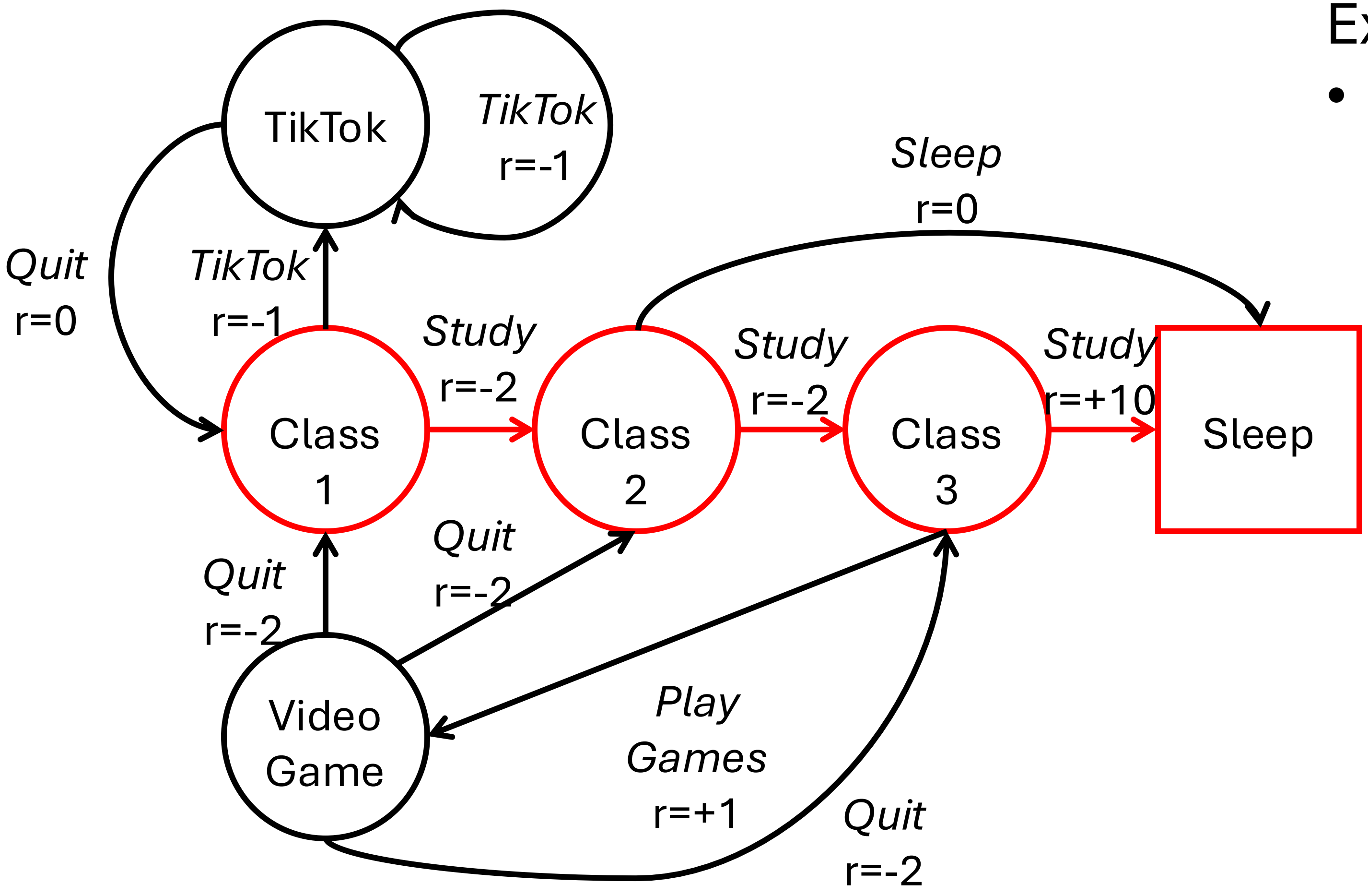
$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)\, V_\pi(s')\right)$$

Bellman expectation equation for action-value



$$Q_\pi(s,a) = r + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) \sum_{a' \in \mathcal{A}} \pi(a'|s')\, Q_\pi(s',a')$$

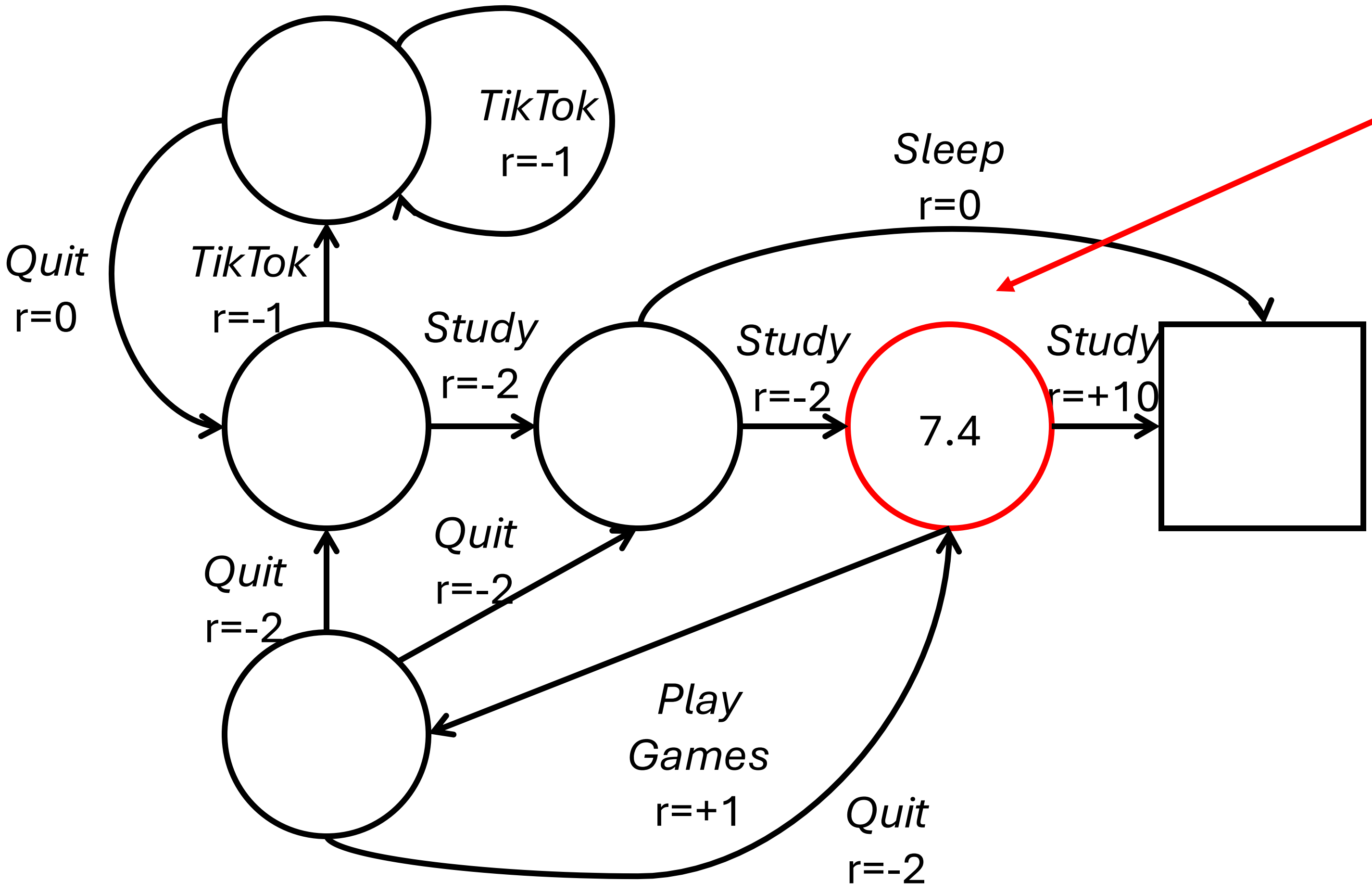
Evaluating a policy with Bellman expectation



