Econ 425 Week 10

Reinforcement learning

Grigory Franguridi

UCLA Econ

USC CESR

franguri@usc.edu

Motivation

- problem: every morning, choose a coffee place
- if you **stick to one coffee place** (exploitation), you are missing out on the coffee served by competitors
- however, if you **try different coffee places one by one** (exploration), the probability of encountering the worst coffee of your life is high!
- on the other hand, chances to find better coffee

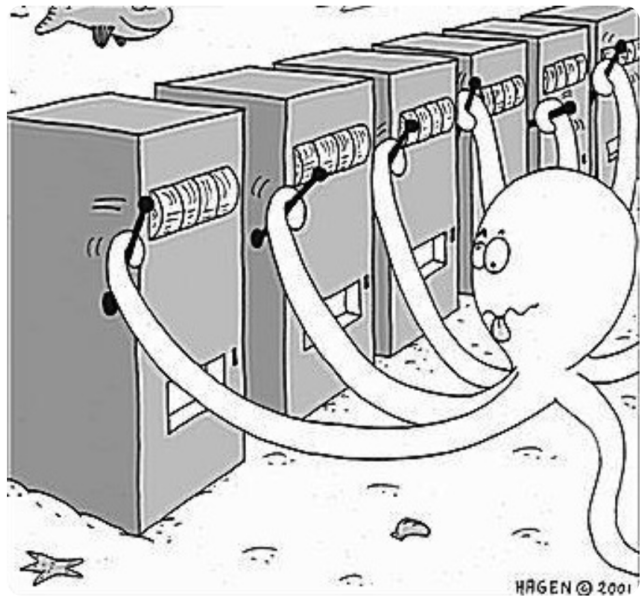
Motivation

- dilemma arises from **incomplete information**: we need to gather enough information by exploring new actions to formulate the best overall strategy
- eventually leads to minimizing the overall bad experiences
- **multi-armed bandit** is a mathematical representation
  - used in online advertising, e-commerce, A/B testing, healthcare, finance, etc.

Multi-armed bandit problem (MABP)

- bandit is someone who steals your money
- 1-armed bandit is a slot machine wherein you insert a coin, pull a lever, and get a reward
  - all casinos configure these slot machines in such a way that all gamblers end up losing money
- multi-armed bandit is a slot machine
  with several levers with different rewards
- not only reward is random, but **even distribution of reward is unknown** to gambler

Multi-armed bandit problem (MABP)



- **goal**: identify which lever to pull to get maximum reward after a given set of trials
- choosing an arm = *action* $\rightarrow$ *reward*
- mathematically, a single-step Markov decision process

Bernoulli MABP

- Bernoulli: reward $\in \{0, 1\}$
- sample results for a 5-armed Bernoulli bandit:

Arm	Reward
1	0
2	0
3	1
4	1
5	0
3	1
3	1
2	0
1	1
4	0
2	0

Bernoulli MABP

- looks like arm 3 gives the maximum return and hence one idea is to keep playing this arm to obtain the maximum reward (pure exploitation)
- arm 5 might look like a bad arm to play, but we have played this arm only once; maybe should play it a few more times (exploration) to gain info
- only then decide which arm to play (exploitation)

MABP: applications

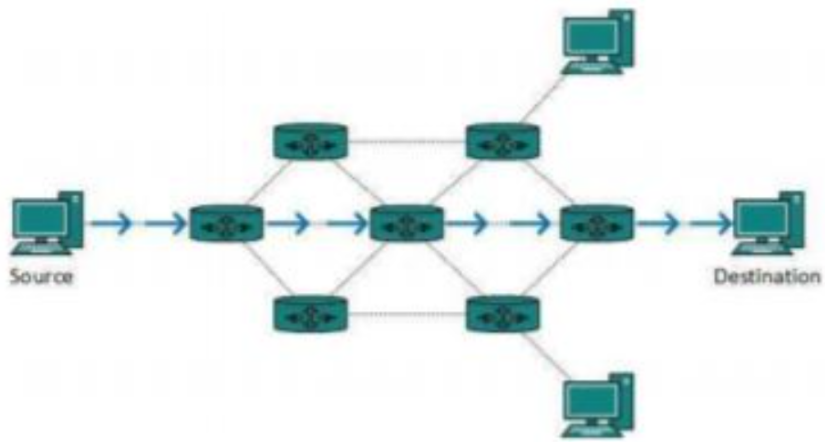
- **Online advertising** maximize ad revenue
- advertiser makes revenue every time an ad is clicked
- exploration (collect information on ad's performance using click-through rates) vs exploitation (stick with ad that has performed best so far)

MABP: applications

Clinical trials: the well-being of patients is as important as the results of the study
Exploration = identifying the best treatment
Exploitation = treating patients most effectively during the trial

