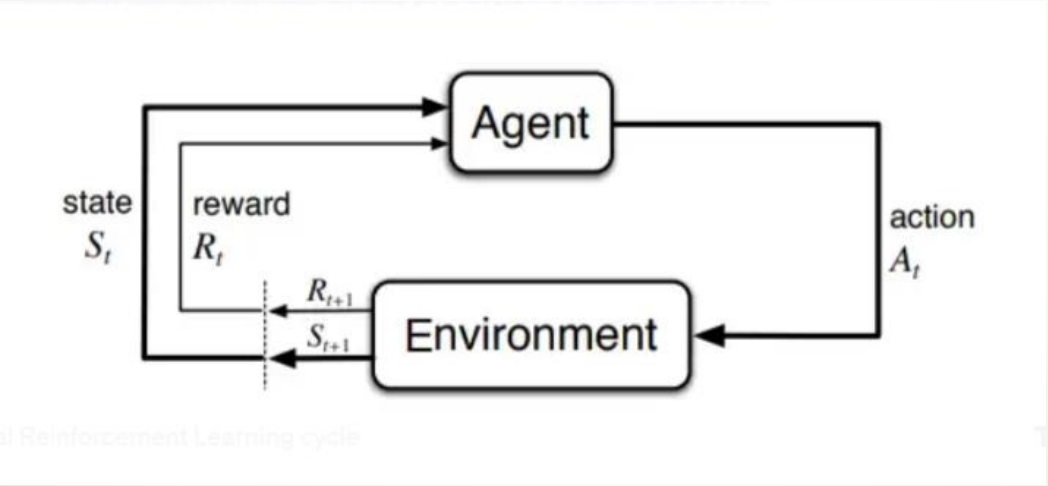
Markov Decision Process

Prof. Shivali Dhaka

OUTLINES

Reinforcement Learning : MDP

- *In a typical Reinforcement Learning (RL) problem, there is a learner and a decision maker called **agent** and the surrounding with which it interacts is called **environment**. The environment, in return, provides **rewards** and a **new state** based on the **actions** of the agent. So, in reinforcement learning, we do not teach an agent how it should do something but presents it with rewards whether positive or negative based on its actions.*



Reinforcement Learning : Markov-Decision Process (Part 1) | by blackburn | Towards Data Science

- ***How we formulate any problem in RL mathematically?***
  - *This is where the* ***Markov Decision Process****(MDP) comes in.*

Let's Discuss some terms:

- The Agent-Environment relationship
- Markov Property
- Markov Process and Markov chains
- Markov Reward Process (MRP)
- Bellman Equation

The Agent-Environment relationship

- ***Agent*** *: Software programs that make intelligent decisions and they are the learners in RL which interact with the environment by actions and receive rewards based on there actions.*
- ***Environment*** *:We can have a real-world environment or a simulated environment with which our agent will interact.*
- ***State*** *: This is the position of the agents at a specific time-step in the environment. So, whenever an agent performs a action the environment gives the agent reward and a new state where the agent reached by performing the action.*

- Actions can be **any decision we want the agent to learn** and **state can be anything which can be useful in choosing actions**.
- We do not assume that everything in the environment is unknown to the agent, for example, reward calculation is considered to be the part of the environment even though the agent knows a bit on how it's reward is calculated as a function of its actions and states in which they are taken.
- So, we can safely say that the agent-environment relationship represents the limit of the agent control and not it's knowledge.

Markov Property

- ***Transition*** *: Moving from one state to another is called Transition.*
- ***Transition Probability****: The probability that the agent will move from one state to another is called transition probability.*
- The Markov Property state that :

  "Future is Independent of the past given the present"

- Mathematically we can express this statement as :
- $P[s_{t+1} \mid s_t] = P[s_{t+1} \mid s_1, \ldots\ldots\ldots\ldots s_t]$
  - S[t] denotes the current state of the agent and s[t+1] denotes the next state. What this equation means is that the transition from state S[t] to S[t+1] is entirely independent of the past. So, the **RHS** of the Equation means the same as **LHS** if the system has a Markov Property. Intuitively meaning that our current state already captures the information of the past states.

State Transition Probability :

- For Markov State from S[t] to S[t+1] i.e. any other successor state , the state transition probability is given by
- $P_{ss'}=P[s_{t+1} = s' \ | s_t = s]$
- We can formulate the State Transition probability into a State Transition probability matrix by :

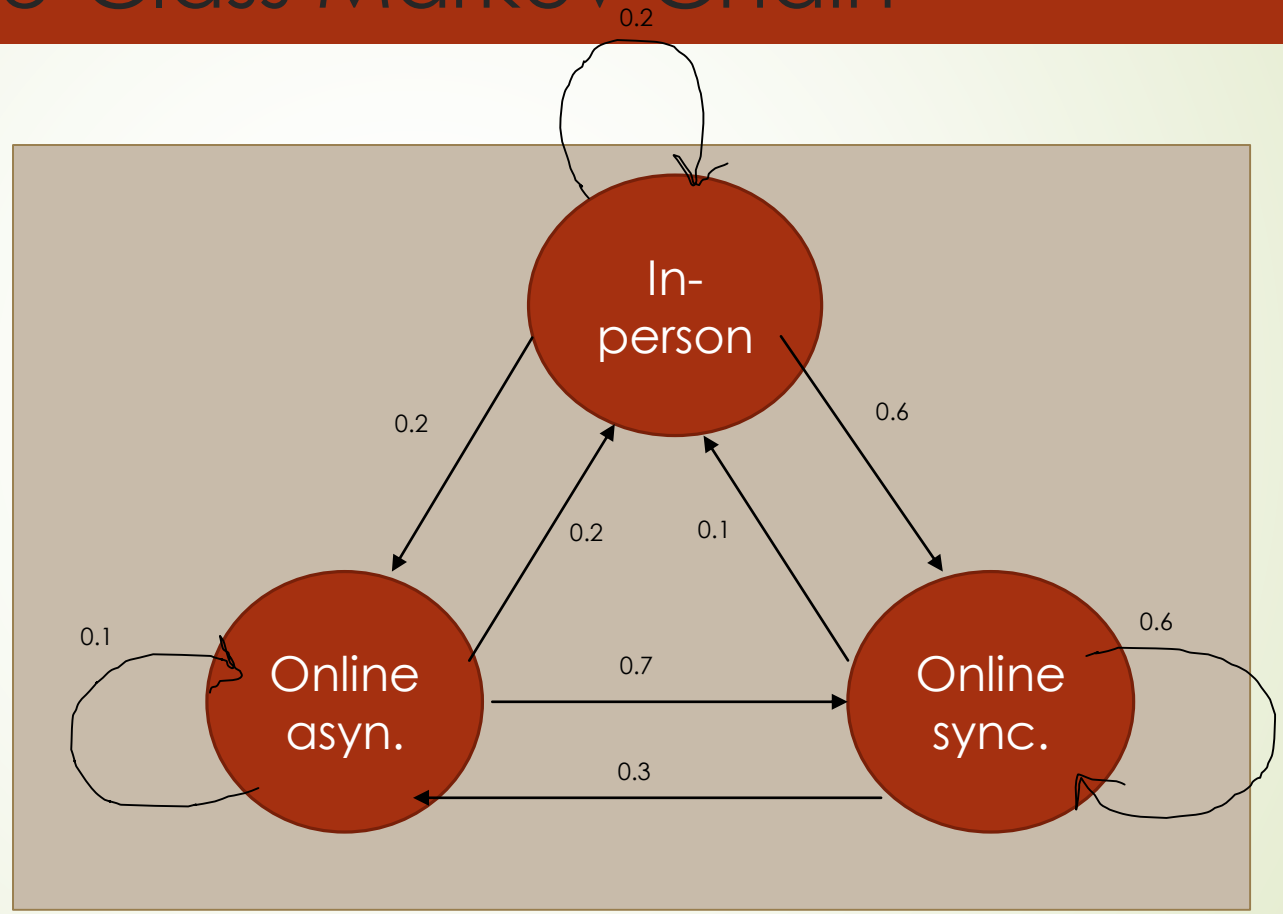
$$P = \begin{bmatrix} P_{11} & P_{12} \ldots\ldots\ldots\ldots\ldots\ldots P_{1n} \\ P_{21} & P_{22} \ldots\ldots\ldots\ldots\ldots\ldots P_{2n} \\ P_{n1} & P_{n2} \ldots\ldots\ldots\ldots\ldots P_{nn} \end{bmatrix}$$

- Each row in the matrix represents the probability from moving from our original or starting state to any successor state. Sum of each row is equal to 1.

Markov Process or Markov Chains

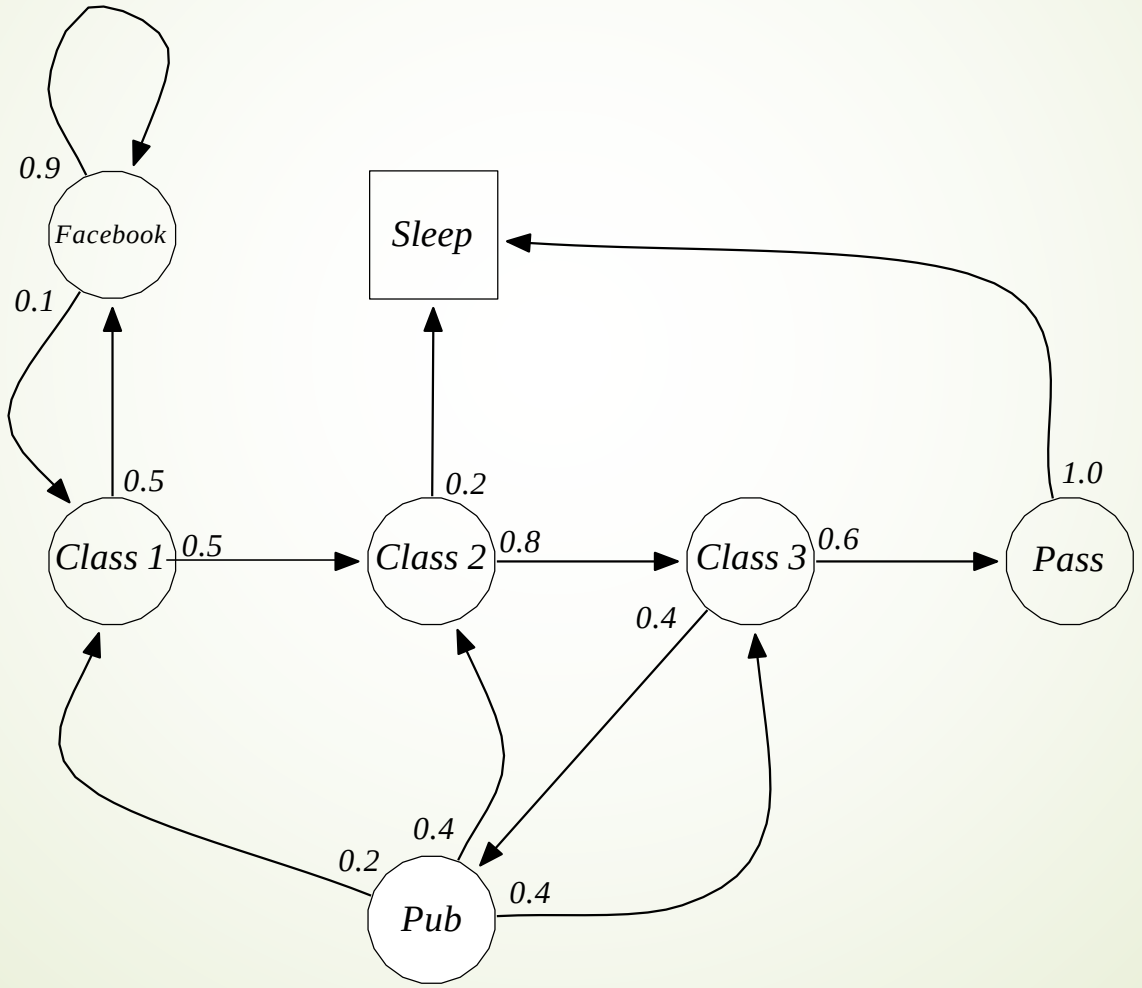
- It's basically a sequence of states with the Markov Property.
- It can be defined using a set of states(S) and transition probability matrix (P).The dynamics of the environment can be fully defined using the States(S) and Transition Probability matrix(P).