Example policy rollout:

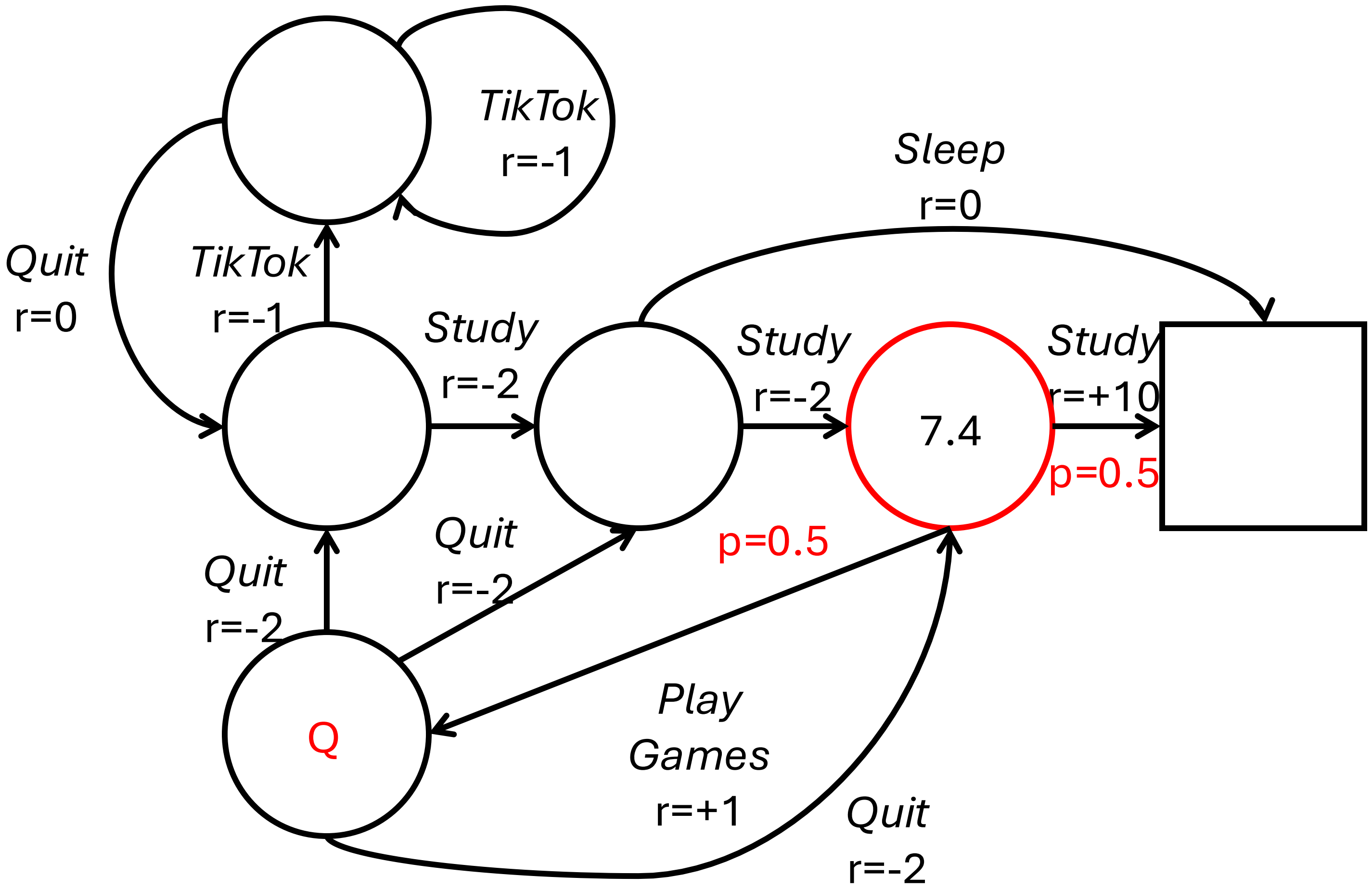
- Class 1, Class 2, Class 3, Sleep

Evaluating a policy with Bellman expectation



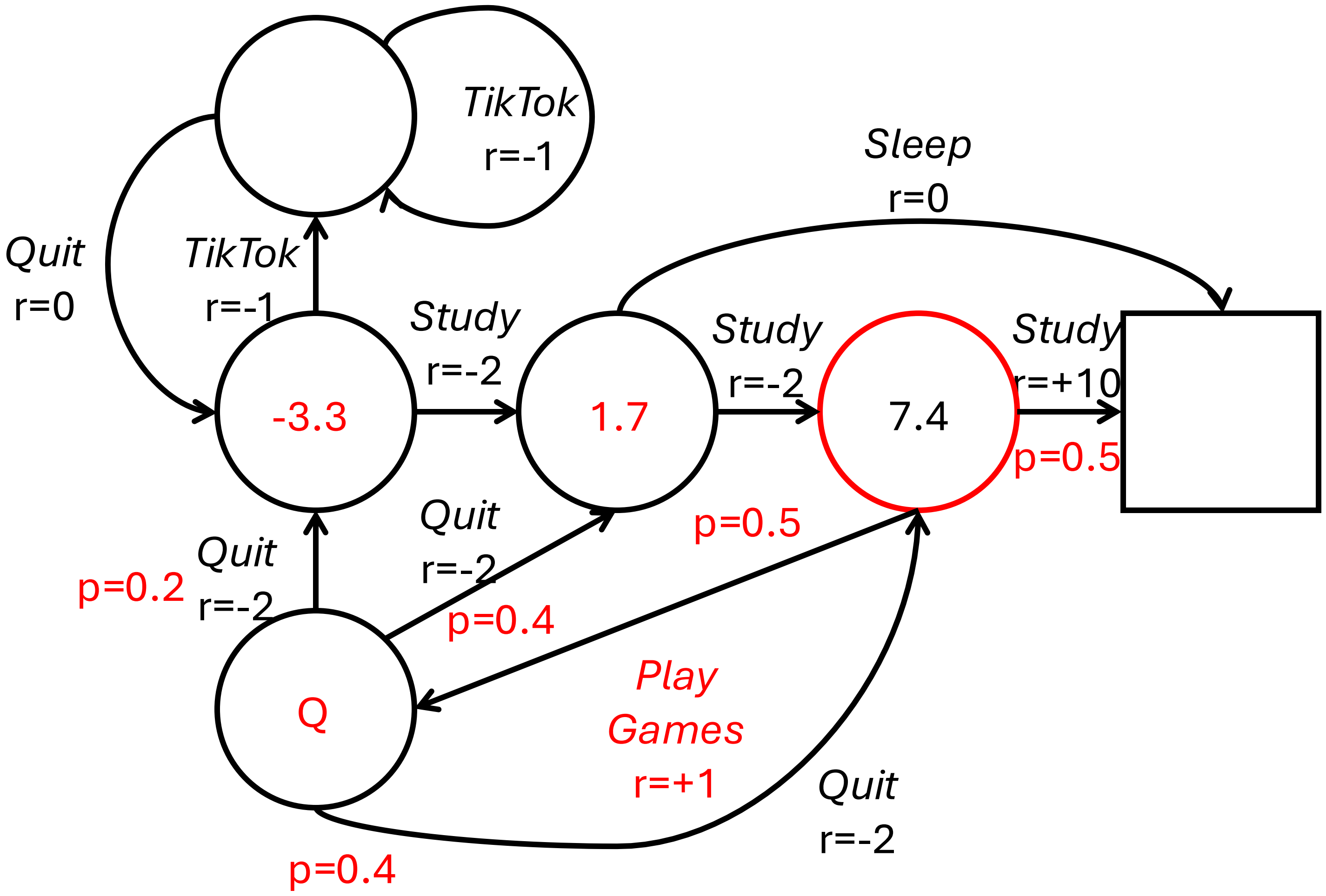
Let's see how we compute V_π for the Class 3 state with $\gamma = 1$

Evaluating a policy with Bellman expectation



$$V_\pi = 0.5 * 10 \\ + \ 0.5 \ * Q_\pi(\text{VG, Play VG})$$

Evaluating a policy with Bellman expectation



$$V_\pi = 0.5 * 10 + 0.5 * Q_\pi(\text{VG, Play VG}) = 7.4$$

$$Q_\pi(\text{VG, Play VG}) = (1 + 0.2 * -3.3 + 0.4 * 1.7 + 0.4 * 7.4)$$

Closed-form solution

The Bellman expectation equation can be represented as a system of linear equations which results in a closed-form solution:

$$V_\pi = R_\pi + \gamma T_\pi V_\pi$$

$$V_\pi = (I - \gamma T_\pi)^{-1} R_\pi$$

R_π is an $|\mathcal{S}|$-dimensional vector where j-th entry = $\mathbb{E}[r|s_j, a = \pi(s_j)]$

V_π is an $|\mathcal{S}|$-dimensional vector where j-th entry = $V_\pi(s_j)$

T_π is an $|\mathcal{S}| \times |\mathcal{S}|$-dimensional matrix where (j,k) = $p(s_k|s_j, a = \pi(s_j))$

How do we know if a value function is optimal?

The *optimal state-value* function is the max over all policies

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

And similarly for the *optimal action-value* function...