MABP: applications

- **Network routing**: process of selecting a path for traffic in a network (telephone/computer networks, internet)
- allocation of channels to users maximizing the overall throughput can be formulated as a MABP



MABP: applications

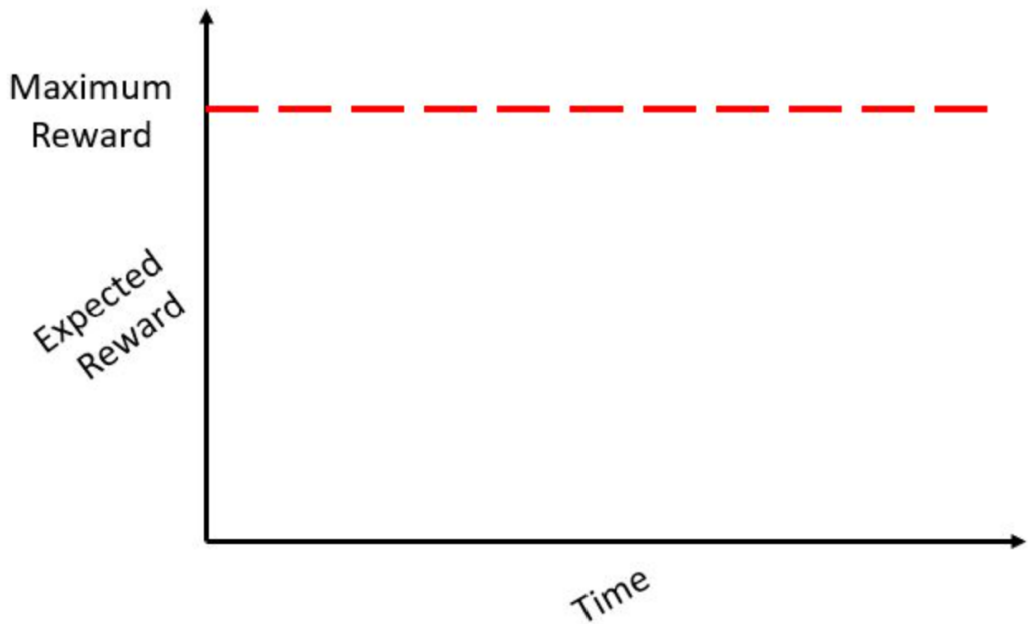
Game design: test experimental changes in gameplay/UI and exploit the changes delivering positive experiences for players

Solution strategies

- **value function**: $Q_t(a) = \mathbb{E}[r|a]$, where r is reward, a is action, t is time
- $Q_t(a_k) = p_k$, where $p_1, \ldots, p_K$ are the reward probabilities for a K-armed bandit
- how to evaluate a given strategy?
- one way is to compare the total/average reward after n trials
- another way is **regret**

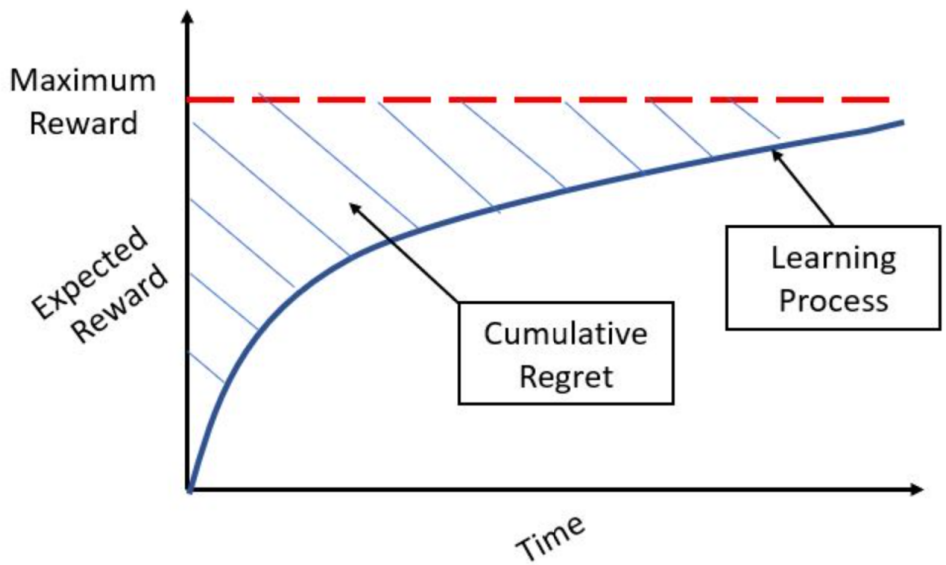
Regret

if pull the best arm repeatedly, get a maximum *expected* reward (horizontal line):