Markov Chain
Example-Class Markov Chain

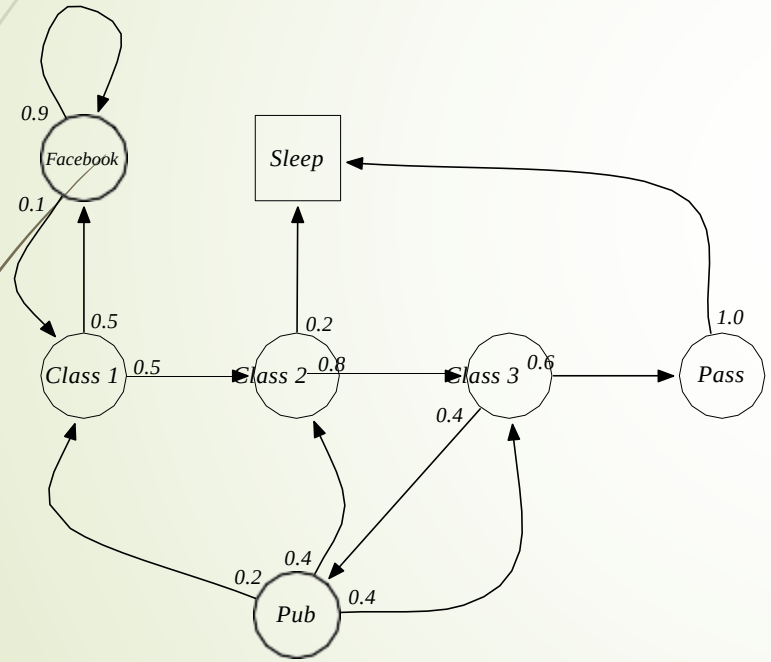


- The edges of the tree denote **transition probability**. From this chain let's take some sample. Now, suppose that we were attending in-person class and the according to the probability distribution there is a **0.6** chance that we will **attend online synchronous class** and **0.2** chance we **attend in-person class more** and again **0.2** that we will **attend online asynchronous class**. Similarly, we can think of other sequences that we can sample from this chain.
- Some samples from the chain :
- In-person — online sync — online async — in-person
- In-person — online async — online async — online sync
- In the above two sequences what we see is we get random set of States(S) (i.e. in-person, online sync, in-person) every time we run the chain. That is why Markov process is called random set of sequences.

Example: Student Markov Chain



Example: Student Markov Chain Episodes

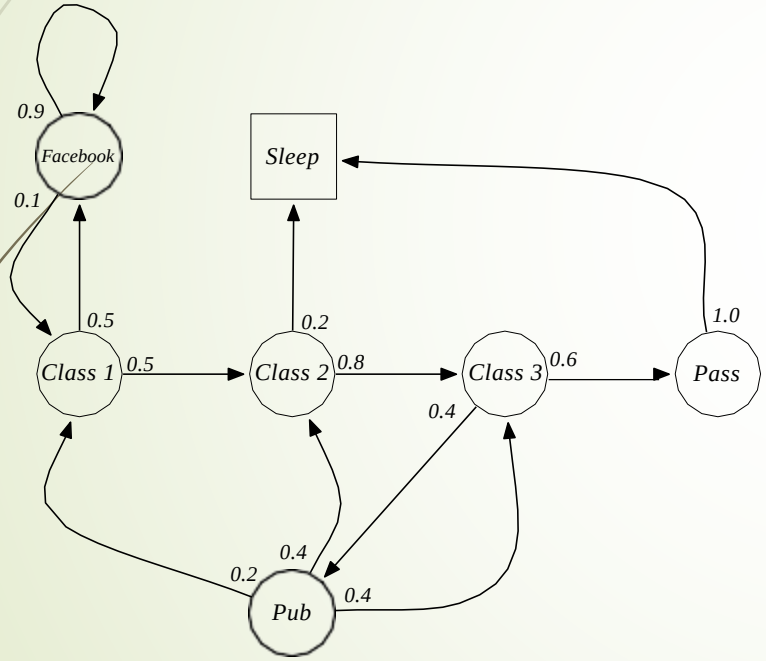


Sample episodes for Student Markov Chain starting from S_1 = C1

$S_1, S_2, \ldots, S_T$

- C1 C2 C3 Pass Sleep C1
- FB FB C1 C2 Sleep
- C1 C2 C3 Pub C2 C3 Pass Sleep
- C1 FB FB C1 C2 C3 Pub C1 FB FB FB C1 C2 C3 Pub C2 Sleep

Example: Student Markov Chain Transition Matrix



$$\mathcal{P} = \begin{array}{c} \\ C1 \\ C2 \\ C3 \\ Pass \\ Pub \\ FB \\ Sleep \end{array} \begin{array}{c} \begin{array}{ccccccc} C1 & C2 & C3 & Pass & Pub & FB & Sleep \end{array} \\ \begin{bmatrix} & 0.5 & & & & 0.5 & \\ & & 0.8 & & & & 0.2 \\ & & & 0.6 & 0.4 & & \\ & & & & & & 1.0 \\ 0.2 & 0.4 & 0.4 & & & & \\ 0.1 & & & & & 0.9 & \\ & & & & & & 1 \end{bmatrix} \end{array}$$

Reward and Returns

- ***Rewards*** *are the* ***numerical values*** *that the agent receives on performing some action at some* ***state(s)*** *in the environment. The numerical value can be positive or negative based on the actions of the agent.*
- *In* ***Reinforcement learning****, we care about* ***maximizing*** *the cumulative reward (all the rewards agent receives from the environment) instead of, the reward agent receives from the current state(also called immediate reward). This* ***total sum of reward*** *the agent receives from the environment is called* ***returns****.*
- We can define Returns(total Rewards from the environment) as :
  - $G_t = r_{t+1} + r_{t+2} + \cdots \ldots \ldots \ldots \ldots + r_t$
  - r[t+1] is the reward received by the agent at time step t[0] while performing an action(a) to move from one state to another. Similarly, r[t+2] is the reward received by the agent at time step t[1] by performing an action to move to another state. And, r[T] is the reward received by the agent by at the final time step by performing an action to move to another state.

Episodic and Continuous Tasks

- ***Episodic Tasks***: *These are the tasks that have a* ***terminal state*** *(end state).We can say they have finite states. For example, in racing games, we start the game (start the race) and play it until the game is over (race ends!). This is called an episode. Once we restart the game it will start from an initial state and hence, every* ***episode*** *is independent.*
- ***Continuous Tasks*** *: These are the tasks that have no ends i.e. they* ***don't have any terminal state****. These types of tasks will* ***never end****. For example, Learning how to code!*
- Now, it's easy to calculate the returns from the episodic tasks as they will eventually end but what about continuous tasks, as it will go on and on forever. The returns from sum up to infinity! So, how we define returns for continuous tasks?
- This is where we need **Discount factor(ɤ)**.