$$Q^*(s, a) = \max_{\pi} Q_{\pi}(s, a)$$

This represents the best possible performance for a given MDP

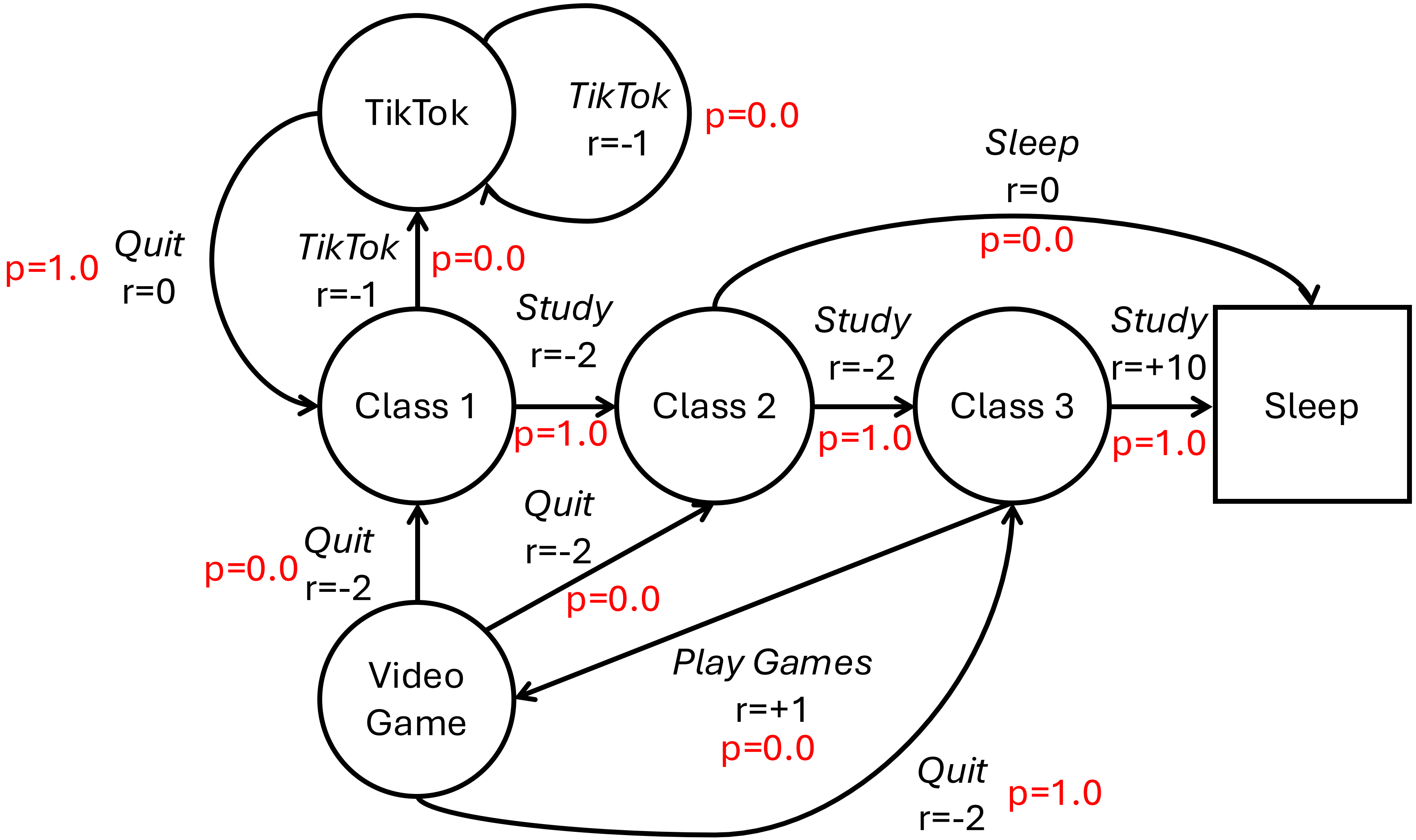
Bellman optimality equations

The value of a state under an optimal policy must equal the expected return for the best action from that state:

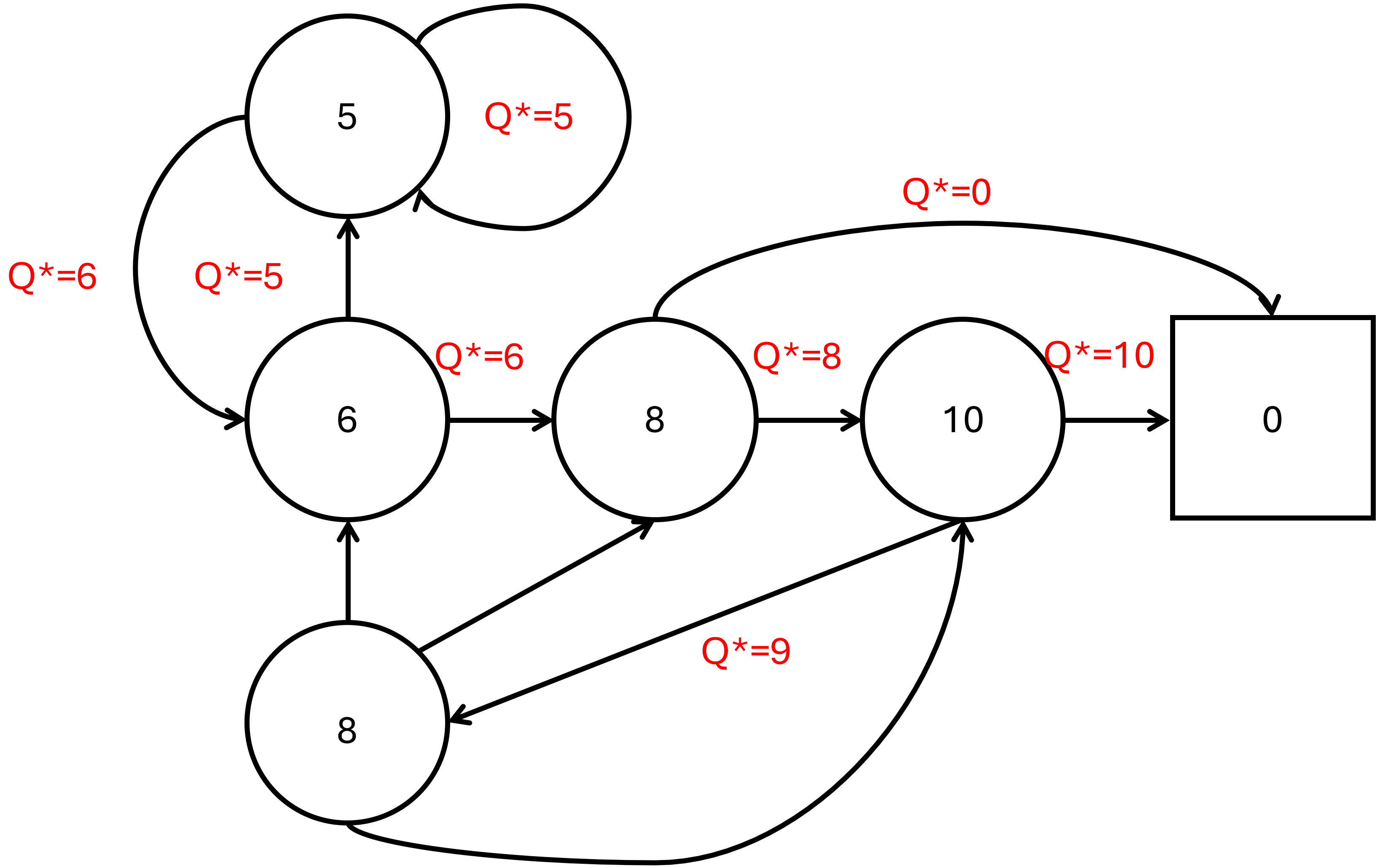
$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s',r} p(s', r | s, a)(r + \gamma V^*(s'))$$

$$Q^*(s, a) = \sum_{s',r} p(s', r | s, a)[r + \gamma \max_{a \in \mathcal{A}} Q^*(s', a')]$$