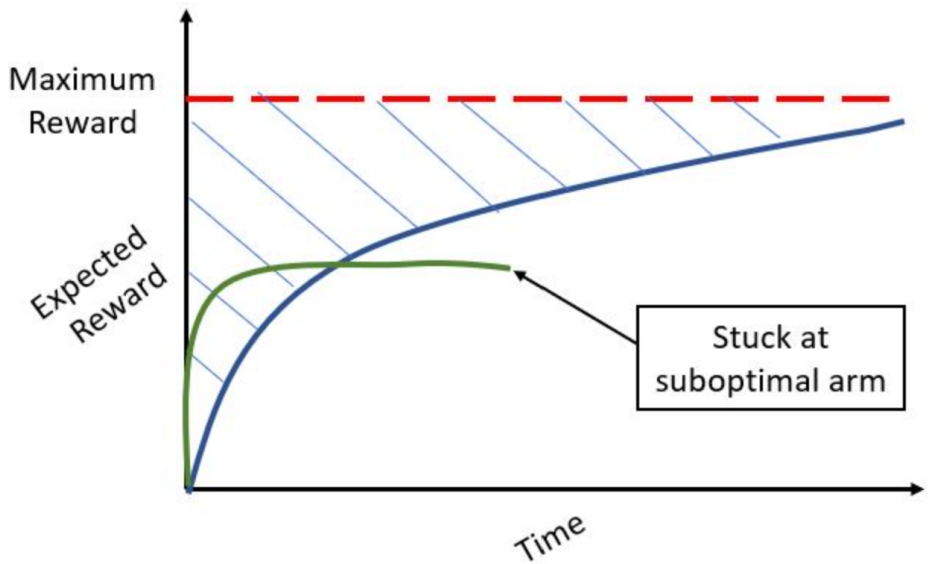
Regret

- in the real world, unsure about which arm is best
- loss incurred during exploitation is called **regret** (from not choosing the best arm)



Regret

- how does regret change if not enough exploration / too much exploitation of a suboptimal arm?



No exploration (greedy approach)

- naive approach: choose the best arm so far

$$Q_t(a) = \frac{\sum_{i=1}^{t} 1\{a_t = a\} \cdot r_i}{\sum_{i=1}^{t} 1\{a_t = a\}}$$

$$a^* = \text{argmax}_{a \in \mathcal{A}} Q_t(a)$$

No exploration (greedy approach)

- need the entire history of rewards
- avoid this by using running sum:

$$Q_t(a) = \frac{Q_{t-1}(a)N_t(a_t) + R_t \cdot 1\{a_t = a\}}{N_t(a_t)}$$

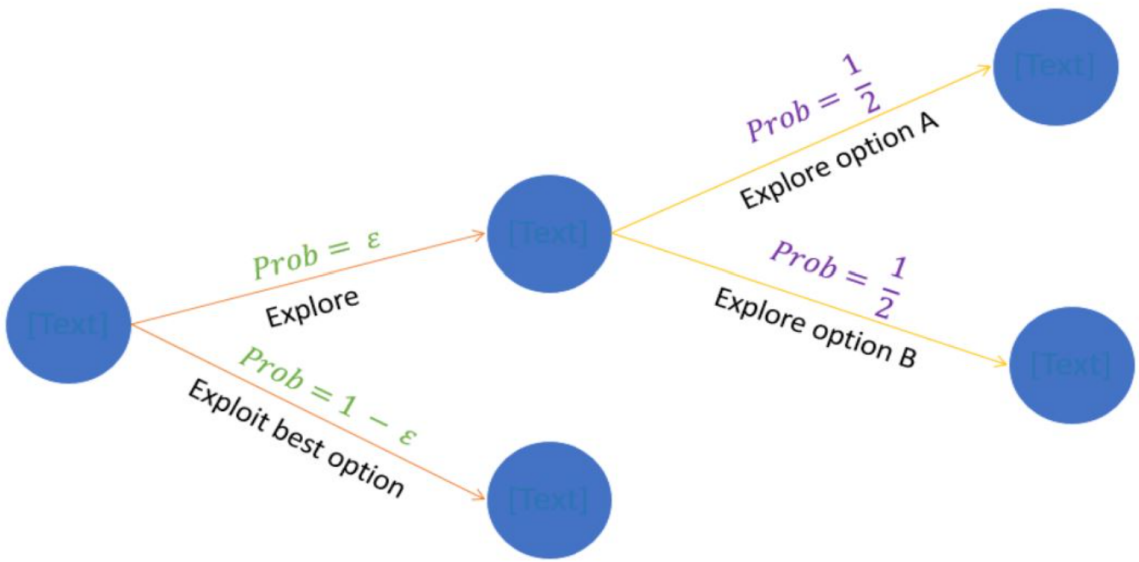
and

$$Q_t(a) = Q_{t-1}(a) + \frac{1}{N_t(a_t)(R_t - Q_{t-1}(a))}$$

- this approach **never explores**, as it always picks the same arm

Epsilon-greedy approach

- w. probability ε, choose a random action (exploration)
- w. probability 1-ϵ, choose an action maximizing $Q_t(a)$
- example w. two actions A and B

