Markov Decision Process

Mattia Pellegrino, Ph.D Fellow
mattia.pellegrino@unipr.it

Summary

- Markov Processes
- Markov Reward Processes
- Markov Decision Processes

INTRODUCTION

- Markov decision processes describe formally an environment for reinforcement learning

- Where the environment is fully observable
    - We know everything

- Almost all RL problems can be described as MDP
    - In some ways we can hypothesize all the problems as MDP
    - Partially observable problems can be converted into MDPs

MARKOV PROPERTY

An information state (a.k.a. Markov state) contains all useful information from the history

- ***A state S_{t} is a Markov state if and only if***

$$\mathbb{P}\left[S_{t+1} \mid S_t\right] = \mathbb{P}\left[S_{t+1} \mid S_1, \ldots, S_t\right]$$

- The future is independent of the past given present

$$H_{1:t} \longrightarrow S_t \longrightarrow H_{t+1:\infty}$$

- Once the state is known, the history is irrelevant
- The environment state S^e_t is a Markov State
- The history H_t is a Markov state

STATE TRANSITION MATRIX

- For every Markov process we can define the transition probability from a state to another on

$$P_{s,s'} = P[S_{t+1} = s' \mid S_t = s]$$

- State transition matrix P defines the transition probabilities from all states s to all successor states s'

$$P = \text{ from } [\begin{matrix} P_{11} \ldots P_{1n} \\ P_{n1} \ldots P_{nn} \end{matrix}]$$

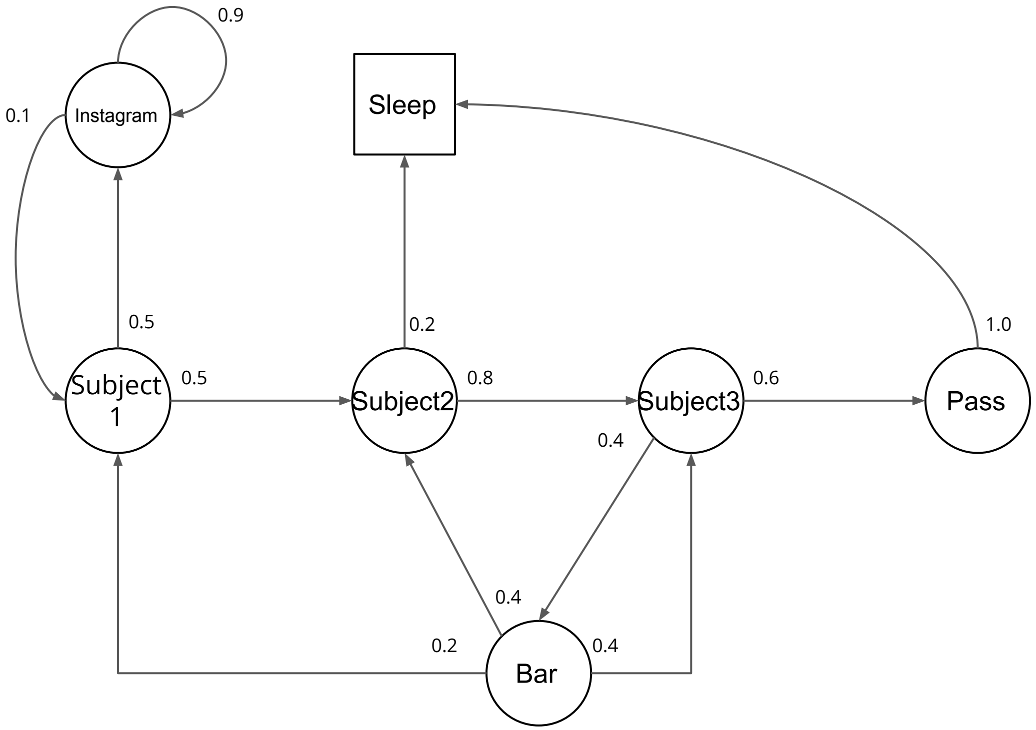
MARKOV PROPERTY

A Markov process is a memoryless random process *(a sequence of random states S_1, S_2 with the Markov property)*

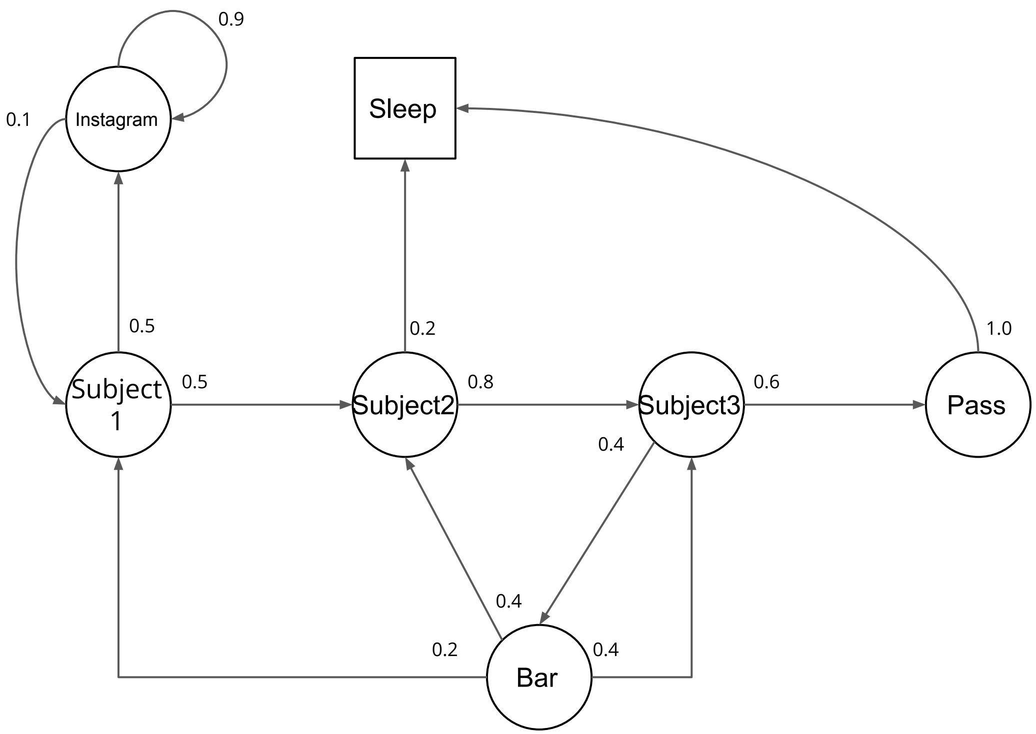
- ***A Markov Process (or Markov Chain) is a tuple <S,P>***
  - *S is a finite set of states*
  - *P is a state transition probability matrix,*