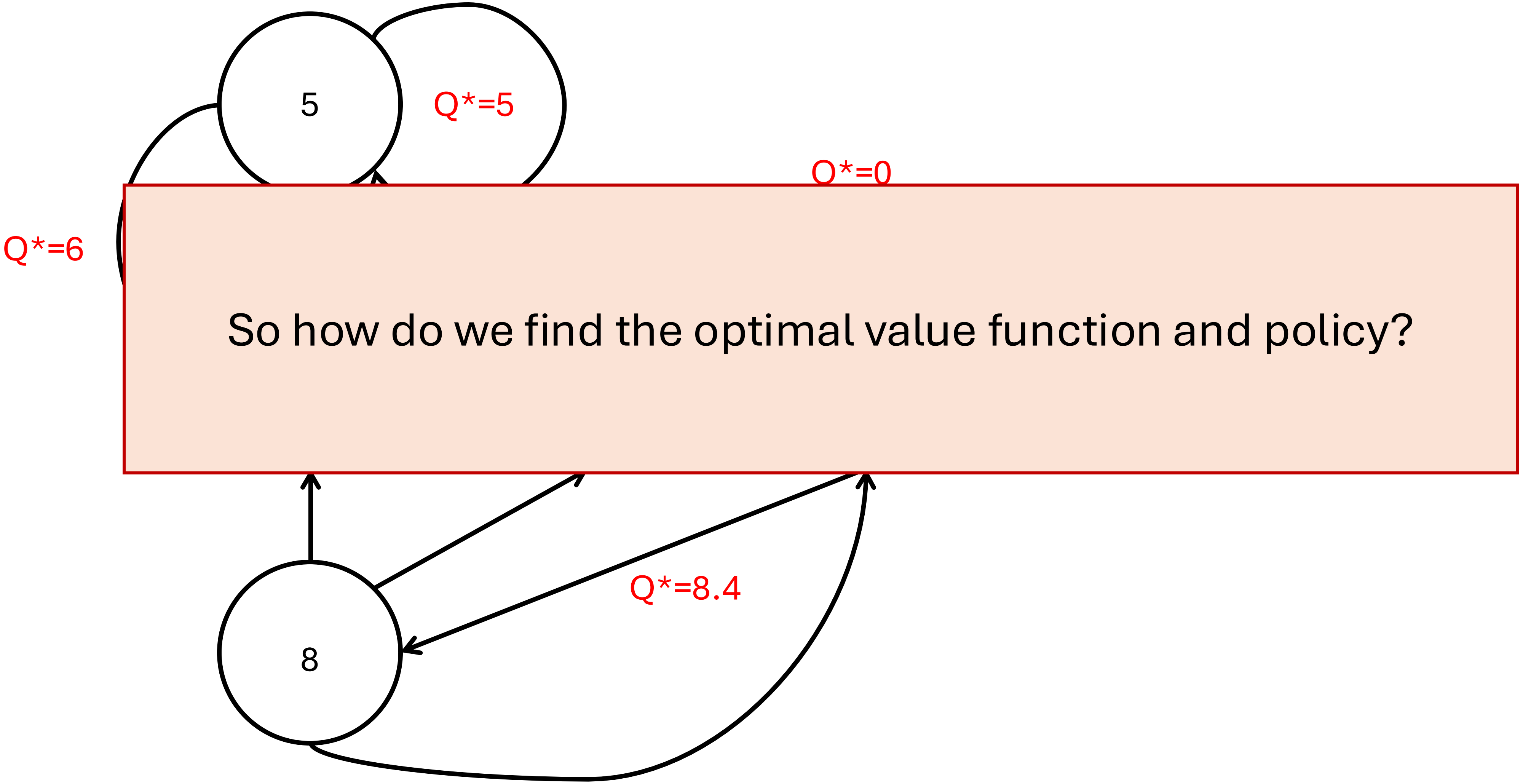
The optimal policy for our example



The optimal value function for our example



The optimal value function for our example



So how do we find the optimal value function and policy?

Known Models: Planning with Dynamic Programming

Fully known model = planning

When we have complete knowledge of the environment transition function and reward function this becomes a planning problem!

We can compute the optimal action-value $Q^*(s, a)$ and state-value $V^*(s)$ functions which allows us to "solve" the MDP

- If we have optimal action-value then we have the optimal policy
- If we have optimal state-value and the environment transition function then we have the optimal policy

No general closed-form solution

Bellman optimality equations are non-linear due to the max operator

This means we cannot find the optimal policy by solving a system of linear equations!

Instead…we use an iterative approach

Dynamic Programming

What is dynamic programming?

Dynamic Programming

An optimization method and programming paradigm where the overall problem is broken into simpler sub-problems

It consists of two steps:

1. Solve the sub-problems
2. Combine sub-problem solutions to obtain overall solution

Policy iteration: finding the optimal policy

Idea: use Dynamic Programming to find *optimal policy* for a given MDP

Two-step iterative algorithm. Given a policy π...

- Evaluate the policy

$$V_\pi(s) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \cdots | s_t = s]$$

- Improve the policy by acting greedily with respect to V_π

$$\pi' = \text{greedy}(V_\pi)$$

Policy iteration intuition

Starting with a random policy, we evaluate it to find V_π

- Key insight: this policy may not be greedy!
- It might not always choose the action that maximizes the immediate expected return based on V_π

By generating a new policy which **is** greedy with respect to V_π, we...

- Make it a little more "greedy" each update
- Monotonic improvement which provably converges to optimum

Iterative policy evaluation

Earlier we discussed a closed-form solution for evaluation

$$V_\pi = (I - \gamma T_\pi)^{-1} r_\pi$$

Unfortunately, this solution has a serious flaw. **What is it?**

Iterative policy evaluation

Earlier we discussed a closed-form solution for evaluation

$$V_\pi = (I - \gamma T_\pi)^{-1} R_\pi$$

Unfortunately, this solution has a serious flaw. **What is it?**

It is not computationally tractable!