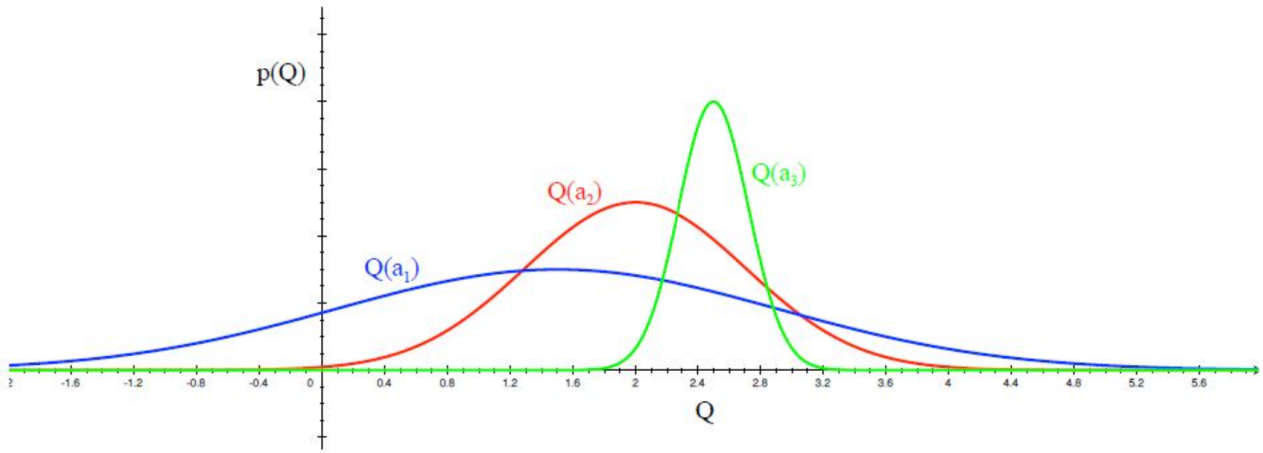


Upper Confidence Bound

- the most widely used solution method for MABP
- based on the principle of **optimism in the face of uncertainty**: the more uncertainty about an arm, the more important to explore it

Upper Confidence Bound

Distribution of reward for 3 arms after a few trials:



- reward for a_1 has the highest variance (maximal uncertainty)
- UCB:
  - choose a_1 and receive a reward
  - if still uncertain about a_1, choose it again until the uncertainty is below a threshold

Upper Confidence Bound

Intuition: at each time, two cases

1. **optimism is justified**: get a positive reward
2. **optimism is not justified**: play an arm that we falsely believe gives a large reward.
    - if this happens sufficiently often, then we learn the payoff distribution and stop choosing this arm in the future

Upper Confidence Bound

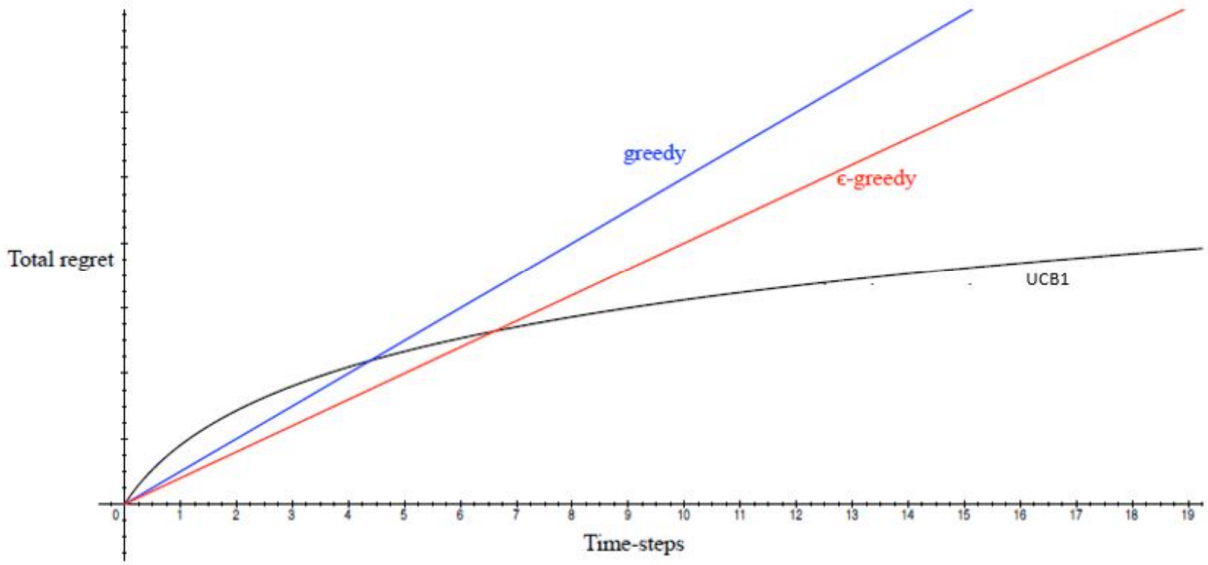
- UCB is a family of algorithms, focus on **UCB1**:
  1. for $t = 1, \ldots, K$: play arm t to obtain initial value of $Q(a_t)$
  2. for $t > K + 1$: play

$$a^* = \arg\max_{a \in \mathcal{A}} \left\{ Q_t(a) + \sqrt{\frac{2 \log t}{N_t(a)}} \right\},$$