$$P_{s,s'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$$

EXAMPLE: STUDENT MARKOV CHAIN



EXAMPLE: STUDENT MARKOV CHAIN

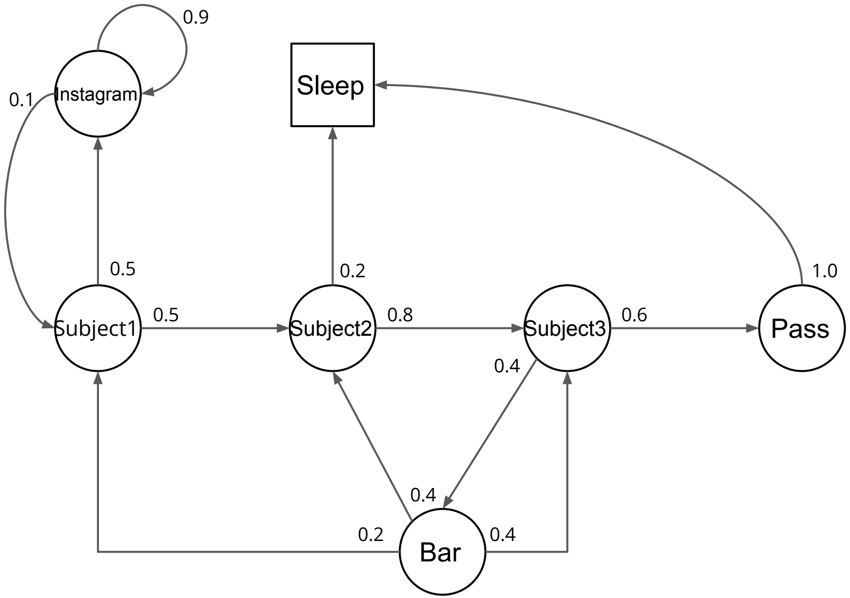


- Sequence of ***episodes*** for Student Markov Chain starting from Subject1

$$S_1, S_2, \ldots, S_T$$

- S1 S2 S3 Pass Sleep
- S1 IN IN S1 S2 Sleep
- S1 S2 S3 Bar S2 S3 Pass Sleep
- S1 IN IN S1 S2 S3 Bar S1 IN IN IN S1 S2 S3 Bar S2 Sleep

EXAMPLE: STUDENT MARKOV CHAIN



	S1	*S2*	*S3*	*Pass*	*Bar*	*IN*	*Sleep*
S1		0.5				0.5	
S2			0.8				0.2
S3				0.6	0.4		
Pass							1.0
Bar	0.2	0.4	0.4				
IN	0.1					0.9	
Sleep							1

MARKOV REWARD PROCESS

- A Markov reward process is a Markov chain with values

- **A Markov Reward process is a tuple <S, P, R, gamma>**
    - **S is a finite set of states**
    - **P is a state transition probability matrix**

$$P_{s',s} = \mathbb{P}[S_{t+1} = s' | S_t = s]$$

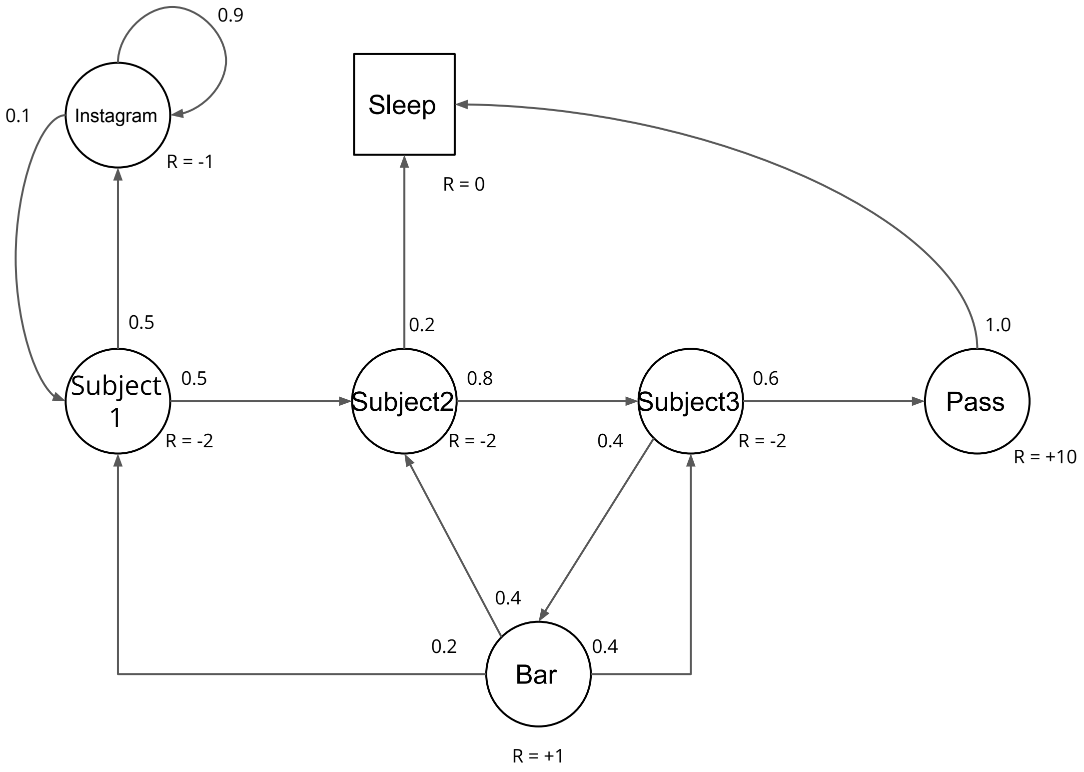
    - **R is a reward function**

$$R_s = \mathbb{E}[R_{t+1} | S_t = s]$$

    - **Gamma is a discount factor**

$$\gamma \in [0, 1]$$

EXAMPLE: STUDENT MARKOV CHAIN



RETURN

- **The return G_t (Goal) is the total discounted reward from time-step t**

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

  - The discount $\gamma \in [0, 1]$ is the present value of the future rewards
  - The value of receiving R after k+1 timetesps is gamma^k r
  - if gamma is close to 0 than we have a myopic evaluation, if is close to 1 is far-sighted evaluation