- Requires us to invert ($\mathcal{O}(n^3)$) an $|\mathcal{S}| \times |\mathcal{S}|$-dimensional matrix
- In large state spaces, we have both compute and memory issues

Iterative policy evaluation

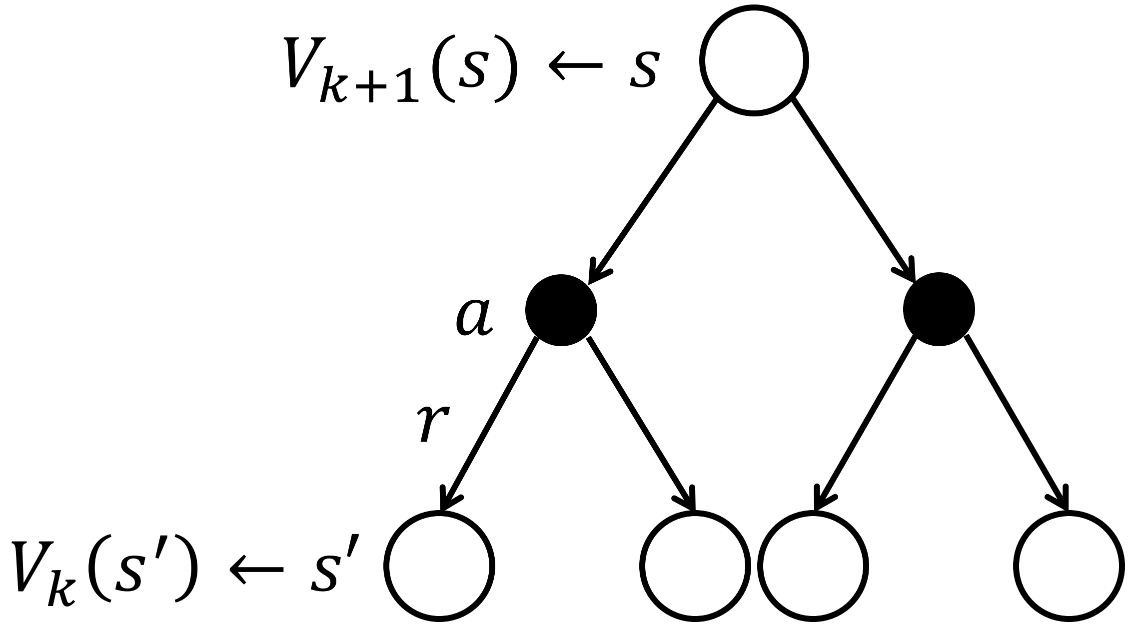
We instead iteratively apply the Bellman equations to convergence

1. Randomly initialize our value function $V_0(s)$ for all $s \in \mathcal{S}$
2. For iteration $k = 1,2, \ldots$

   Update $V_{k+1}(s)$ from $V_k(s')$ for all $s \in \mathcal{S}$

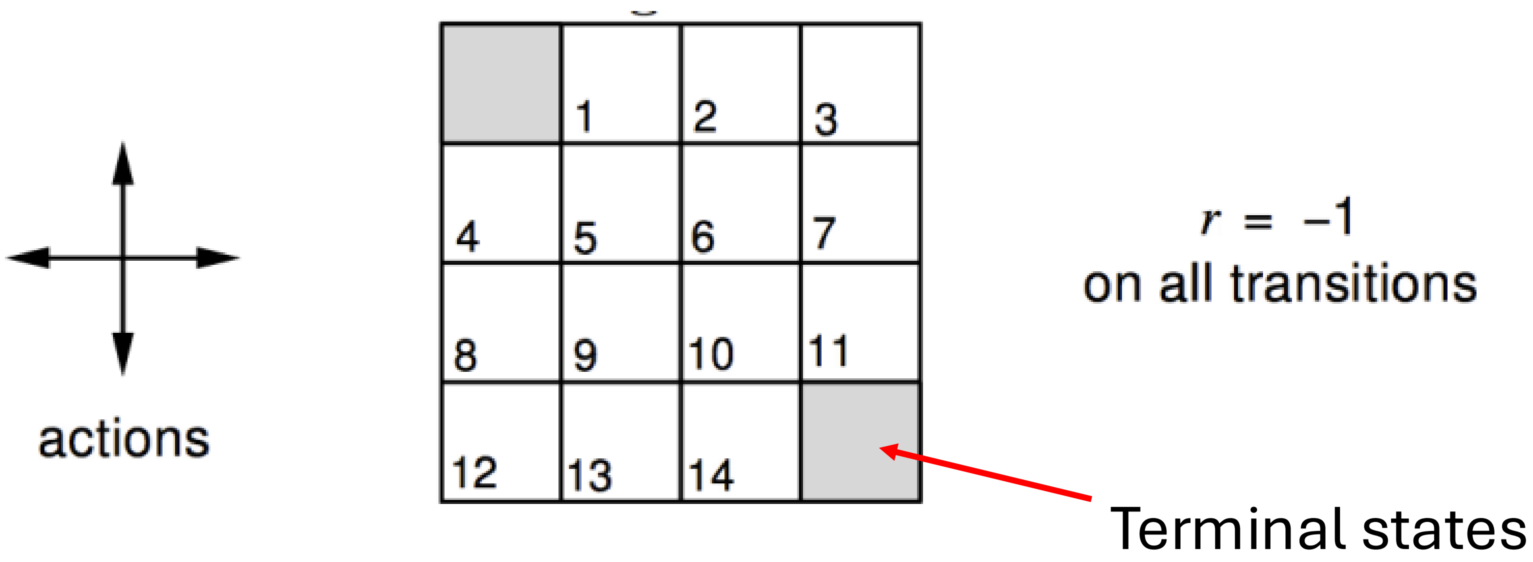
   If $\max_s |V_{k+1}(s) - V_k(s)| < \epsilon$ then stop

Iterative policy evaluation



$$V_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(r + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)\, V_k(s')\right)$$