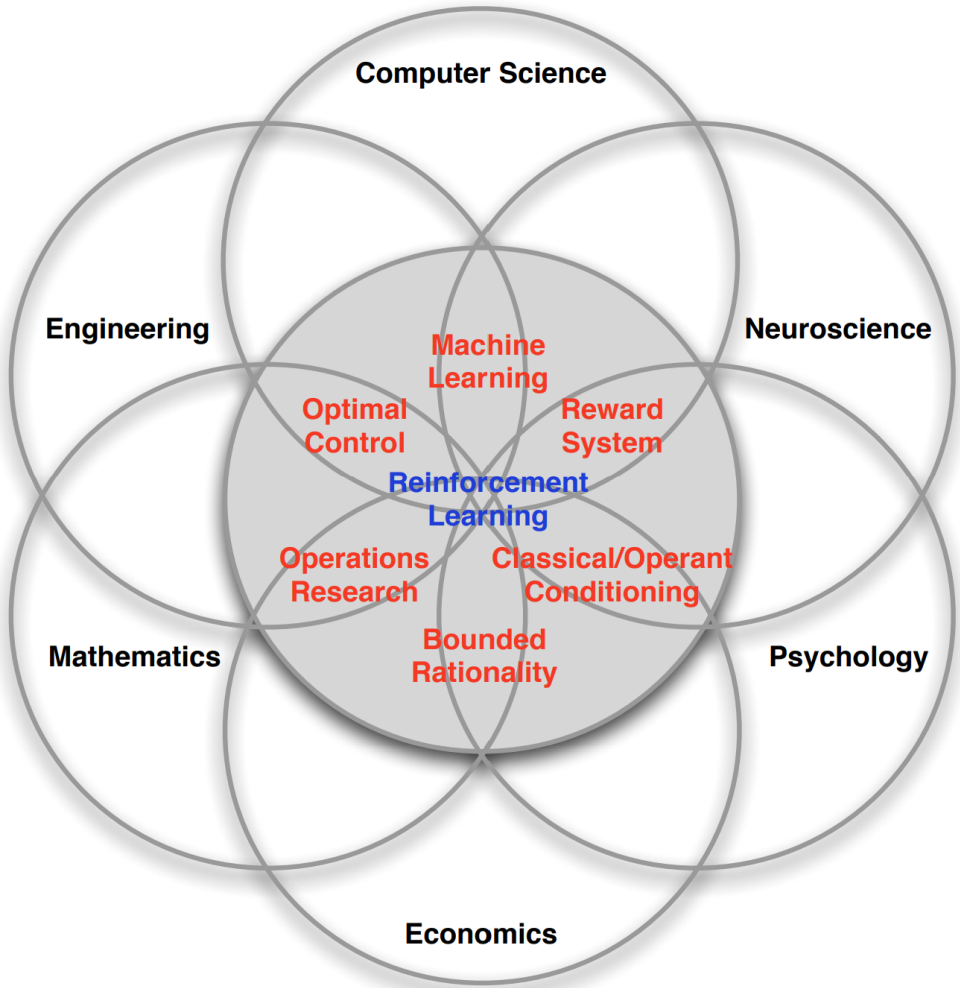
  where $N_t(a)$ is the number of times action a was played so far, and update $Q_t(a^*)$