Discount Factor

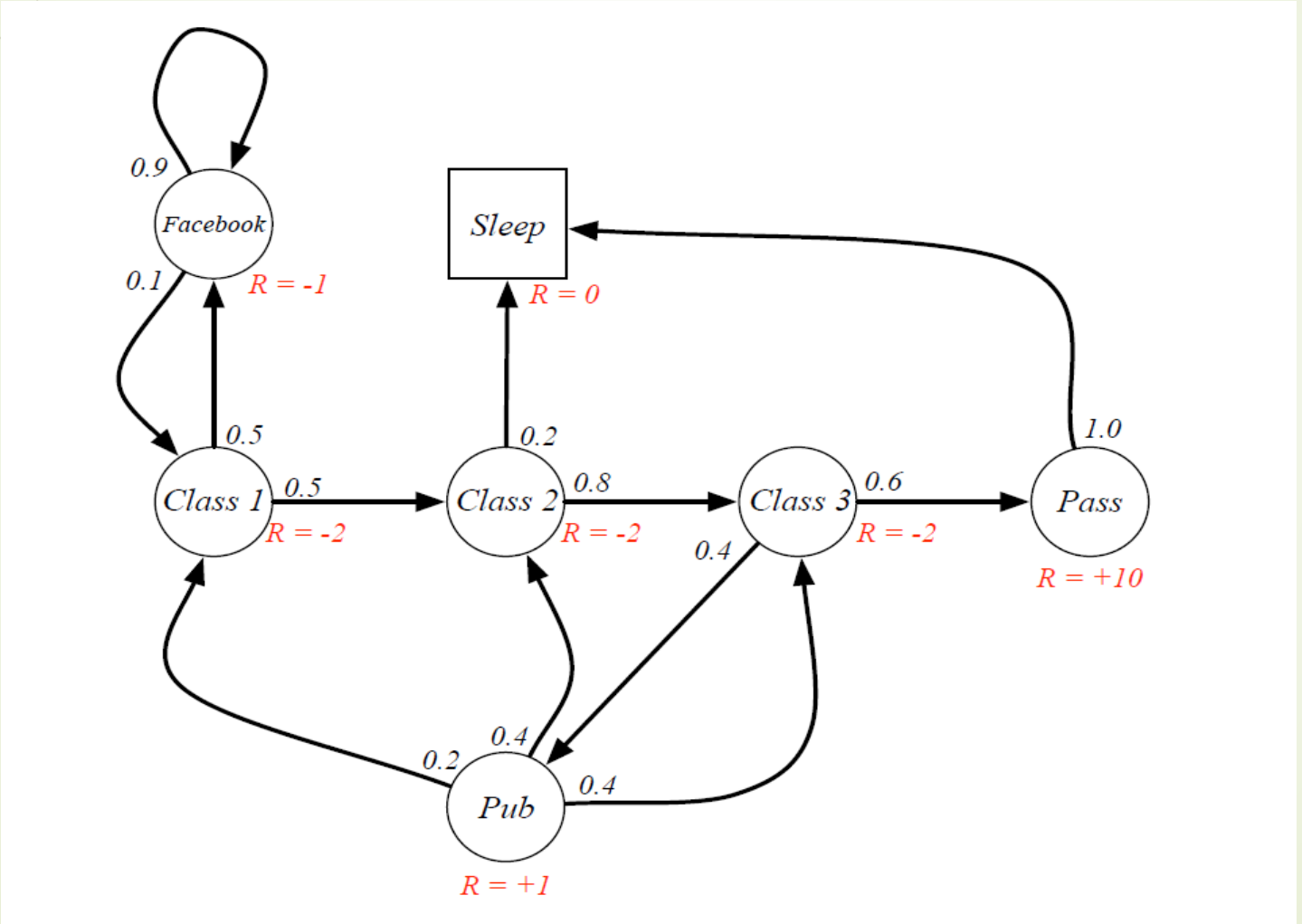
- ***Discount Factor (γ)***: *It determines how much* ***importance*** *is to be given to the* ***immediate reward and future rewards***. *This basically helps us to avoid* ***infinity*** *as a reward in continuous tasks. It has a value between 0 and 1. A value of* ***0*** *means that more importance is given to the* ***immediate reward*** *and a value of* ***1*** *means that more importance is given to* ***future rewards***. ***In practice***, *a discount factor of 0 will never learn as it only considers immediate reward and a discount factor of 1 will go on for future rewards which may lead to infinity. Therefore,* ***the optimal value for the discount factor lies between 0.2 to 0.8***.
- Returns using Discount Factor:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Markov Reward Process

- Reinforcement Learning is all about goal to maximize the reward. So, let's ***add reward*** to our Markov Chain. This gives us **Markov Reward Process.**
- A Markov reward process is a Markov chain with values.
- A *Markov Reward Process* is a tuple *(S, P, R, γ)* *S* is a finite set of states
  - *P* is a state transition probability matrix,
  - $P_{ssj} = P\,[S_{t+1} = s^j \mid S_t = s]$
  - *R* is a reward function, $R_s = E\,[R_{t+1} \mid S_t = s]$
    - This equation means how much reward (Rs) we get from a particular state S[t]. This tells us the immediate reward from that particular state our agent is in.
  - *γ* is a discount factor, $\gamma \in [0, 1]$

Example: Student MRP



Policy Function and Value Function

- **Value Function determines how good it is for the agent to be in a particular state**. Of course, to determine how good it will be to be in a particular state it must depend on some actions that it will take. This is where policy comes in. A policy defines what actions to perform in a particular state s.
- A policy is a simple function, that defines a **probability distribution** over Actions (a∈ A) for each state (s ∈ S). If an agent at time t follows a policy π then **π(a|s)** is the probability that the agent with taking action (a) at a particular time step (t).In Reinforcement Learning the experience of the agent determines the change in policy. Mathematically, a policy is defined as follows :
- Policy Function: $\pi(a|s)=P[A_t = a \,|S_t = s]$

Value Function

The value function $v(s)$ gives the long-term value of state s

Definition

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

$$v(s) = \mathrm{E}[G_t \mid S_t = s]$$

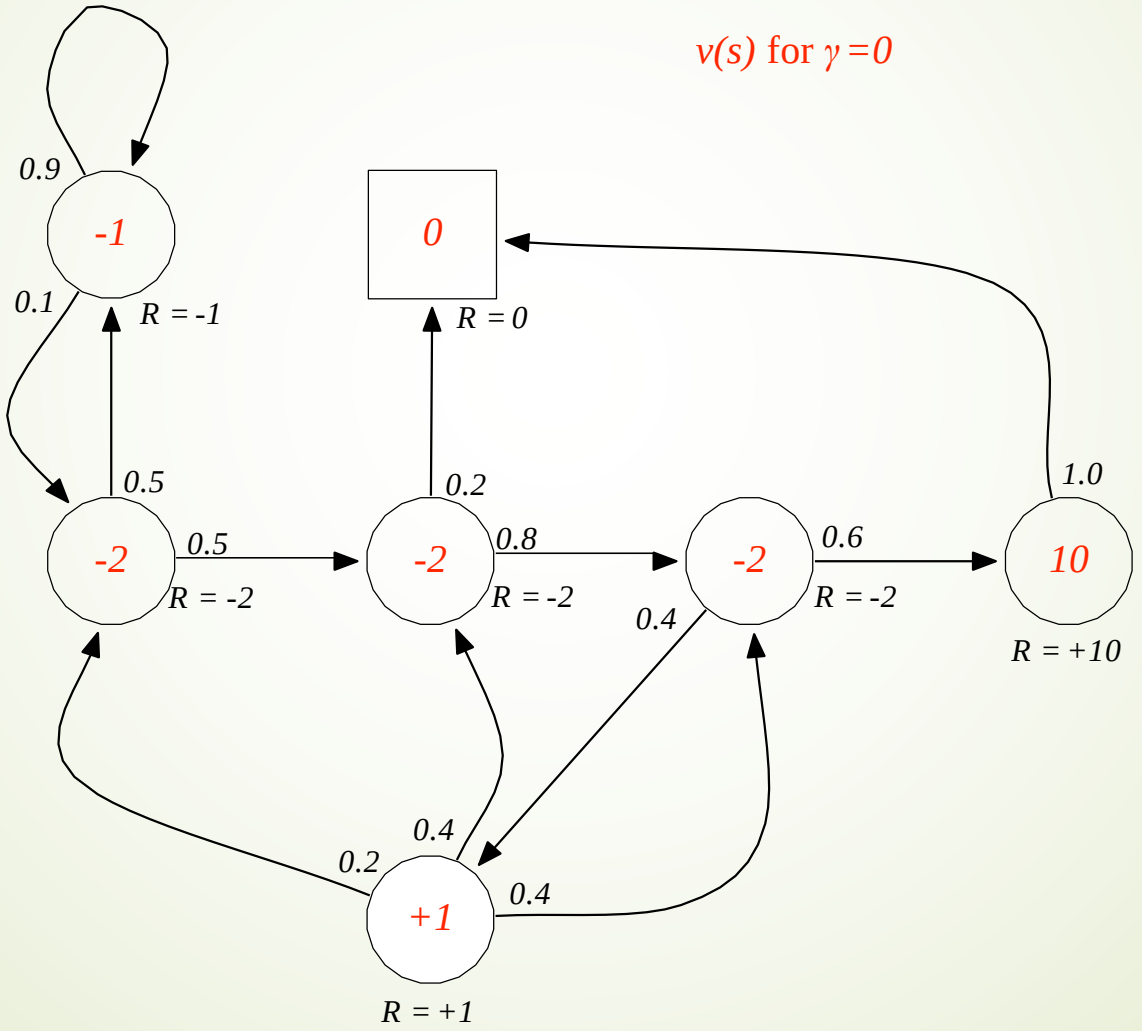
Example: Student MRP Returns

Sample returns for Student MRP:
Starting from $S_1 = \text{C1}$ with $\gamma = \frac{1}{2}$

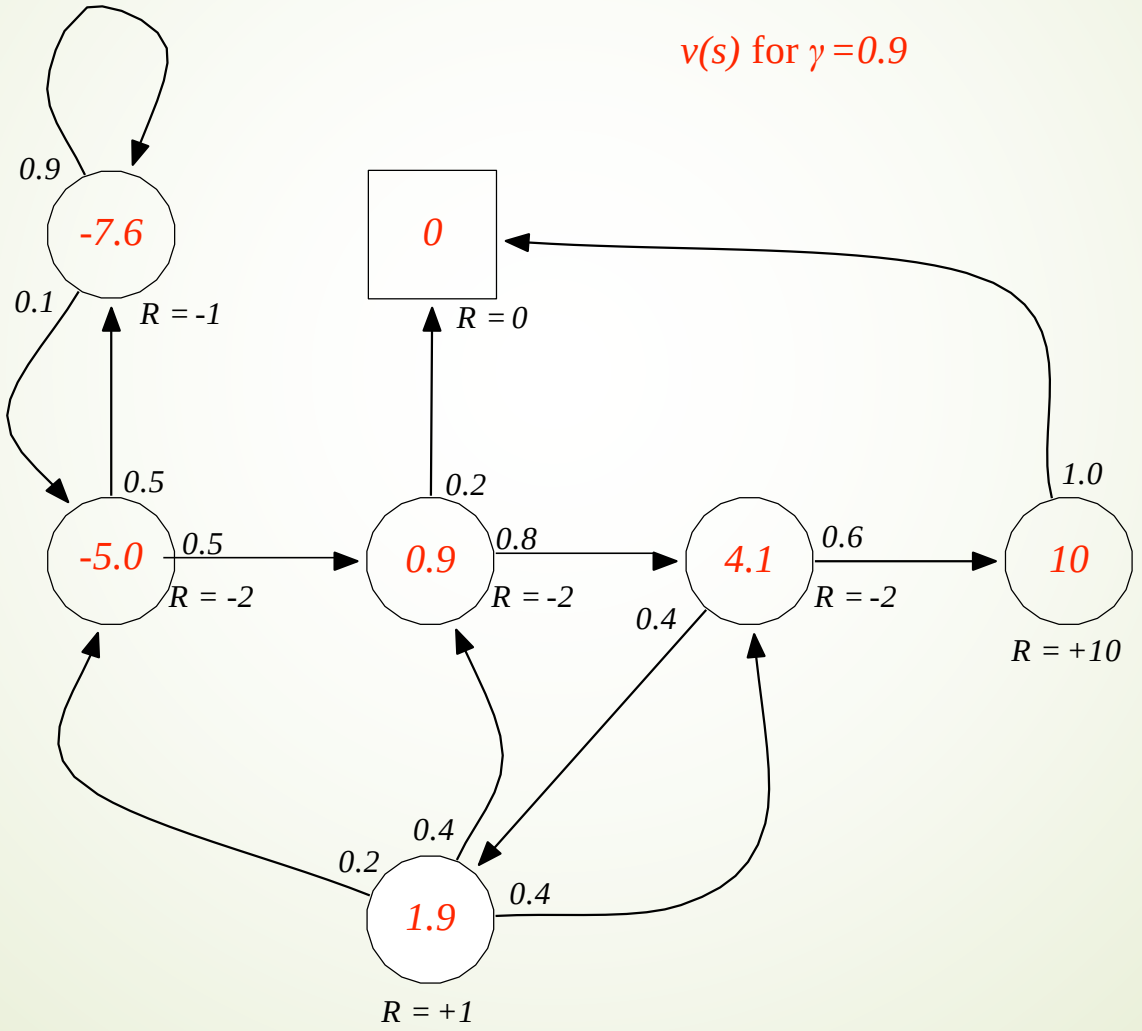
$$G_1 = R_2 + \gamma R_3 + \ldots + \gamma^{T-2} R_T$$