Example: Gridworld



Agent policy is uniformly random

- 25% chance to go north, east, south, or west

Example: Gridworld

v_k for the Random Policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

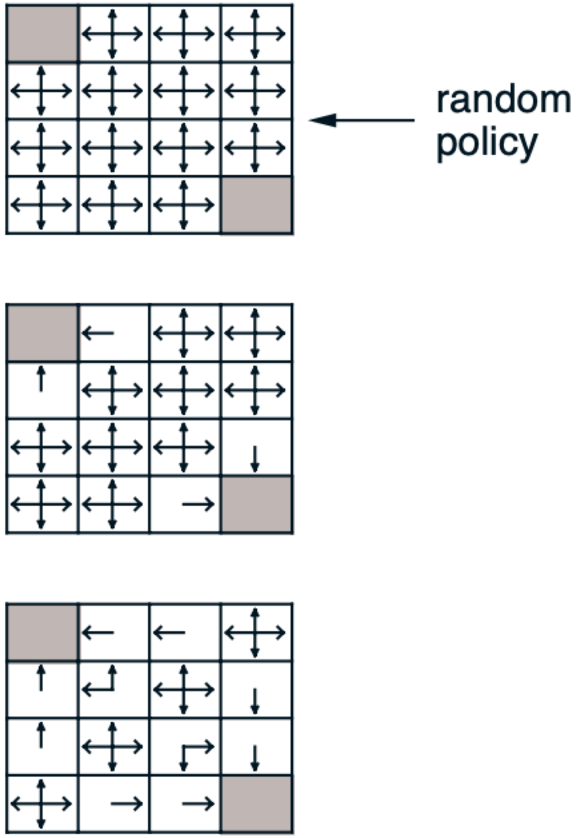
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

Greedy Policy w.r.t. v_k



Example: Gridworld

$k = 3$

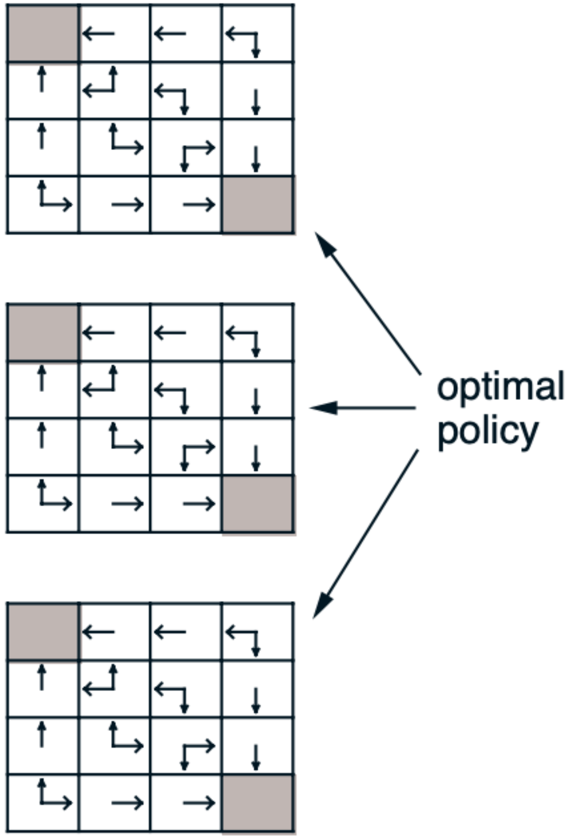
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Policy improvement

Recall that we know the reward and transition function of the MDP

- This means from state s we know all possible successor states s'

To act greedily, we select actions with the highest expected return

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \sum_{s',r} p(s',r|s,a)(r + \gamma V_\pi(s'))$$

Policy improvement

Recall that f the MDP

- This mean r states s'

To act greed ed return

Guaranteed to converge in finite-horizon MDPs
and discounted infinite-horizon MDPs.

Proof: Sutton and Barto, Chapter 4.2

$$a \in \mathcal{A} \qquad \sum_{s',r}$$

Value iteration: finding the optimal value fn

Idea: rather than computing the value function with respect to π, what if we compute the optimal value function directly?

1. Randomly initialize our value function $V_0(s)$ for all $s \in \mathcal{S}$
2. For iteration $k = 1,2, \ldots$

   Update $V_{k+1}(s)$ from $V_k(s')$ for all $s \in \mathcal{S}$

   If $\max_s |V_{k+1}(s) - V_k(s)| < \epsilon$ then stop
3. Compute policy from optimal value function V^*

Value iteration: finding the optimal value fn

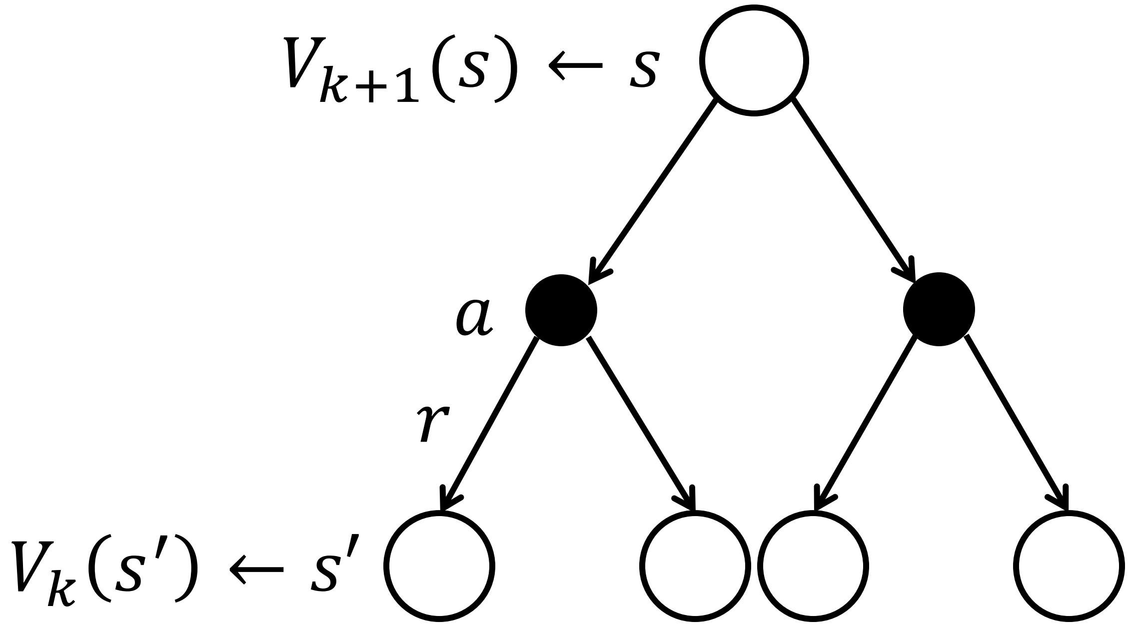
Idea: rather than computing the value function with respect to π, what if we compute the optimal value function directly?