Regret comparison

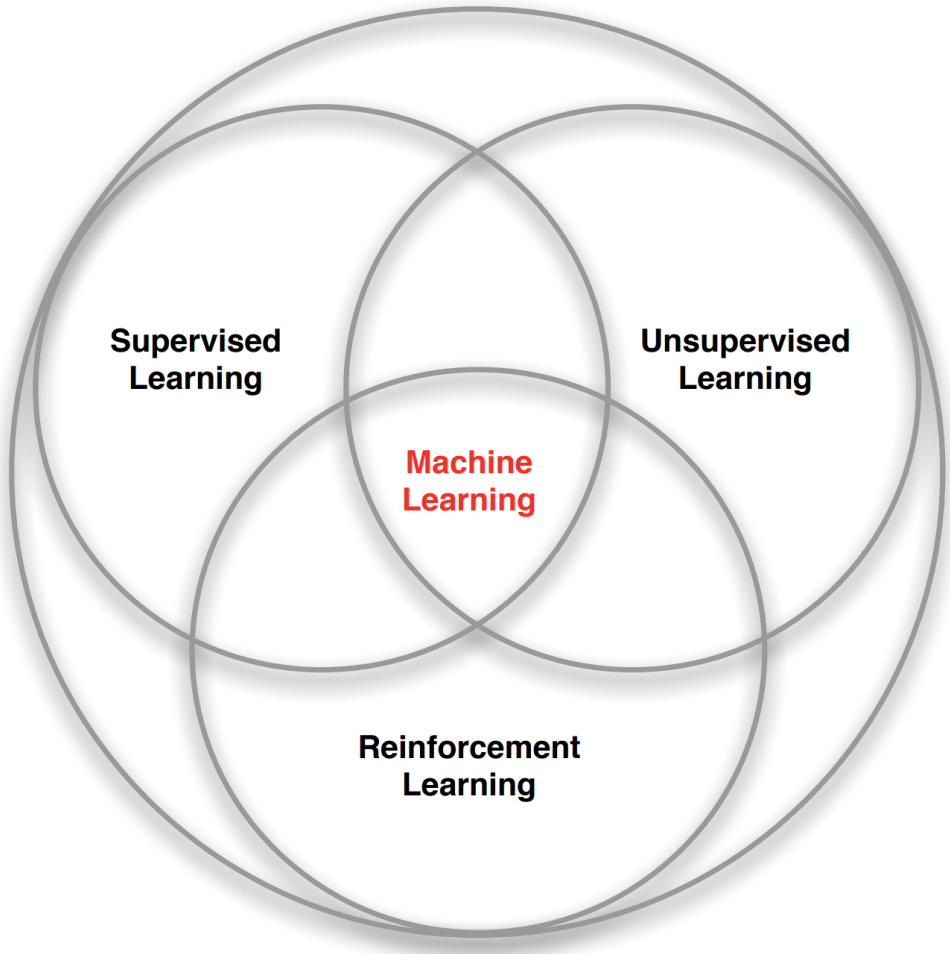
- among the algorithms discussed so far, only the UCB algorithm has regret increasing as $\log(t)$
- other algorithms have linear regret with different slopes



Many faces of reinforcement learning



Branches of machine learning



Reinforcement learning

- key features:
  - no supervisor, only reward signal
  - feedback is delayed, not instantaneous
  - time matters (sequential, non i.i.d data)
  - agent's actions affect subsequent data
- multi-armed bandit is a special example of RL without state space (to be introduced later)

Reward

- Reward R_t at step t is a scalar feedback signal
- goal: maximize cumulative reward