C1 C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 10 * \frac{1}{8}$	=	-2.25
C1 FB FB C1 C2 Sleep	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16}$	=	-3.125
C1 C2 C3 Pub C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 1 * \frac{1}{8} - 2 * \frac{1}{16} \ldots$	=	-3.41
C1 FB FB C1 C2 C3 Pub C1 ...	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16} \ldots$	=	-3.20
FB FB FB C1 C2 C3 Pub C2 Sleep			

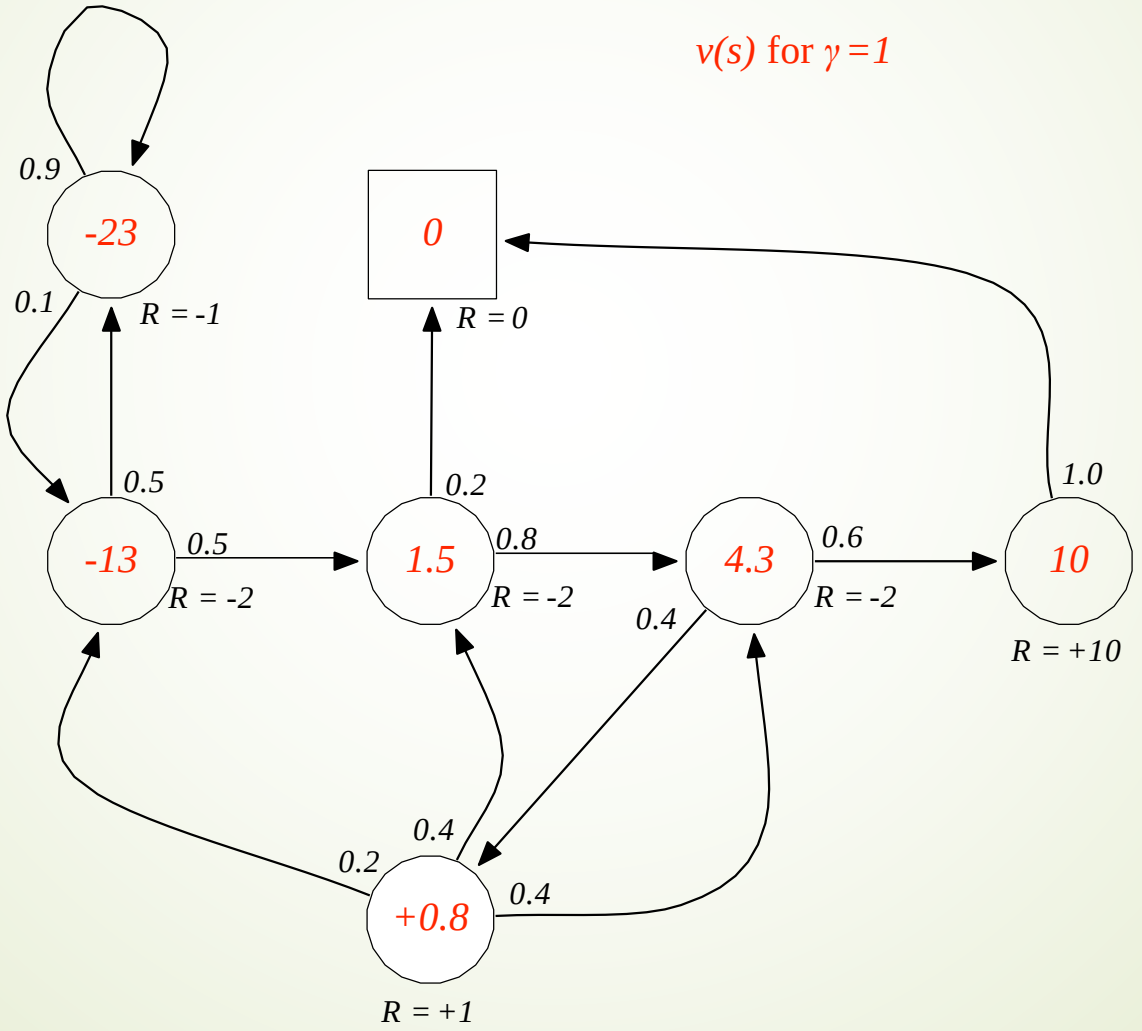
Example: State-Value Function for Student MRP (1)



Example: State-Value Function for Student MRP (2)



Example: State-Value Function for Student MRP (3)



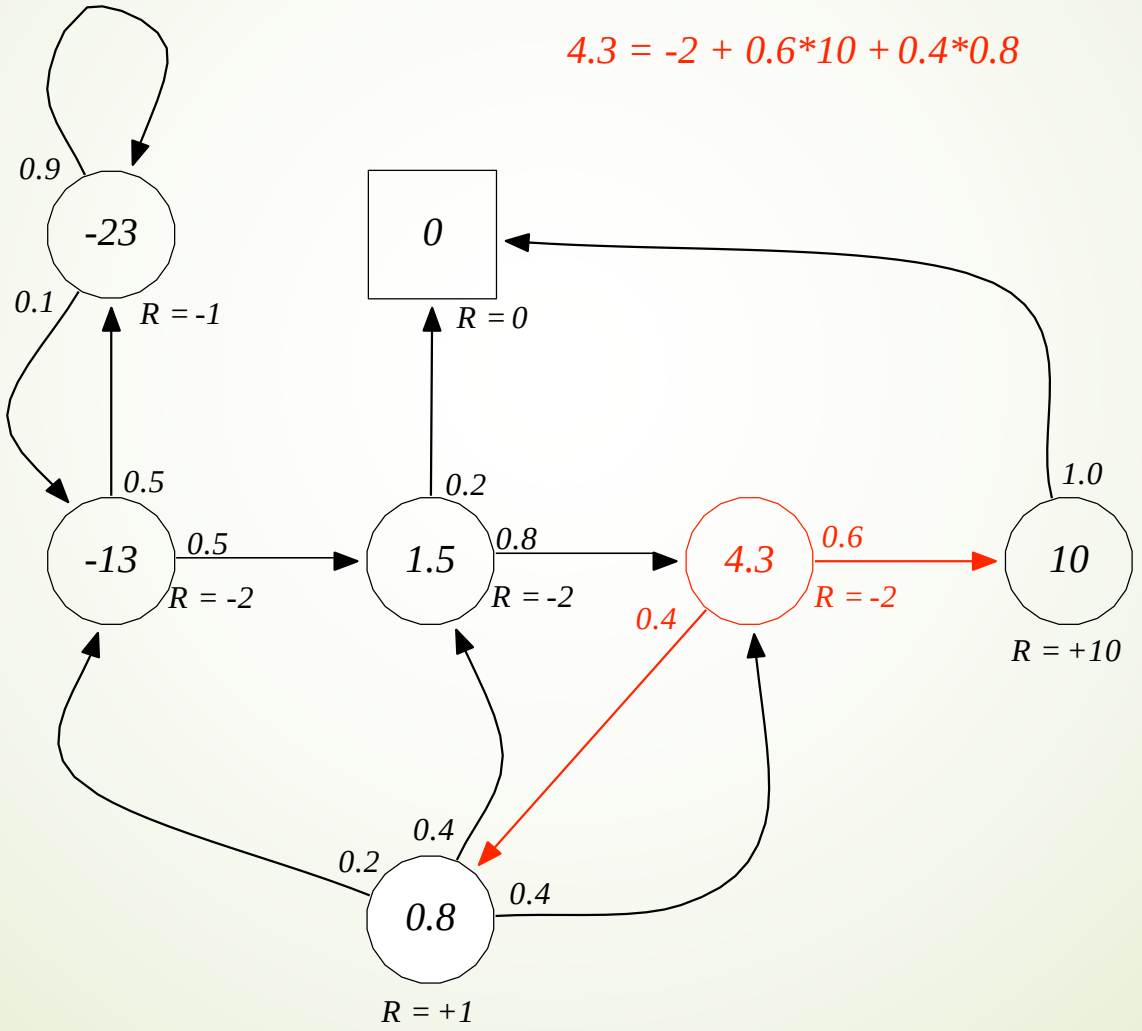
Bellman Equation for MRPs

The value function can be decomposed into two parts:

- immediate reward R_{t+1}
- discounted value of successor state $\gamma v(S_{t+1})$

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

Example: Bellman Equation for Student MRP



Bellman Equation in Matrix Form

The Bellman equation can be expressed concisely using matrices,

$$v = \mathcal{R} + \gamma \mathcal{P} v$$

where v is a column vector with one entry per state

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{11} & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

Solving the Bellman Equation

- The Bellman equation is a linear equation.
- It can be solved directly:

$$v(s) = R_s + \gamma \sum_{s' \epsilon S} P_{ss'} v(s')$$

- The above equation can be expressed in matrix form as follows :

$$v = R + \gamma P v$$

$$(1 - \gamma P)v = R$$

- $v = (1 - \gamma P)^{-1} R$ - ->Bellman Linear Equation
  - Where v is the value of state we were in, which is equal to **the immediate reward plus the discounted value of the next state multiplied by the probability of moving into that state**.