VALUE FUNCTION

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

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

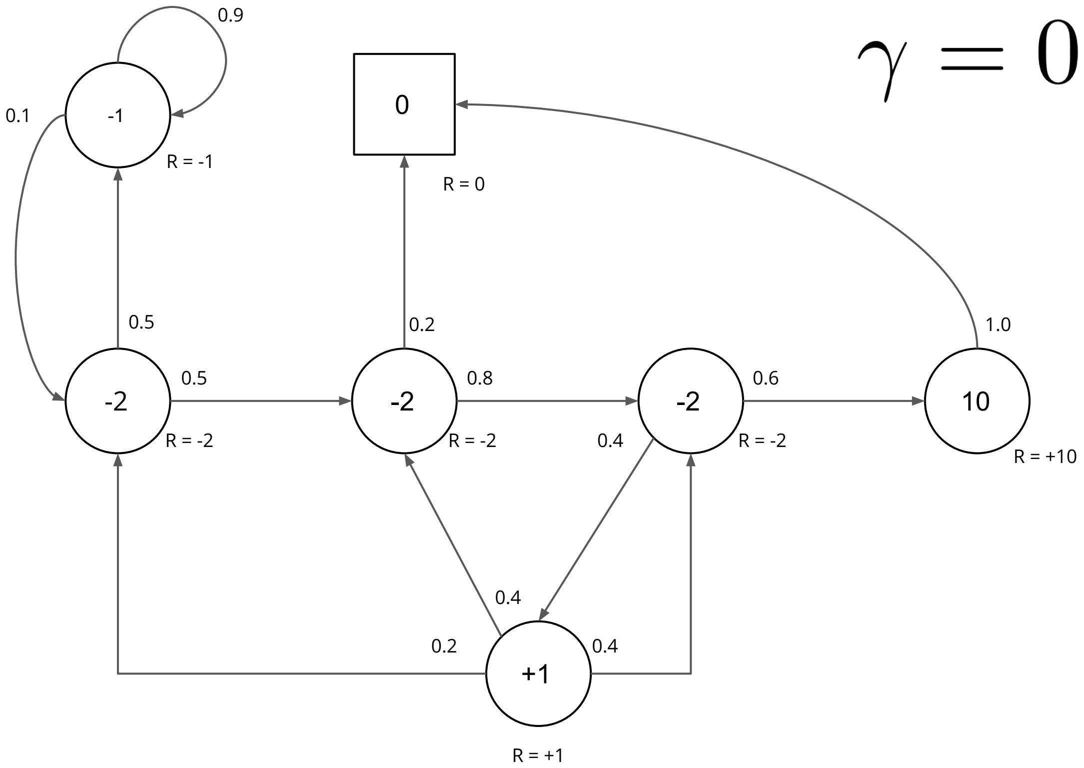
EXAMPLE: STUDENT MRP

- **Starting from S_1 = S1 with gamma = 1/2**

S1 S2 S3 Pass Sleep	$v1 = -2 - 2 * 1/2 - 2 * 1/4 + 10 * 1/8 = -2.25$
S1 IN IN S1 S2 Sleep	$v1 = -2 - 1 * 1/2 - 1 * 1/4 - 2 * 1/8 - 2 * 1/16 = -3.125$
S1 S2 S3 Bar S2 S3 Pass Sleep	$v1 = -2 - 2 * 1/2 - 2 * 1/4 + 1 * 1/8 - 2 * 1/16 \ldots = -3.41$
S1 IN IN S1 S2 S3 Bar	$v1 = -2 - 1 * 1/2 - 1 * 1/4 - 2 * 1/8 - 2 * 1\ /16 \ldots = -3.20$

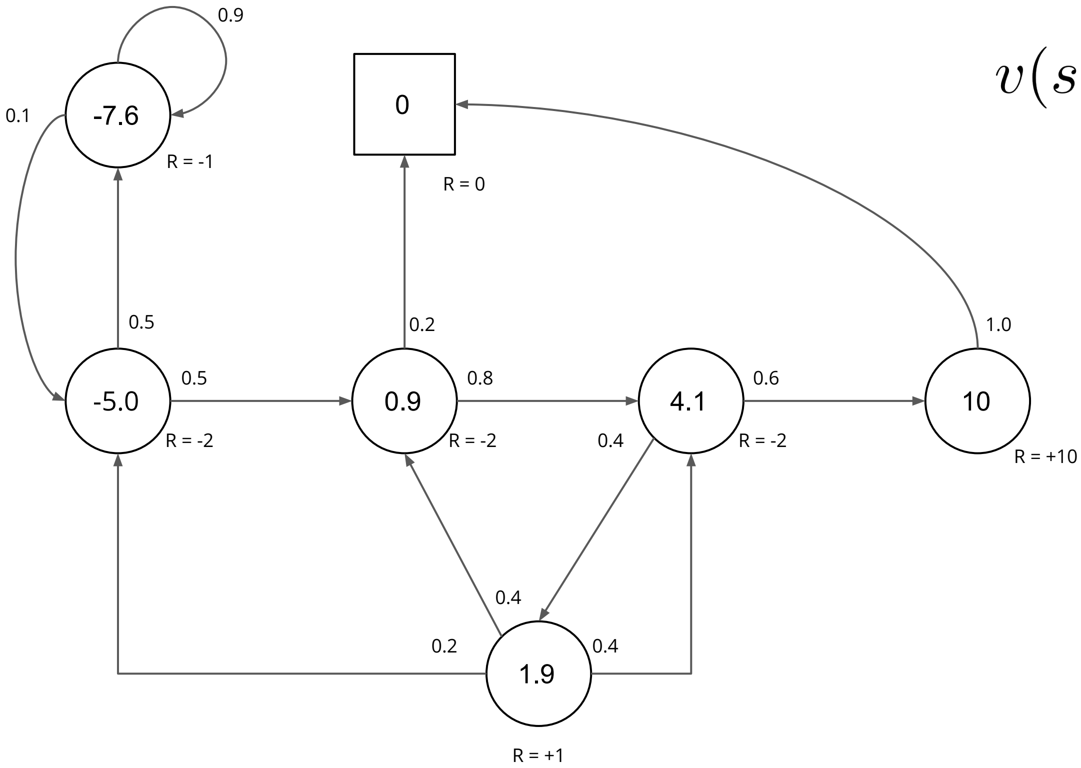
EXAMPLE: STATE-VALUE FUNCTION (1)

$$\gamma = 0$$



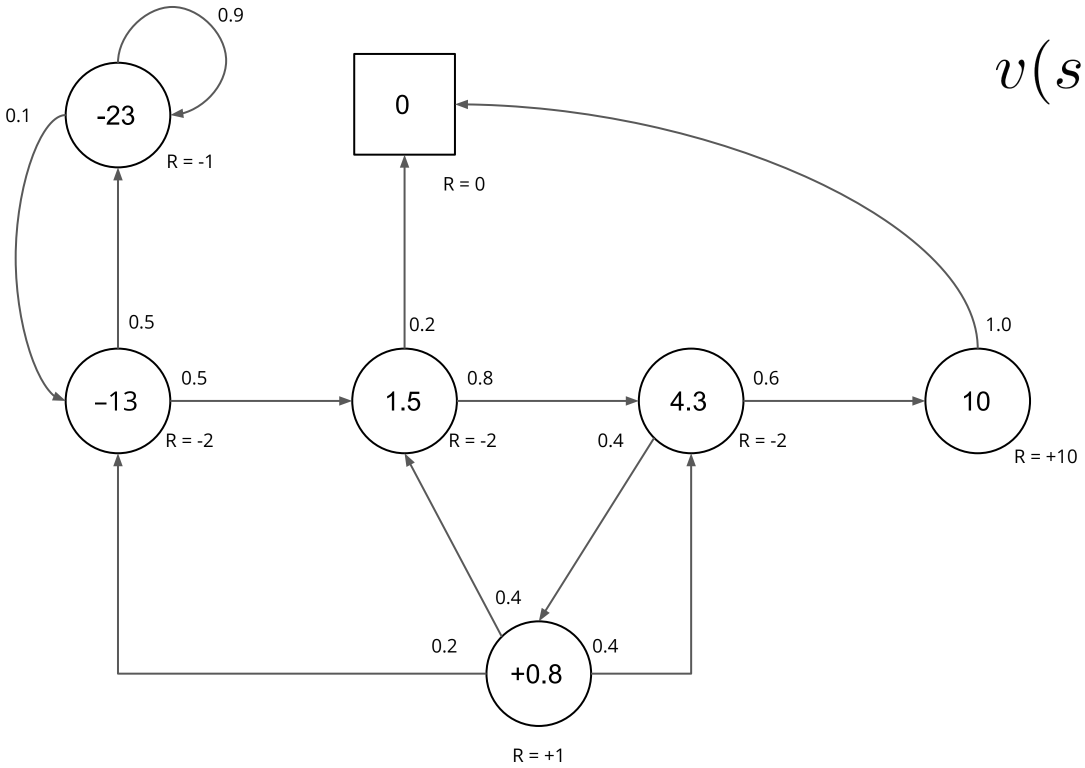
EXAMPLE: STATE-VALUE FUNCTION (2)