Unlike iterative policy evaluation, V_k is not with respect to any explicit policy!

Intermediate value functions may not correspond to any policy at all.

3. Compute policy from optimal value function V^*

Value iteration



$$V_{k+1}(s) = \max_{a \in \mathcal{A}}(r + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a)\, V_k(s'))$$

Value iteration

To obtain the resulting policy, act greedily as before

$$\pi(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \sum_{s',r} p(s', r | s, a)(r + \gamma V^*(s'))$$

Value iteration

To obtain the resulting policy, act greedily as before

Guaranteed to converge in finite-horizon MDPs and discounted infinite-horizon MDPs.

Proof: Sutton and Barto, Chapter 4.4

Unknown Models: Monte Carlo Learning

The problem with known models

Policy and Value Iteration require fully-known MDPs:

- We must know the reward and transition functions

Is this a problem? Why?

The problem with known models

Policy and Value Iteration require fully-known MDPs:

- We must know the reward and transition functions

Is this a problem? Why?

Yes! In most practical applications we will know neither!

- Games, robotics, chatbots, ...

Monte Carlo reinforcement learning

Rather than relying on known environment models, we want to…

- Learn directly from experience (episodes)
- Require no knowledge of transitions/rewards

This is known as model-free learning!

Monte Carlo policy evaluation

Given a policy which generates episodes of experience…

$$s_1, a_1, r_2, s_2, a_2, r_3, \ldots \sim \pi$$

Previously, the value function is the expected return -- the weighted average of all possible returns that could be obtained from state s:

$$V_\pi = \mathbb{E}[G_t | s_t = s]$$

Monte Carlo policy evaluation

Given a policy which generates episodes of experience…

$$s_1, a_1, r_2, s_2, a_2, r_3, \ldots \sim \pi$$

Previously, the value function is the expected return -- the weighted average of all possible returns that could be obtained from state s:

$$V_\pi = \mathbb{E}[G_t | s_t = s]$$

In Monte Carlo learning we use the empirical mean of rewards from experience instead of the expected return

- *This is an approximation of the expected return!*

Monte Carlo policy evaluation

Idea: calculate the value of a state as the average of returns observed after visiting that state, using episodes sampled from π

Whenever state s is visited in an episode,

1. Increment visitation counter: $N(s) \leftarrow N(s) + 1$
2. Increment total return: $S(s) \leftarrow S(s) + G_t$

Estimate value by mean return: $V(s) = S(s)/N(s)$

- By the law of large numbers, $V(s) \rightarrow V_\pi(s)$ as $N(s) \rightarrow \infty$

Monte Carlo policy improvement

We can evaluate policies, but how can we **improve** them?

- What has to change about Policy Iteration?

Recall...

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \sum_{s',r} p(s', r | s, a)(r + \gamma V_\pi(s'))$$

Monte Carlo policy improvement

We can evaluate policies, but how can we **improve** them?

- What has to change about Policy Iteration?

Recall...

We don't have successor states!

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \sum_{s',r} p(s', r | s, a)(r + \gamma V_\pi(s'))$$

Monte Carlo policy improvement

We can evaluate policies, but how can we **improve** them?

- What has to change about Policy Iteration?

Solution: use the action-value function instead

- No successor states / transition function needed

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(s, a)$$

MC policy improvement with action-values

To obtain action-value empirical means instead of state-value...

MC policy improvement with action-values

To obtain action-value empirical means instead of state-value…

Whenever state-action (s, a) is visited in an episode,

1. Increment visitation counter: $N(s, a) \leftarrow N(s, a) + 1$
2. Increment total return: $S(s, a) \leftarrow S(s, a) + G_t$

Estimate value by mean return: $Q(s, a) = S(s, a)/N(s, a)$

MC policy improvement with action-values

There is an important assumption being made, what is it?

Hint: think about step 2

- Increment total return: $S(s, a) \leftarrow S(s, a) + G_t$

MC policy improvement with action-values

There is an important assumption being made, what is it?

Hint: think about step 2

- Increment total return: $S(s, a) \leftarrow S(s, a) + G_t$

To compute G_t we must have complete episodes!

- This means all episodes must terminate

Exploration-exploitation dilemma revisited

Since we're computing action-values using empirical means, we have to actually *try* sub-optimal actions to learn their values

We want to learn action-values for an optimal policy…but we need to act suboptimally to explore all actions

Exploration-exploitation dilemma revisited

Since we're computing action-values using empirical means, we have to actually *try* sub-optimal actions to learn their values

We want to learn action-values for an optimal policy…but we need to act sub-optimally to explore all actions

Easy solution: ϵ-greedy

Recap: ϵ-greedy

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon, \text{ if } a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(s,a) \\ \epsilon/m, \quad \text{otherwise} \end{cases}$$

For m actions

Recap: ϵ-greedy

$$\pi(a|s) = \begin{cases} \epsilon/m + 1 - \epsilon, \text{ if } a^* = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(s,a) \\ \epsilon/m, \qquad\qquad \text{otherwise} \end{cases}$$

Monte Carlo policy improvement with an ϵ-greedy policy is guaranteed to converge to the optimal action-value function

- *Caveat: with infinite exploration*
- Proof: Sutton / Barto, Chapter 5.4

Monte Carlo on-policy learning

Two-step iterative algorithm. Randomly initialize policy π...

- Evaluate the policy with sampled episodes

$$Q_\pi(s, a) \text{ approximated with empirical means}$$

- Improve the policy by acting ϵ-greedily with respect to V_π

$$\pi' = \epsilon\text{−greedy } Q(s, a)$$

Note: consider decaying ϵ to converge to an optimal policy

What is “on-policy” learning?

On-policy learning:

- “Learn on the job”
- The policy learns from its own experience
- Improve policy π from episodes sampled from π

Off-policy learning:

- “Look over someone’s shoulder”
- The policy learns from another policy’s experience
- Improve policy π from episodes sampled from β

Take-aways

- We evaluate policy quality by decomposing value functions with the Bellman equations
- In fully known MDPs (reward + transition), find the optimal policy using Policy Iteration and Value Iteration
- When we don’t know the full model, use Monte Carlo methods

Next time…

Temporal Difference learning (non-complete episodes?)

- This forms the basis of modern reinforcement learning

Introduction to deep reinforcement learning (and DQN)

Interactive development session with Pytorch / TorchRL?