Contd…

- Computational complexity is $O(n^3)$ for n states.
- Direct solution only possible for small MRPs
- There are many iterative methods for large MRPs, e.g.
  - Dynamic programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning

Markov Decision Process

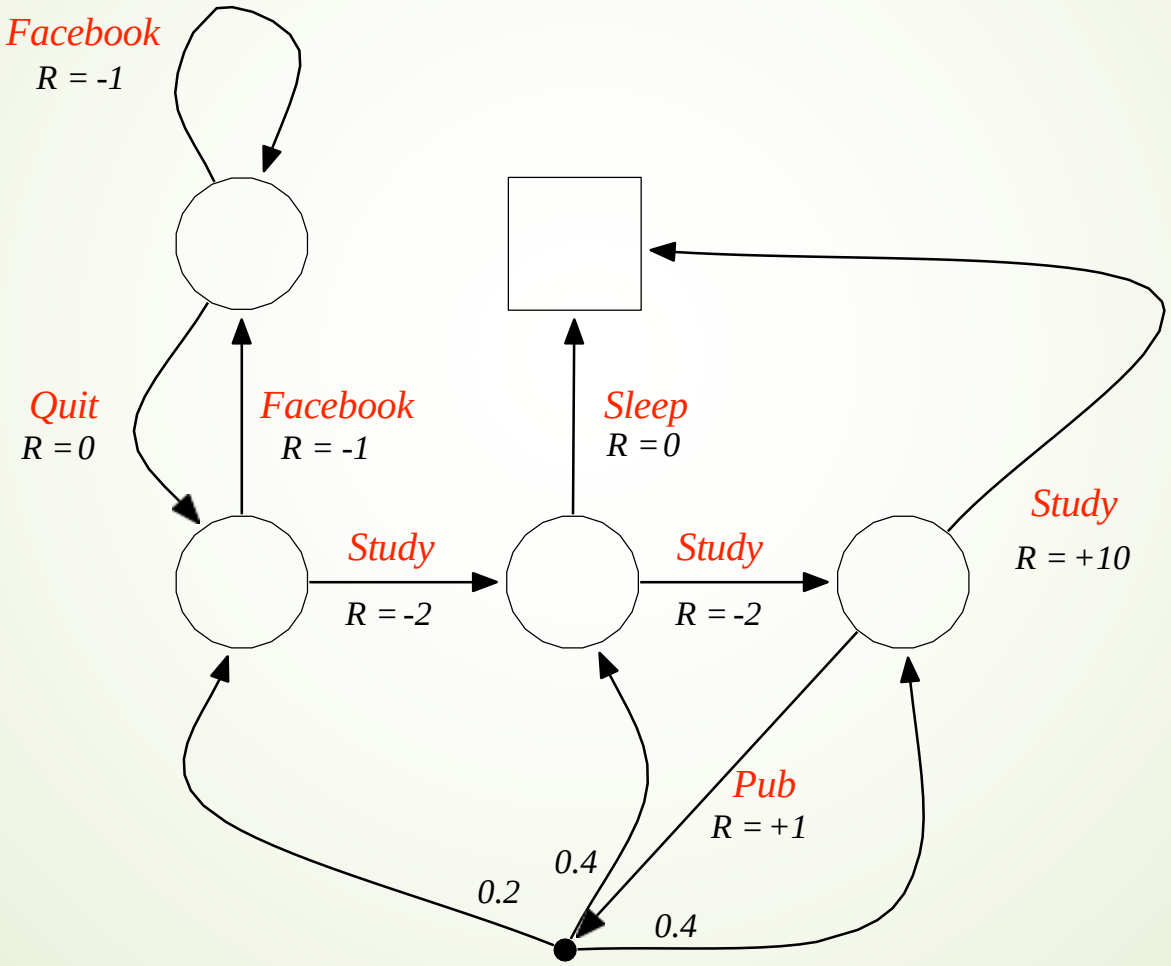
A Markov decision process (MDP) is a Markov reward process with decisions. It is an *environment* in which all states are Markov.

Definition

A *Markov Decision Process* is a tuple (S, A, P, R, γ)

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

Example: Student MDP



Policies (1)

Definition

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

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

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

Policies (2)

- Given an MDP $M = (S, A, P, R, \gamma)$ and a policy π
- The state sequence $S_1, S_2, \ldots$ is a Markov process (S, P^{π})
- The state and reward sequence $S_1, R_2, S_2, \ldots$ is a Markov reward process $(S, P^{\pi}, R^{\pi}, \gamma)$
- where

$$\mathcal{P}^{\pi}_{s,s'} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{P}^{a}_{ss'}$$

$$\mathcal{R}^{\pi}_{s} = \sum_{a \in \mathcal{A}} \pi(a|s) \mathcal{R}^{a}_{s}$$

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

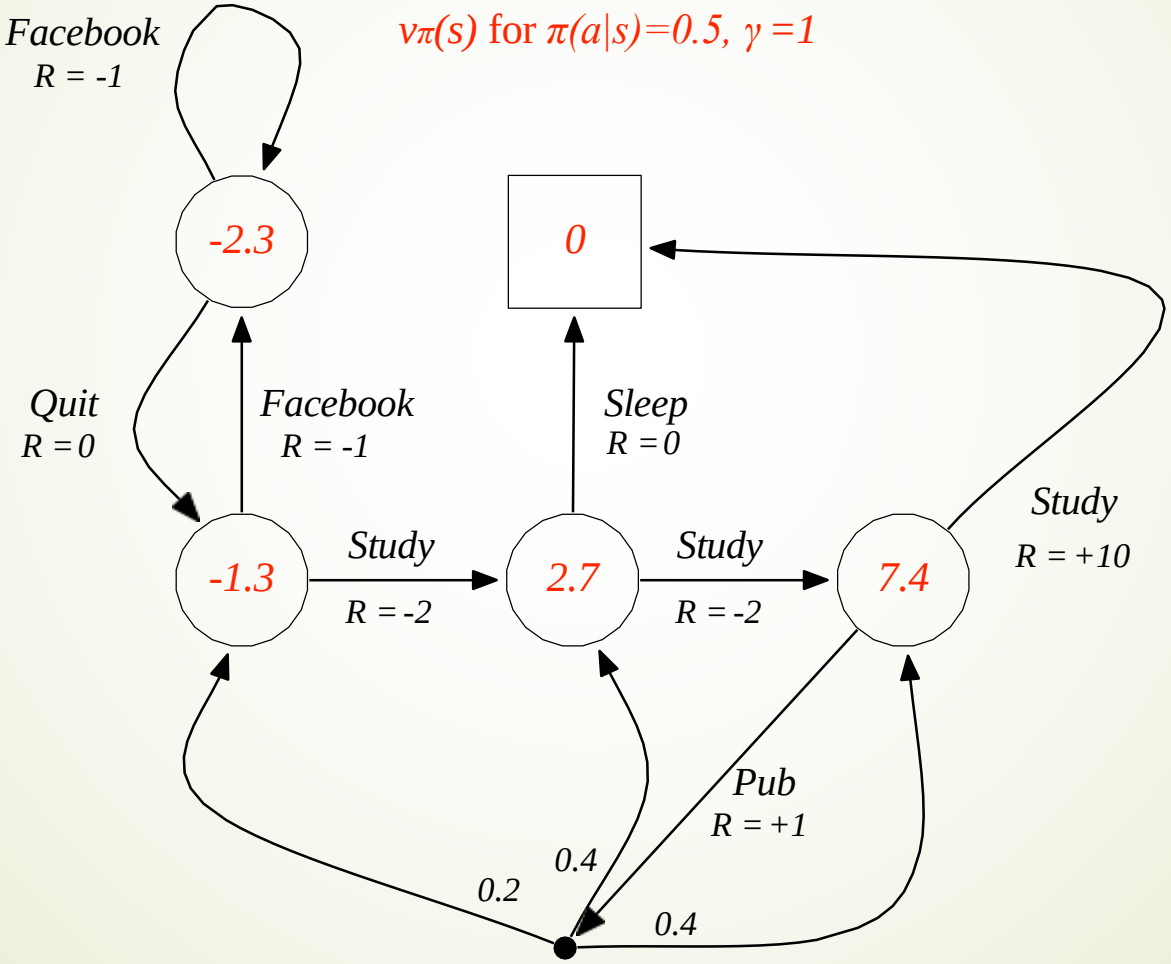
$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

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

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

Example: State-Value Function for Student MDP



Bellman Expectation Equation

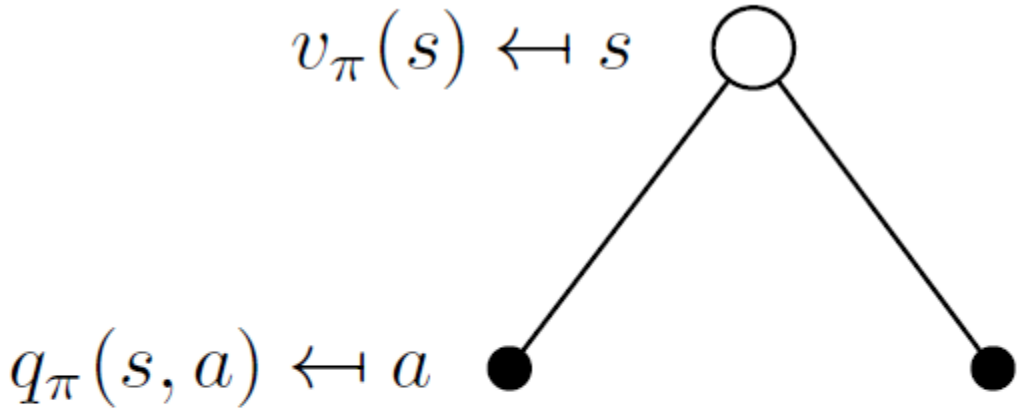
The state-value function can again be decomposed into immediate reward plus discounted value of successor state,

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

action-value function can similarly be decomposed,

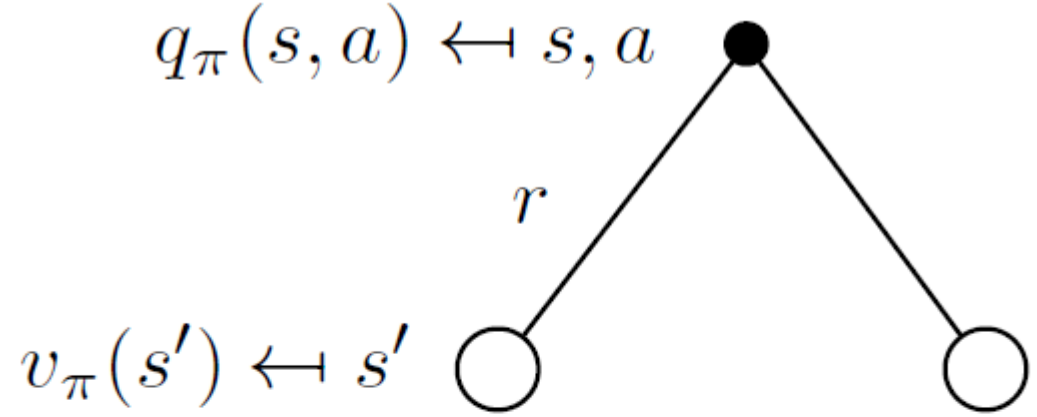
$$q_\pi(s, a) = \mathrm{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a]$$