$$v(s), \gamma = 0.9$$



EXAMPLE: STATE-VALUE FUNCTION (3)

$$v(s), \gamma = 1$$



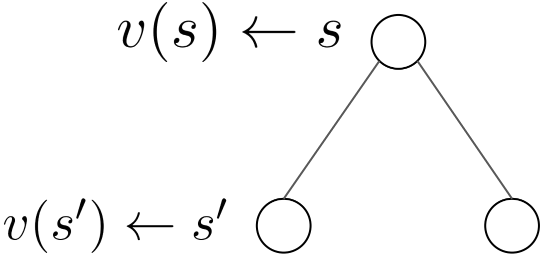
BELLMAN EQUATION

- **The value function can be divided into two parts:**
  - **Immediate reward** R_{t+1}
  - **Discounted value of successor state**

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

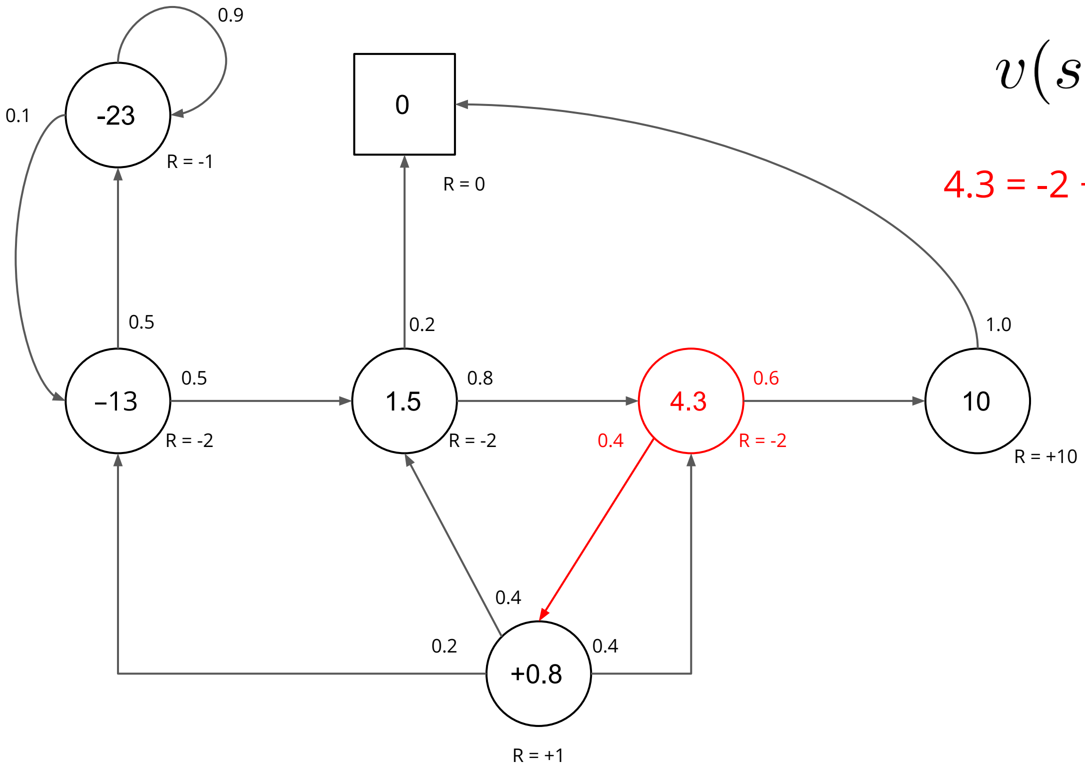
BELLMAN EQUATION

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



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

EXAMPLE: BELLMAN EQUATION



$$v(s), \gamma = 1$$

4.3 = -2 + 0.6 * 10 + 0.4 * 0.8

BELLMAN EQUATION (MATRIX)

- The Bellman equation can be converted into a matrix form

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

Where v is a column vector in which every row is a 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 linear and ca be solved

$$v = R + \gamma P v$$
$$(I - \gamma P)v = R$$
$$v = (I - \gamma P)^{-1} R$$

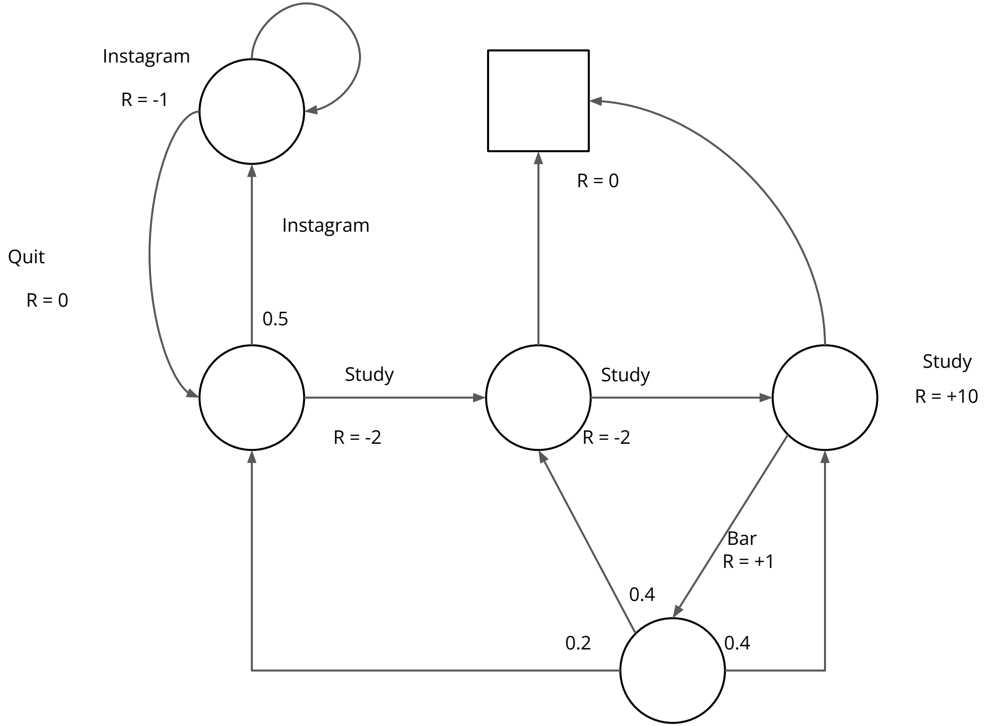
- Direct solution only possible for small MRPs
- There are many iterative methods for large MDPs:
  - Dynamic programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning

MARKOV DECISION PROCESS

- **A Markov Decision process (MDP) is a Markov reward process with decision. All states are Markov in the environment.**

  A Markov decision Process is a tuple $\langle S, A, P, R, \gamma \rangle$
  - *S* is a finite set of states
  - *A* is a finite set of actions
  - *P* is a state transition probability matrix, $P^a_{ss'} = \mathbb{P}[S_{t+1} = s | S_t = s, A_t = a]$
  - *R* reward function, $R^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
  - γ is a discount factor $\gamma \in [0, 1]$

EXAMPLE: STUDENT MDP



POLICIES

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

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

  - A policy defines (in a fully way) the agent's behaviour
  - MDP policies depend on the current state (we throw away the history)
  - The policies are stationary and time-independent

POLICIES

- Given an MDP $M = \langle S, A, P, R, \gamma \rangle$ and a policy π:
  - The state sequence $S_i, S_2 \ldots$ is a Markov process $\langle S, P^\pi \rangle$
  - The state and the reward sequence $S_1, R_2, S_2 \ldots$ is a Markov reward process $\langle S, P^\pi, R^\pi, \gamma \rangle$

    Where:

$$P^\pi_{s,s'} = \sum_{a \in A} \pi(a|s) P^a_{s,s'}$$

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

VALUE FUNCTION

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

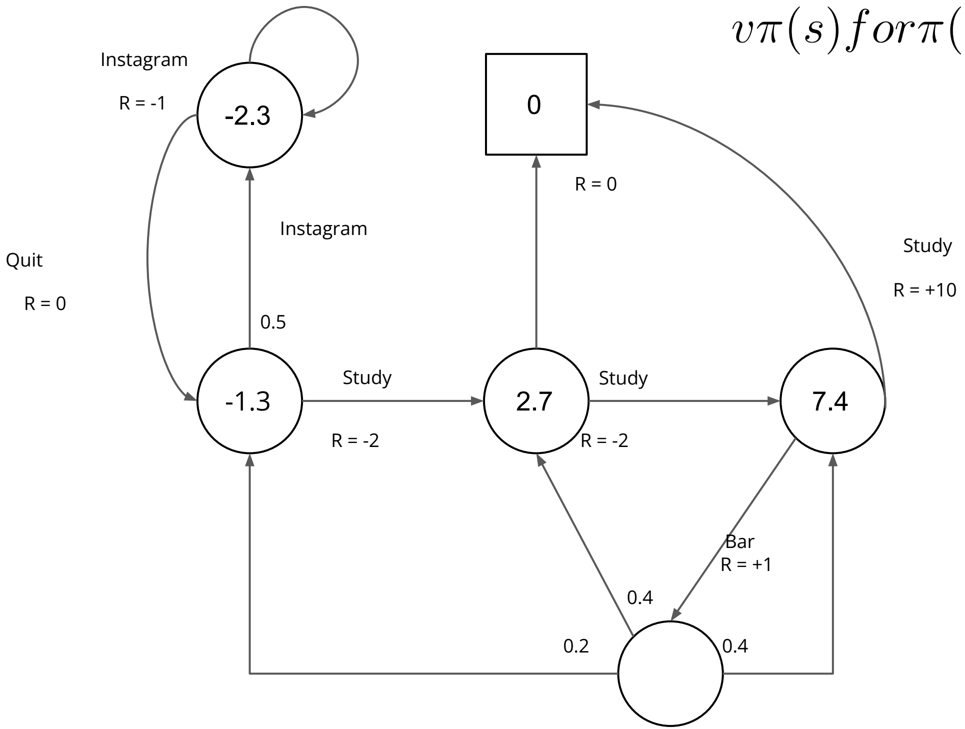
$$v_\pi = \mathbb{E}_\pi[G_t | S_t = s]$$

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

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

EXAMPLE: STUDENT MDP

$$v\pi(s) for \pi(a|s) = 0.5, \gamma = 1$$



BELLMAN EXPECTATION EQUATION

- The state-value function can be split into immediate reward plus the discounted value of successor state

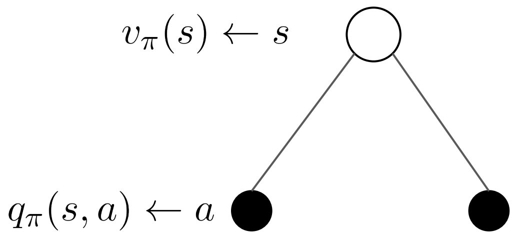
$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$$