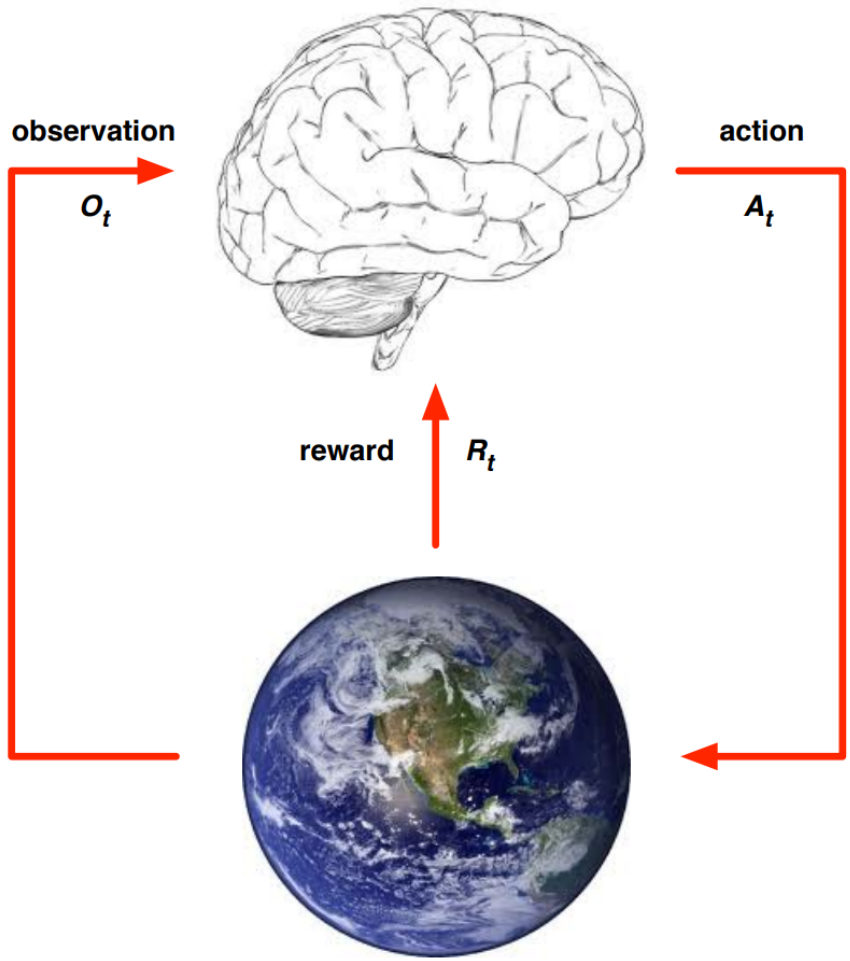
Reward: examples

- Piloting a helicopter
  - positive/negative reward for following/deviating from the desired trajectory
- Playing Go
  - positive/negative reward for winning/losing a game
- Controlling a power station
  - positive/negative reward for producing power / exceeding safety thresholds

Sequential decision making

- Goal: select actions to maximize total future reward
- Actions may have long-term consequences
- Reward may be delayed
- May be optimal to sacrifice immediate reward to gain long-term reward
- Examples:
  - Financial investment (may take months to mature)
  - Refuelling a helicopter (may prevent a crash in several hours)
  - Blocking opponent moves (may increase winning chances many moves from now)

Agent and environment



Agent and environment

- At each step t the agent:
  - Executes action A_t
  - Receives observation O_t
  - Receives scalar reward R_t
- The environment:
  - Receives action A_t
  - Emits observation O_{t+1}
  - Emits scalar reward R_{t+1}

Markov decision process

- in stats terms, RL is **Markov decision processes (MDP)**
- the environment is fully observable and the current state completely characterizes the process
- almost all RL problems can be formalized as MDPs:
  - optimal control primarily deals with continuous MDPs
  - partially observable problems can be converted into MDPs
  - bandits are MDPs with one state

Markov property

- the process (S_t) is Markov if

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \cdots, S_t]$$

  (current state captures all relevant information from history)
- **the future is independent of the past given the present"**

State transition matrix

- state transition probability is

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

- state transition matrix $\mathcal{P}$ is

$$\mathcal{P} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{bmatrix}$$

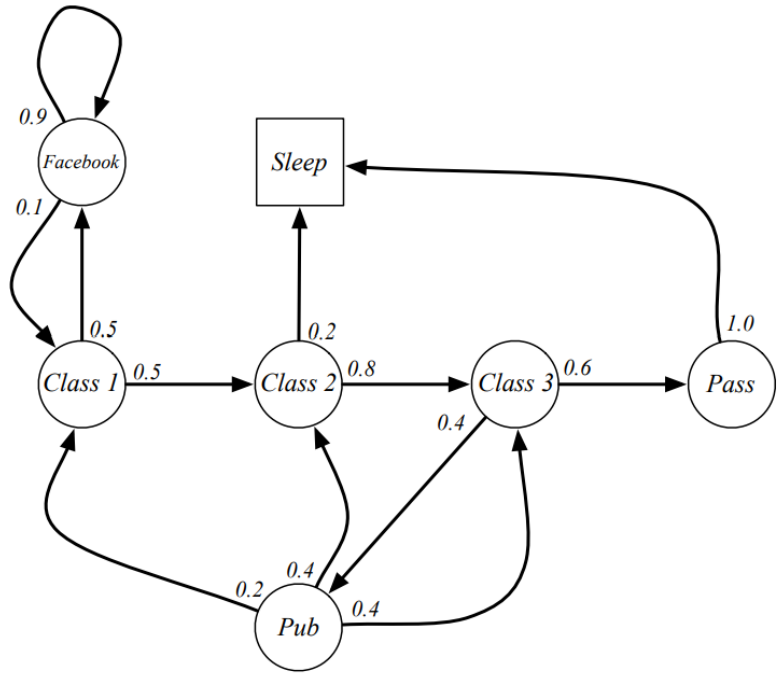
  where each row of the matrix sums to 1

Markov Process

- Markov process is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$, where
  - $\mathcal{S}$ is a set of states
  - $\mathcal{P}$ is a state transition probability matrix

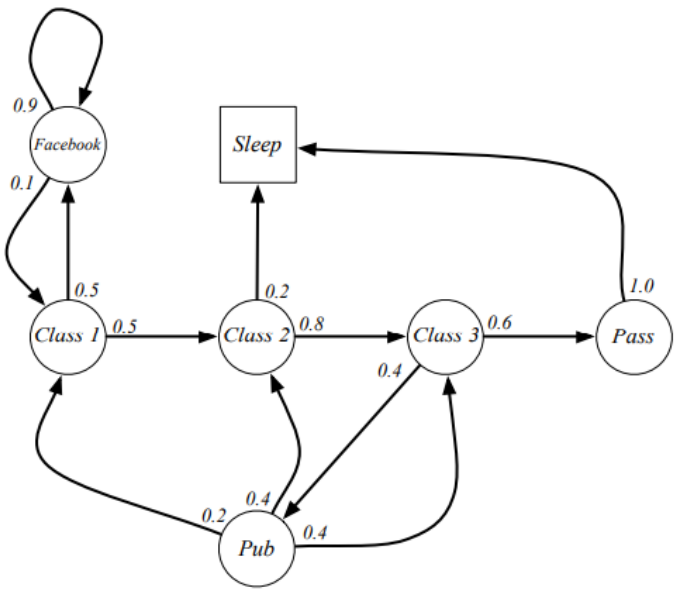
$$p_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$$