Bellman Expectation Equation for V^{Π}



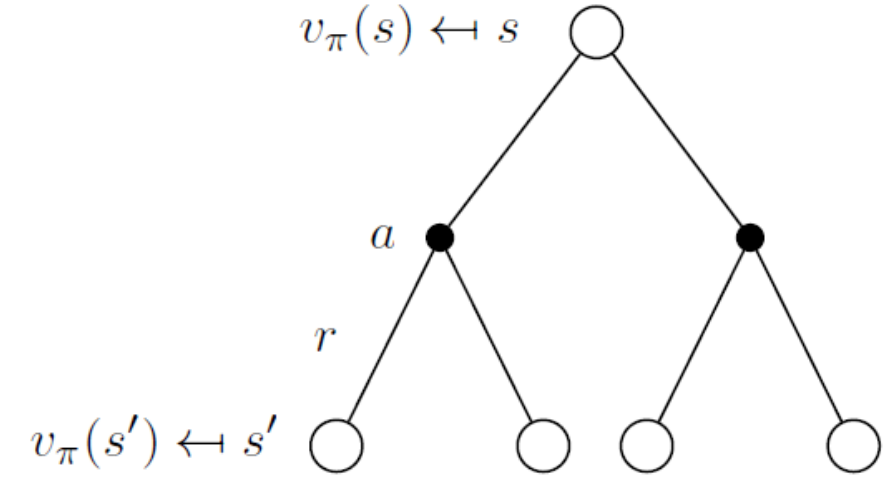
$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

Bellman Expectation Equation for Q^{π}



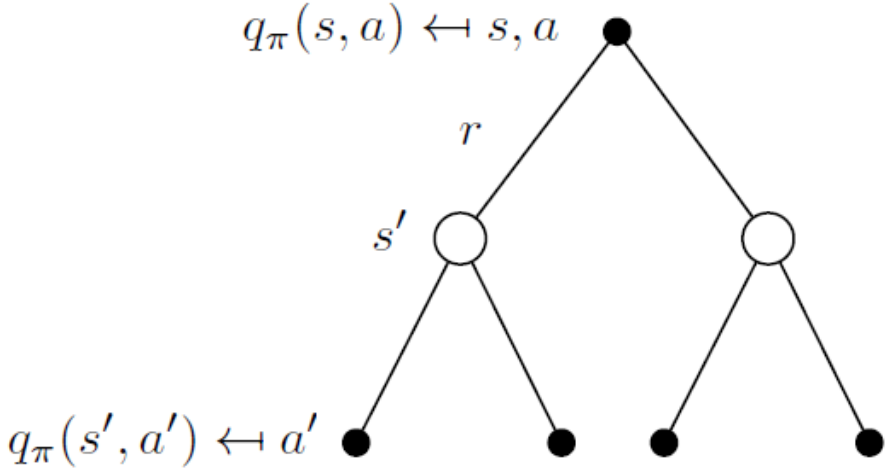
$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

Bellman Expectation Equation for v_Π (2)



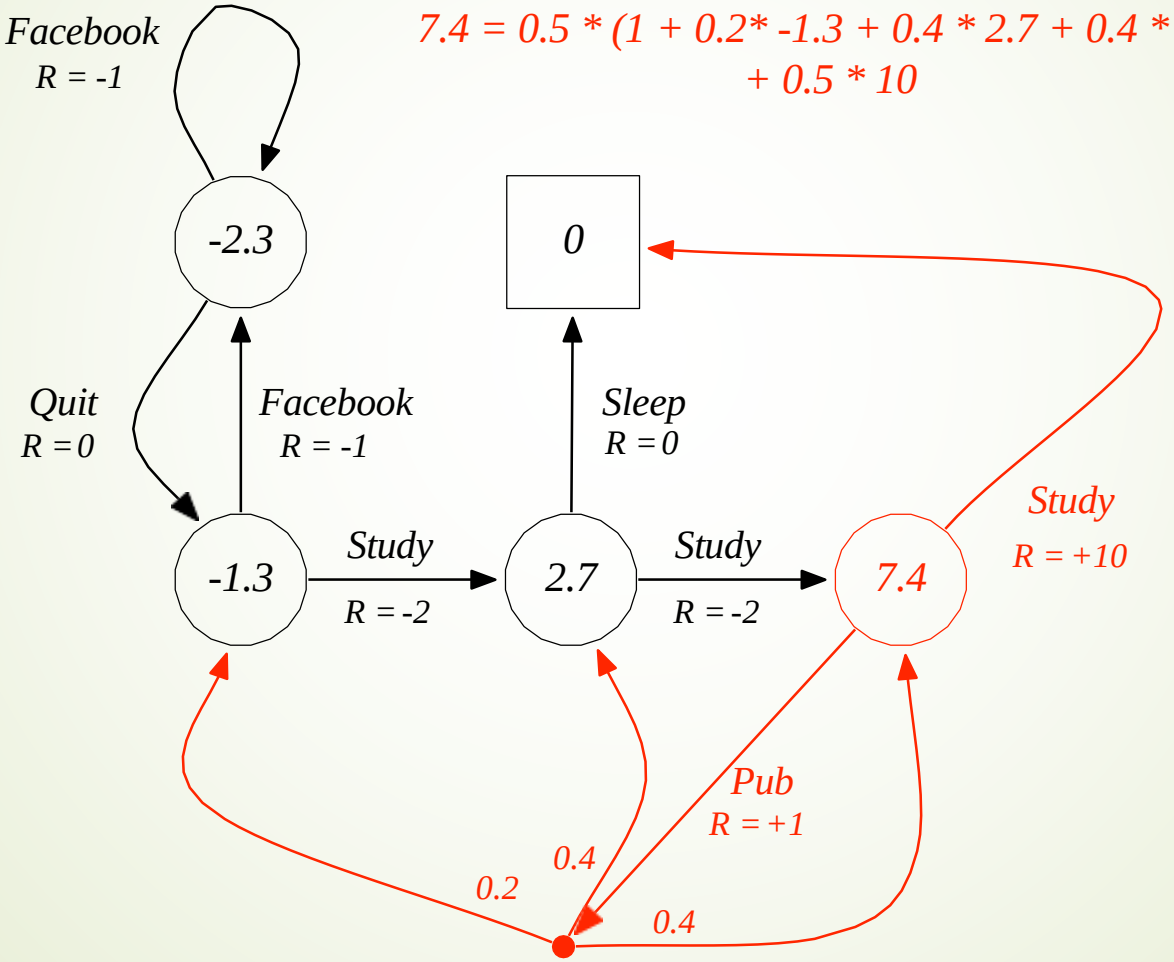
$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

Bellman Expectation Equation for q_Π (2)



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

Example: Bellman Expectation Equation in Student MDP



Bellman Expectation Equation (Matrix Form)

The Bellman expectation equation can be expressed concisely using the induced MRP,