- The action-value function can similarly be splitted

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

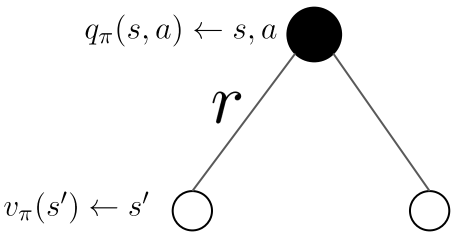
BELLMAN EXPECTATION EQUATION

$$V_\pi$$



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

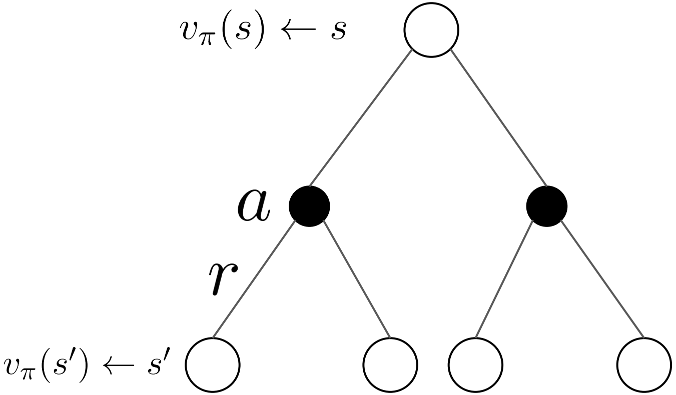
$$Q_\pi$$



$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s')$$

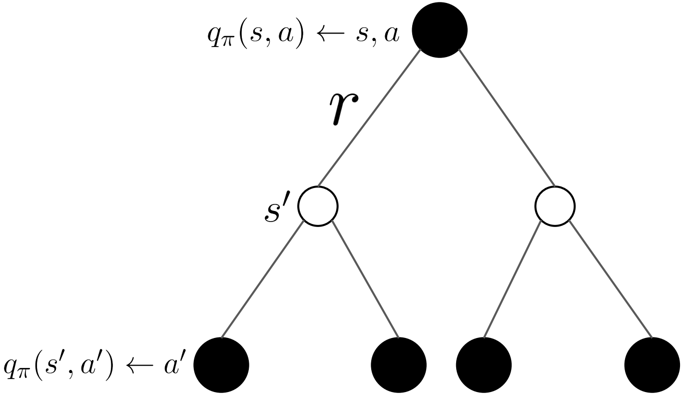
BELLMAN EXPECTATION EQUATION – 2

V_π



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

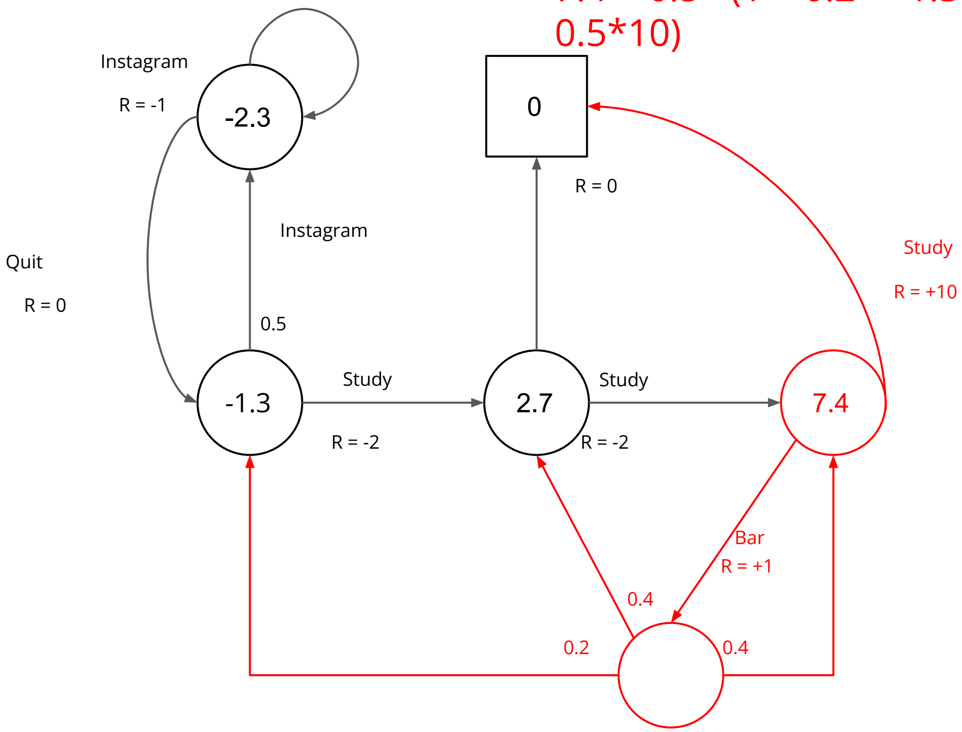
Q_π



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

EXAMPLE: STUDENT MDP

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



OPTIMAL VALUE FUNCTION

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

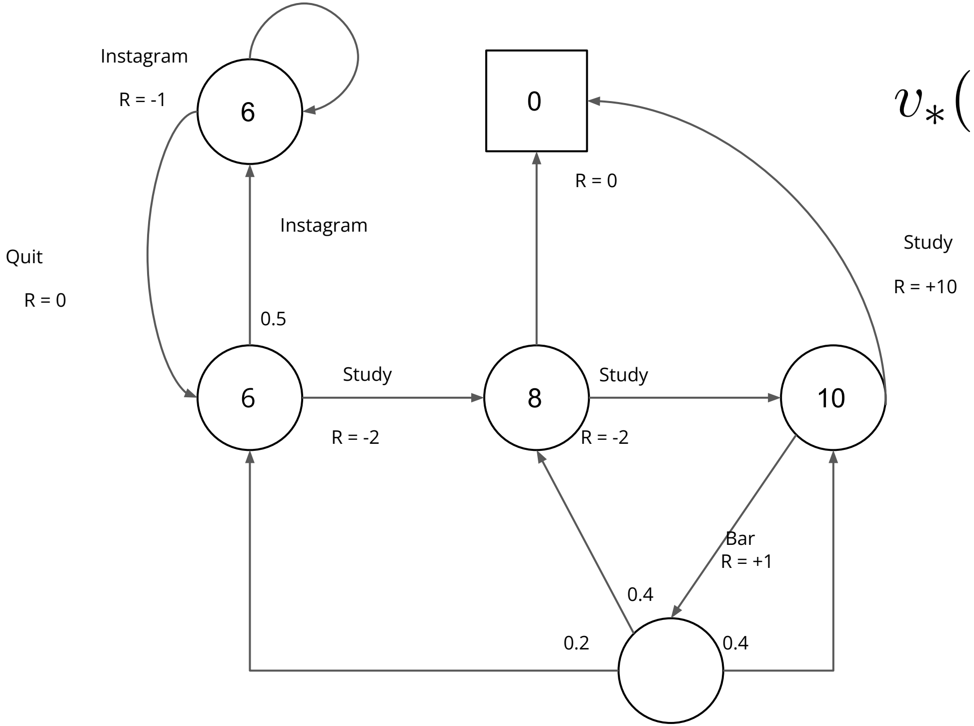
$$v_*(s) = \max_\pi v_\pi(s)$$

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

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

- This function specifies the best possible performance in the MDP.
- An MDP is “solved” when we discover the optimal value

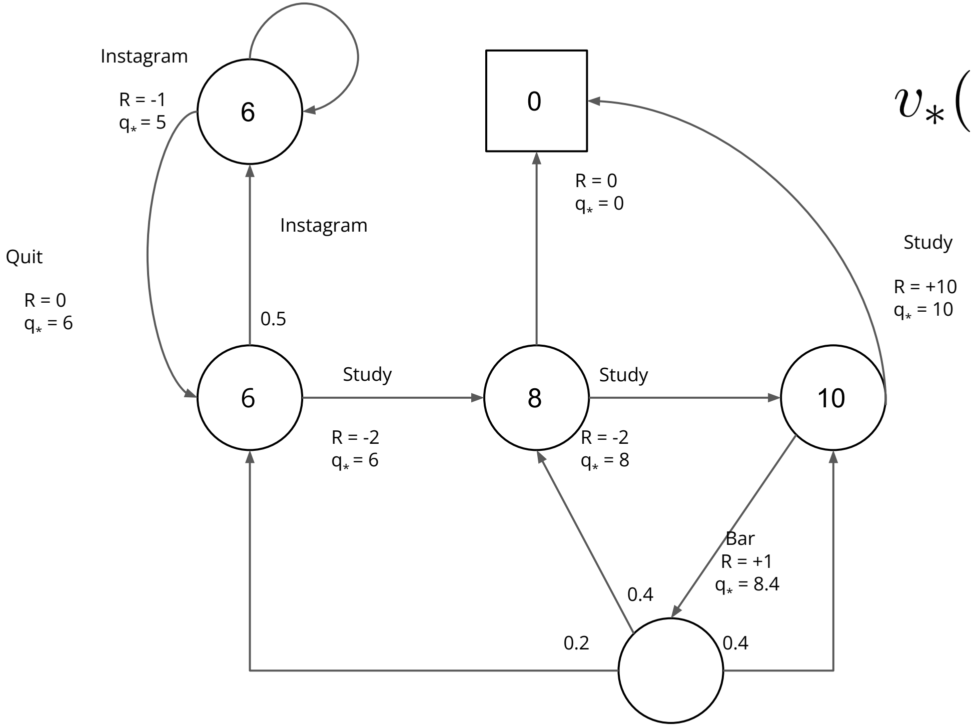
EXAMPLE: STUDENT MDP (optimal value function)