$$v_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi$$

with direct solution

$$v_\pi = (I - \gamma \mathcal{P}^\pi)^{-1} \mathcal{R}^\pi$$

Optimal Value Function

Definition

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

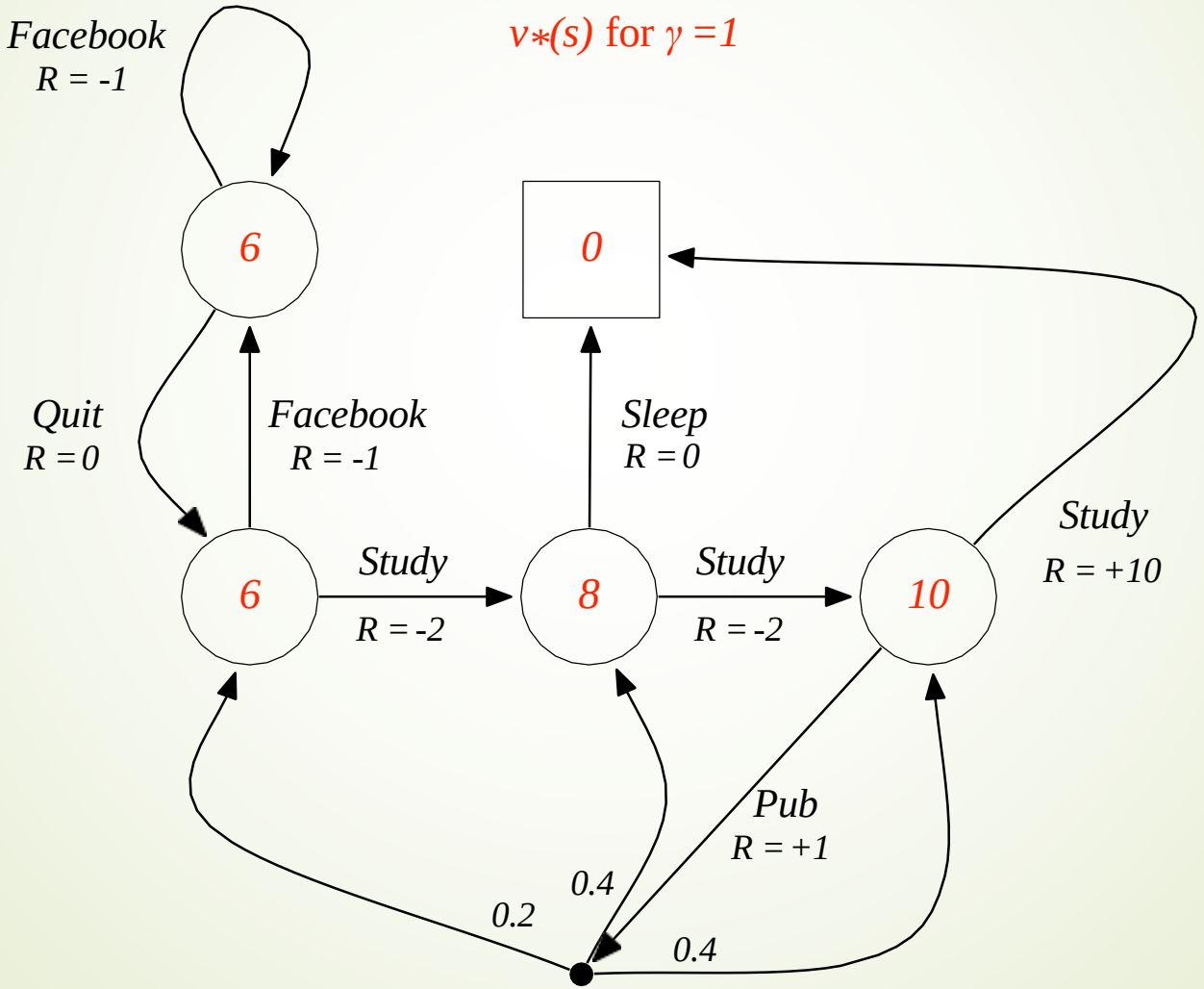
$$v_*(s) = \max_\pi v_\pi(s)$$

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

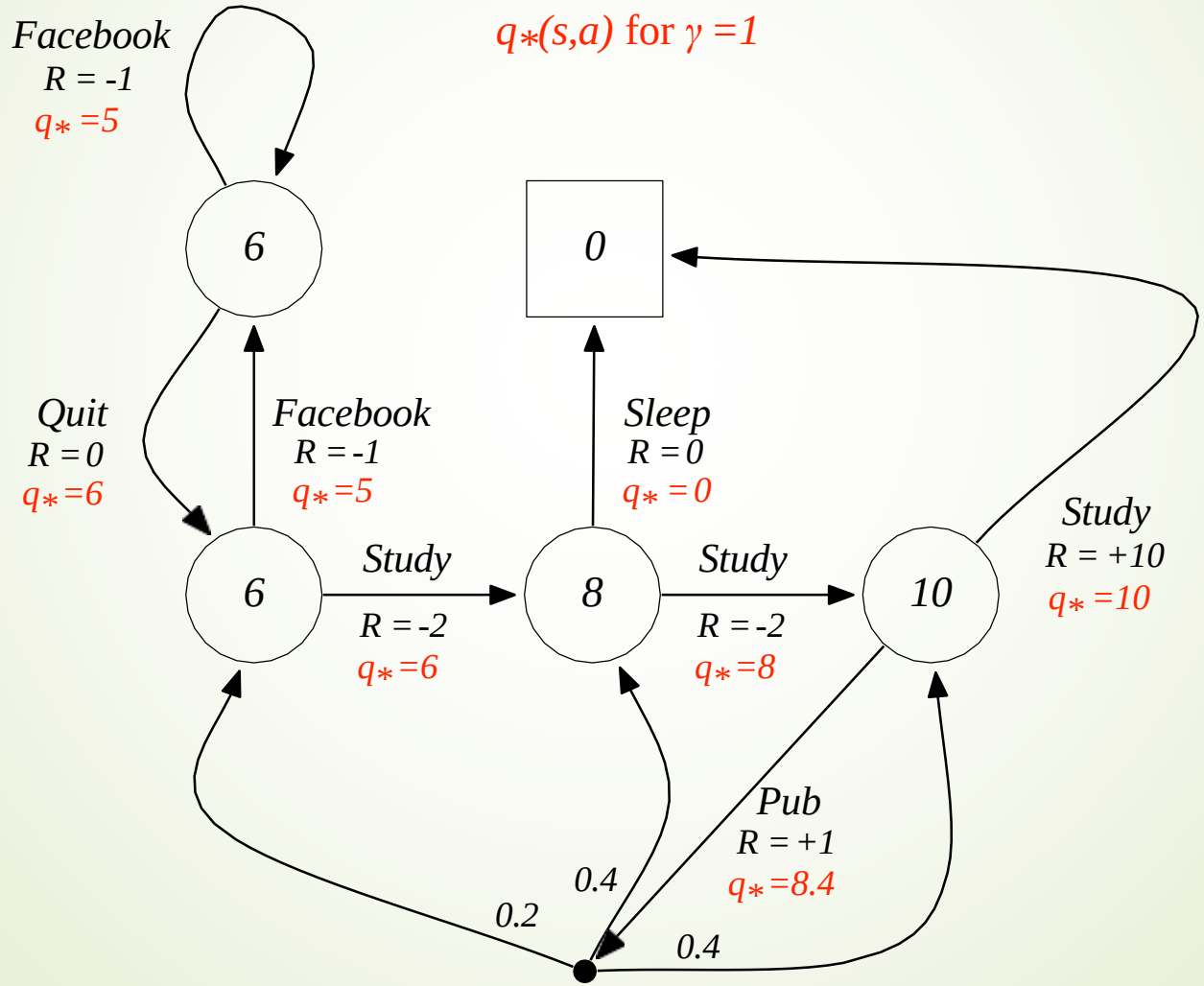
$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

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

Example: Optimal Value Function for Student MDP



Example: Optimal Action-Value Function for Student MDP



Optimal Policy

Define a partial ordering over policies

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

Theorem

For any Markov Decision Process

- *There exists an optimal policy 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, $q_{\pi_*}(s, a) = q_*(s, a)$*

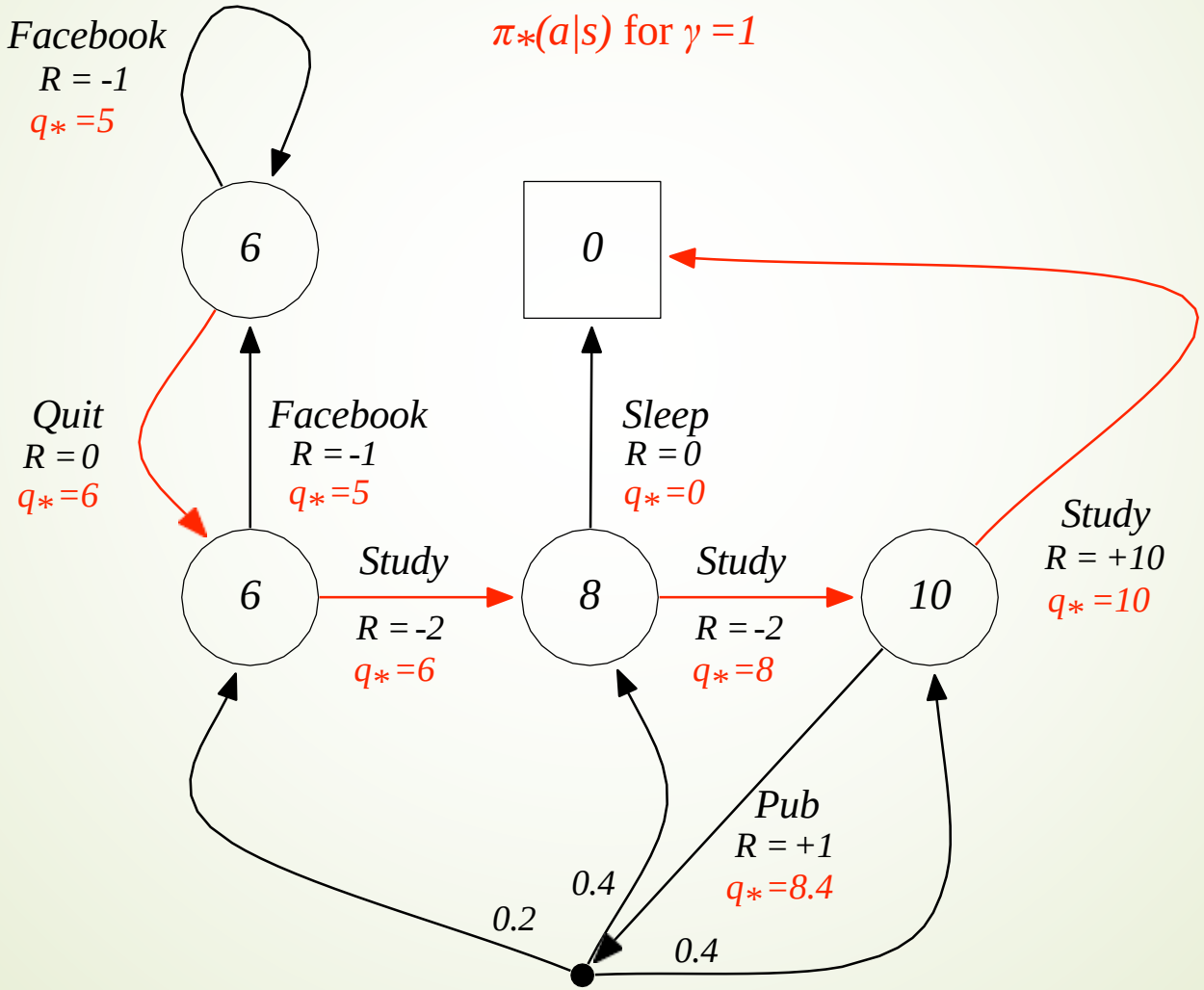
Finding an Optimal Policy

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

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

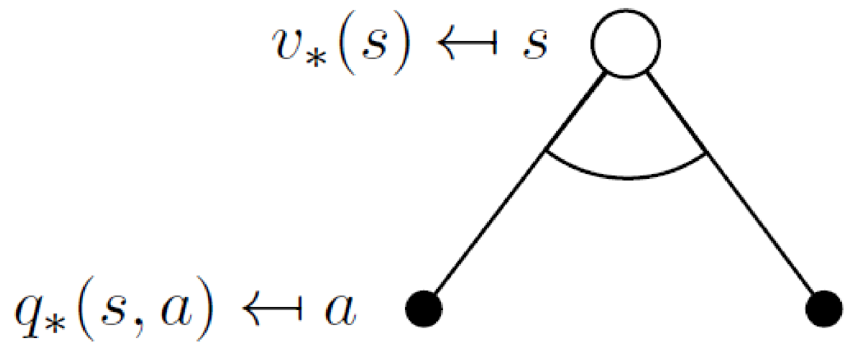
- There is always a deterministic optimal policy for any MDP If
- we know $q_*(s, a)$, we immediately have the optimal policy

Example: Optimal Policy for Student MDP



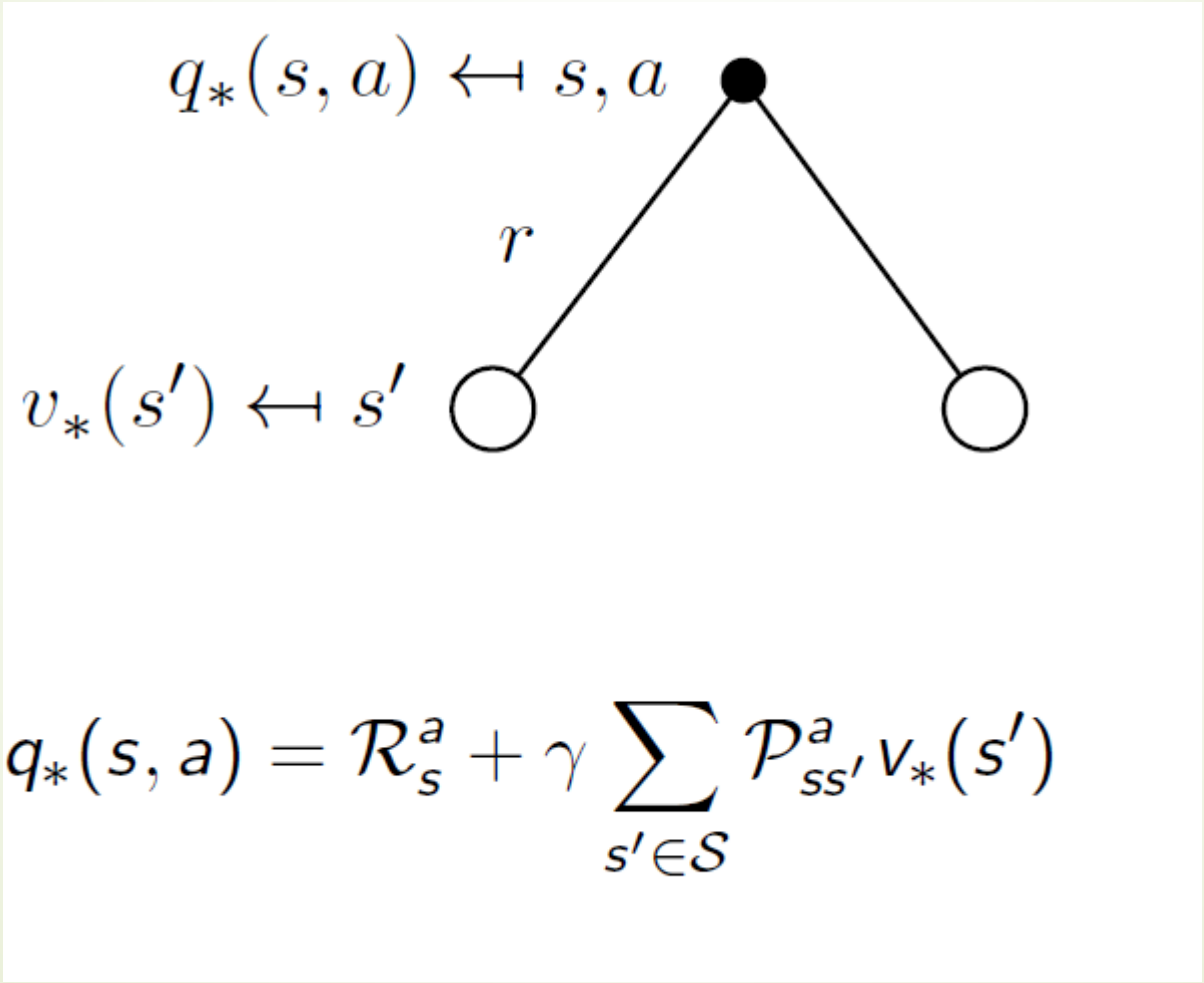
Bellman Optimality Equation for v_*

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



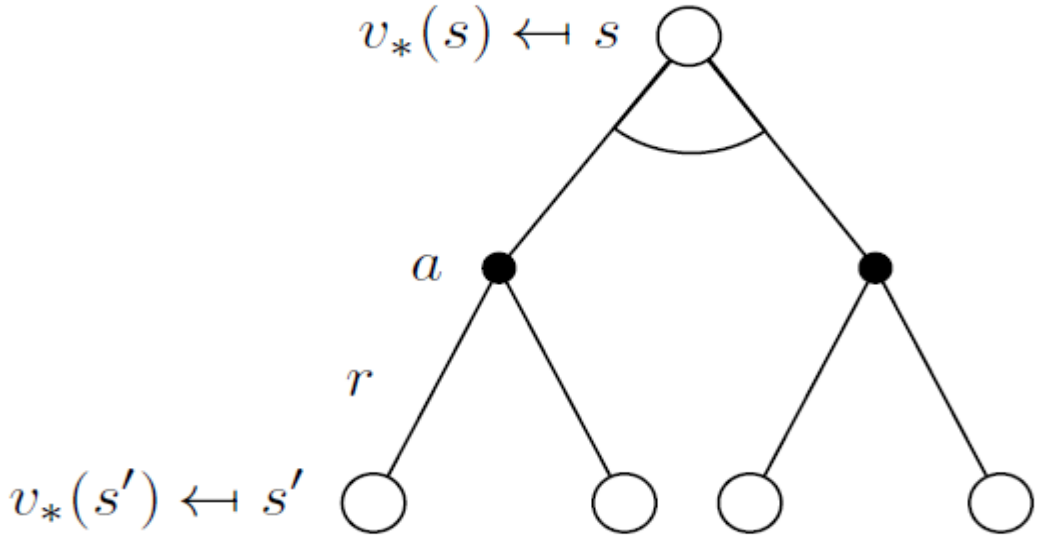
$$v_*(s) = \max_a q_*(s,a)$$

Bellman Optimality Equation for Q^*



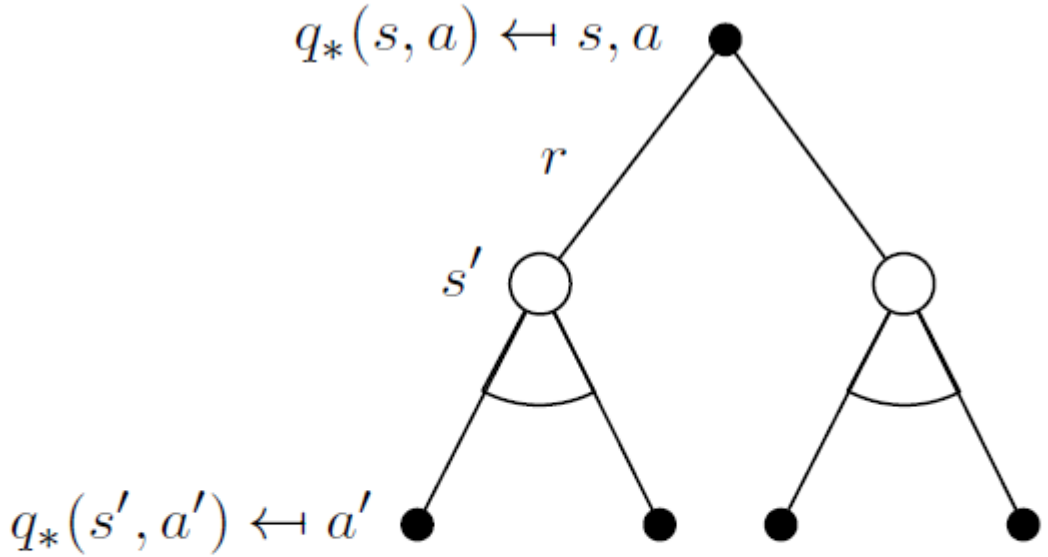
$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

Bellman Optimality Equation for $V^*(2)$



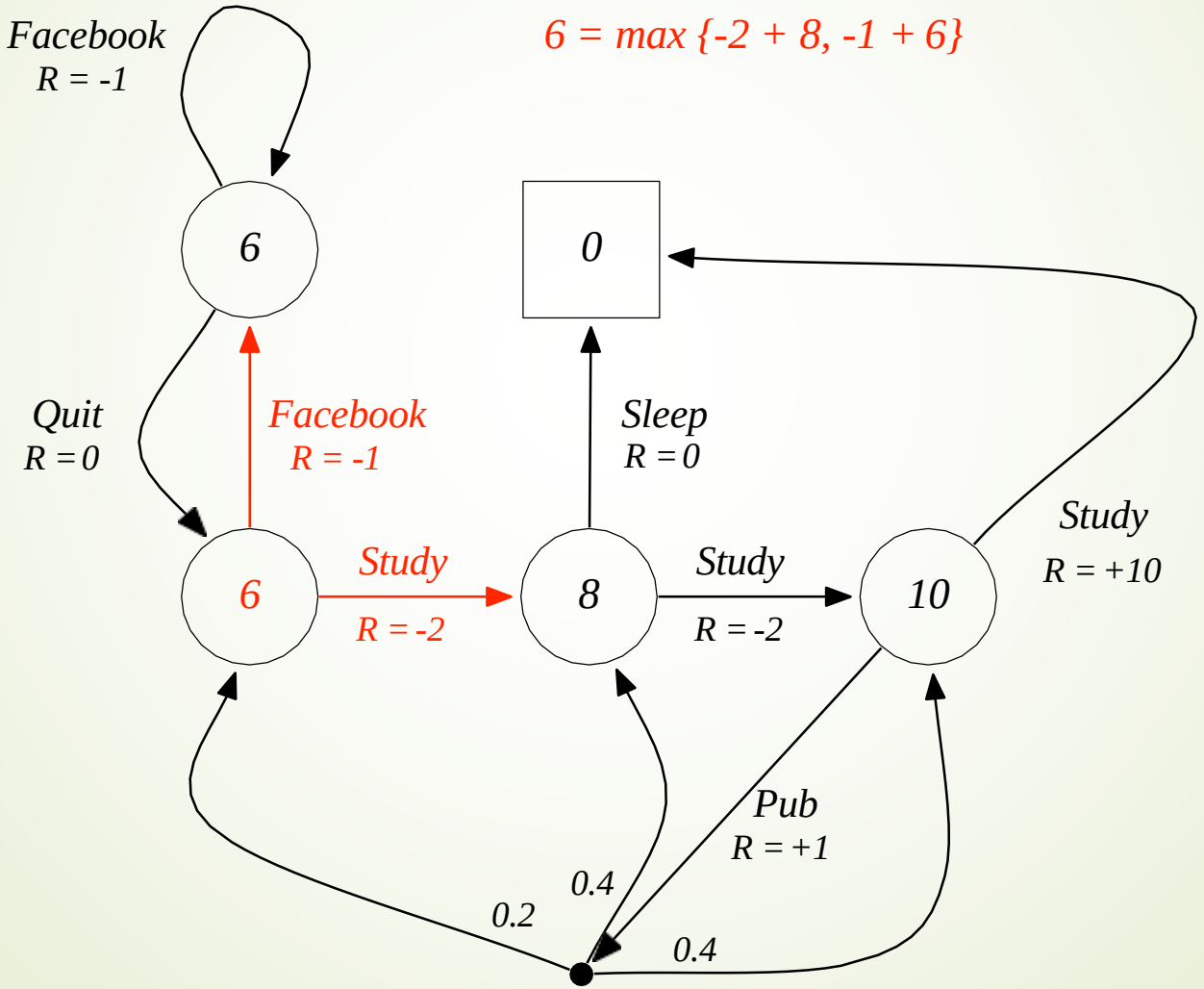
$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

Bellman Optimality Equation for Q^* (2)



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

Example: Bellman Optimality Equation in Student MDP



Solving the Bellman Optimality Equation

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