$$v_*(s)\ for\ \gamma = 1$$

EXAMPLE: STUDENT MDP (optimal action-value function)

$$v_*(s) \; for \; \gamma = 1$$



OPTIMAL POLICY

- For any Markov Decision Process
  - There is 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, $v_{\pi_*} = v_*(s)$
  - All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s, a) = q_*(s, a)$

OPTIMAL POLICY

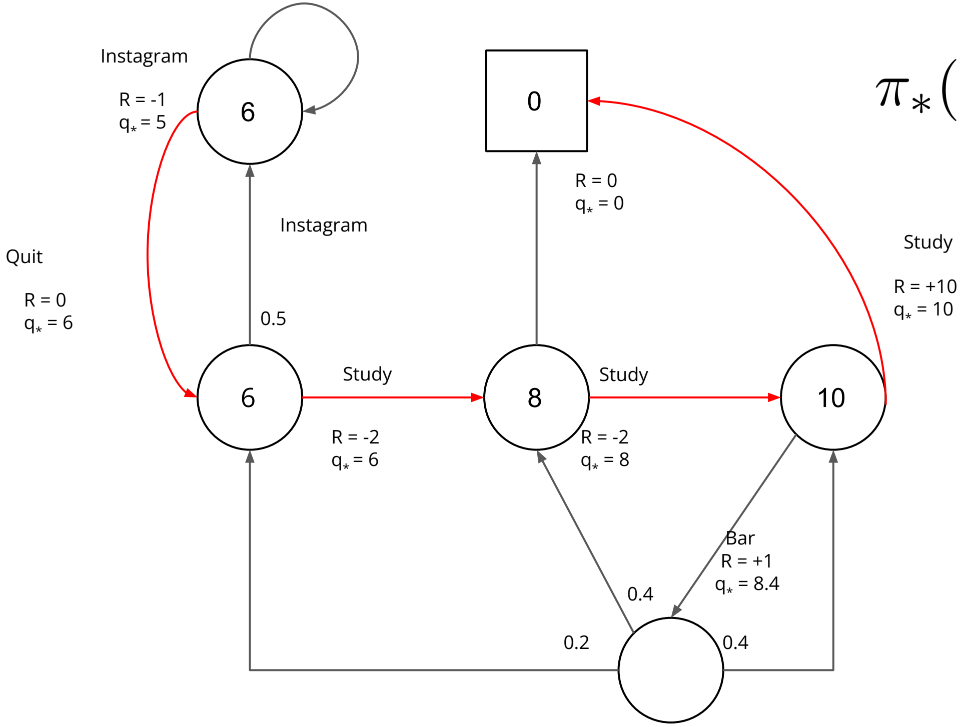
- An optimal policy can be found maximizing over $q_*(s,a)$

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

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

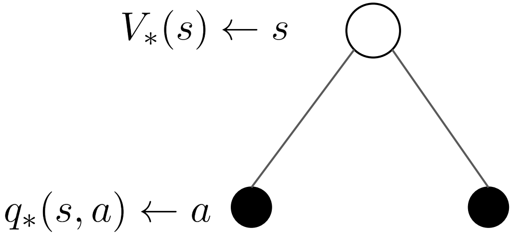
EXAMPLE: STUDENT MDP (optimal policy)

$$\pi_*(a|s)\ for\ \gamma = 1$$



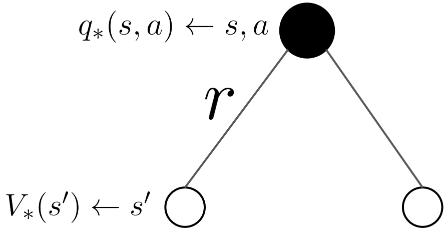
BELLMAN OPTIMALITY EQUATION

$$V_*$$



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

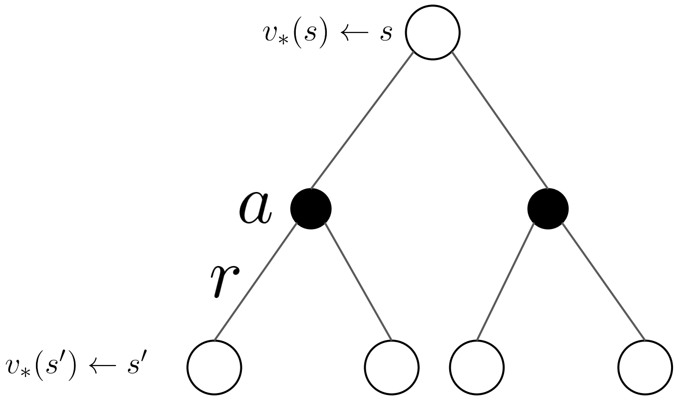
$$Q_*$$



$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$$

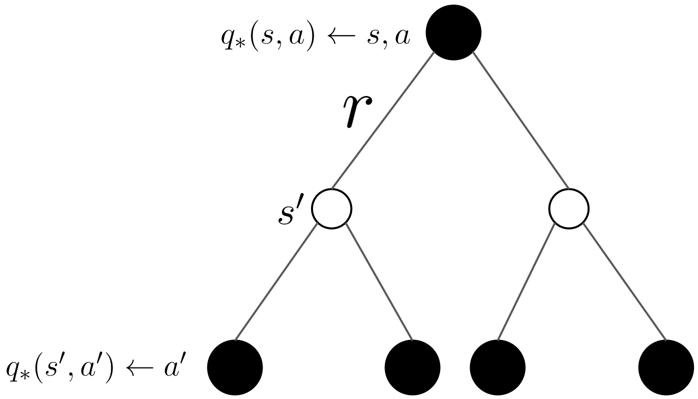
BELLMAN OPTIMALITY EQUATION – 2

V_*



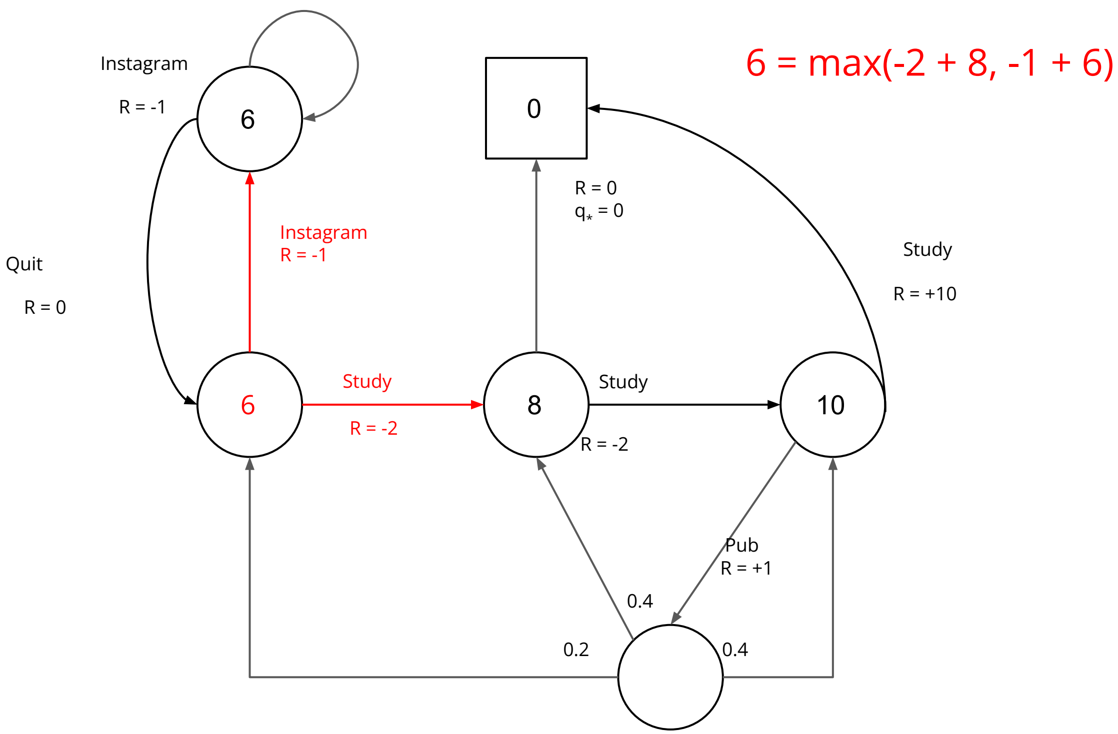
$$v_*(s) = \max_a R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$$

Q_*



$$q_*(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_a q_*(s', a')$$

EXAMPLE: STUDENT MDP (optimal policy)

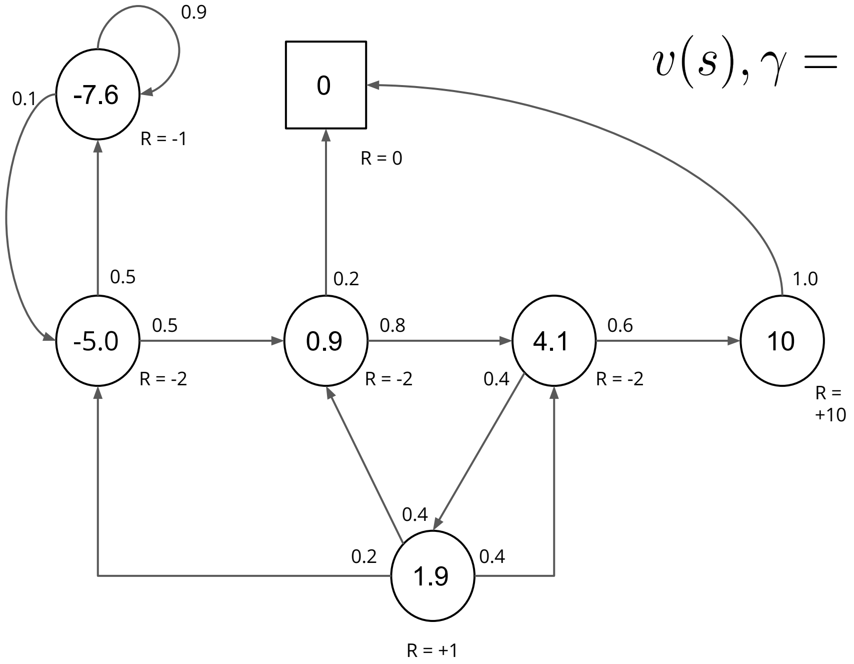


SOLVING BELLMAN OPTIMALITY EQ

- The Bellman Optimality Equation
  - Is not linear
  - No closed form solution
  - Can be solved with many iterative solution methods:
    - Value Iteration
    - Policy Iteration
    - Q-learning
    - Sarsa

ASSIGNMENT: BUILDING A STUDENT DECISION-MAKING PROBLEM

- **Objective**: **build** a **graph** representing a **student decision-making problem**, **compute value functions** using the **Bellman equation**, and **visualize** the **graph with** annotated **value functions**.



$$v(s), \gamma = 0.9$$

ASSIGNMENT: BUILDING A STUDENT DECISION-MAKING PROBLEM

- **1 - Building the graph:**
  - Import necessary libraries: networkx, matplotlib.pyplot, numpy, seaborn, and pyvis.network.
  - Define the states and rewards for each state.
  - Define the transition probabilities between states.
  - Create a directed graph using nx.DiGraph().
  - Add nodes to the graph for each state.
  - Add edges to represent transitions between states, including associated actions and rewards.

ASSIGNMENT: BUILDING A STUDENT DECISION-MAKING PROBLEM

- **2 - Visualizing Transition Probabilities:**
  - Compute the transition matrix based on the defined transition probabilities.
  - Plot the transition matrix using seaborn.heatmap().

- **3 - Computing Value Functions:**
  - Define the discount factor.
  - Use the Bellman equation to compute the value function of each state.

ASSIGNMENT: BUILDING A STUDENT DECISION-MAKING PROBLEM

- **4 - Visualizing the Graph with Value Functions:**
  - Draw the graph using nx.draw() with appropriate node positions.
  - Annotate each node with its corresponding value function.
  - Annotate each edge with associated actions and rewards.
- **5 - Optional: Visualizing Value Function Dynamics:**
  - Use matplotlib to visualize how the value function changes with different discount factors.

ASSIGNMENT: BUILDING A STUDENT DECISION-MAKING PROBLEM

- **Submission:**
  - Submit the Python script containing the code to construct the graph, compute value functions, and visualize the graph with annotated value functions.
  - https://shorturl.at/gjyO7 - Expiration date: 25th May 2024
- **Note:** Ensure the code is well-commented and understandable. Avoid using external guidance or assistance, as the purpose of the assignment is to demonstrate understanding and implementation independently.