Example: student's Markov chain



- sample paths:
- 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

Example: student's Markov chain



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

Markov reward process

Markov chain with values: tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where

- $\mathcal{S}$ is a finite set of states
- $\mathcal{P}$ is a state transition probability matrix,

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

- $\mathcal{R} : \mathcal{S} \to \mathbb{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}[R_{t+1} | S_t = s]$
- $\gamma \in [0, 1]$ is a discount factor

Markov reward process

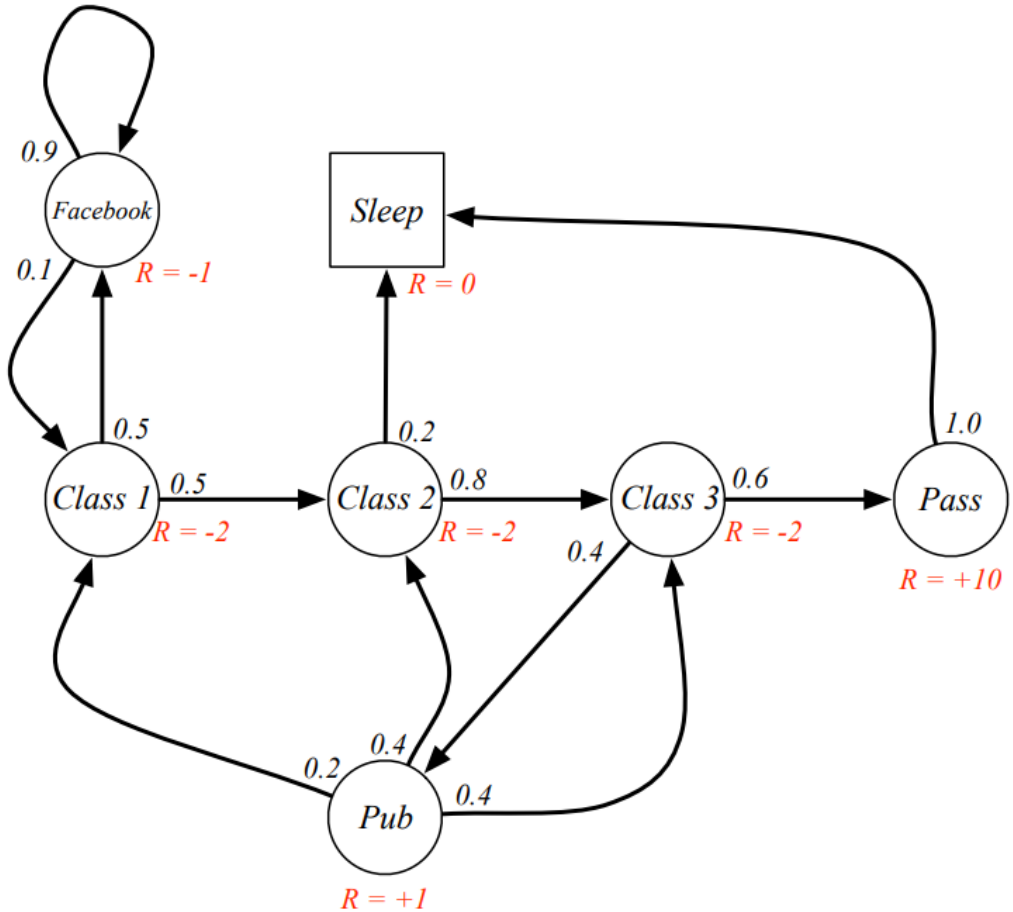
Markov chain with values: tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where

- $\mathcal{S}$ is a finite set of states
- $\mathcal{P}$ is a state transition probability matrix,

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

- $\mathcal{R} : \mathcal{S} \to \mathbb{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}[R_{t+1} | S_t = s]$
- $\gamma \in [0, 1]$ is a discount factor

Example: student's MRP



Return

- return G_t is the total discounted reward from step t:

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

- discount factor $\gamma \in [0, 1]$ yields present value of future rewards (immediate reward is valued above delayed reward)
  - γ close to 0: "myopic" evaluation
  - γ close to 1: "far-sighted" evaluation
- value of receiving reward R after $k + 1$ time-steps is $\gamma^k R$

Markov decision process

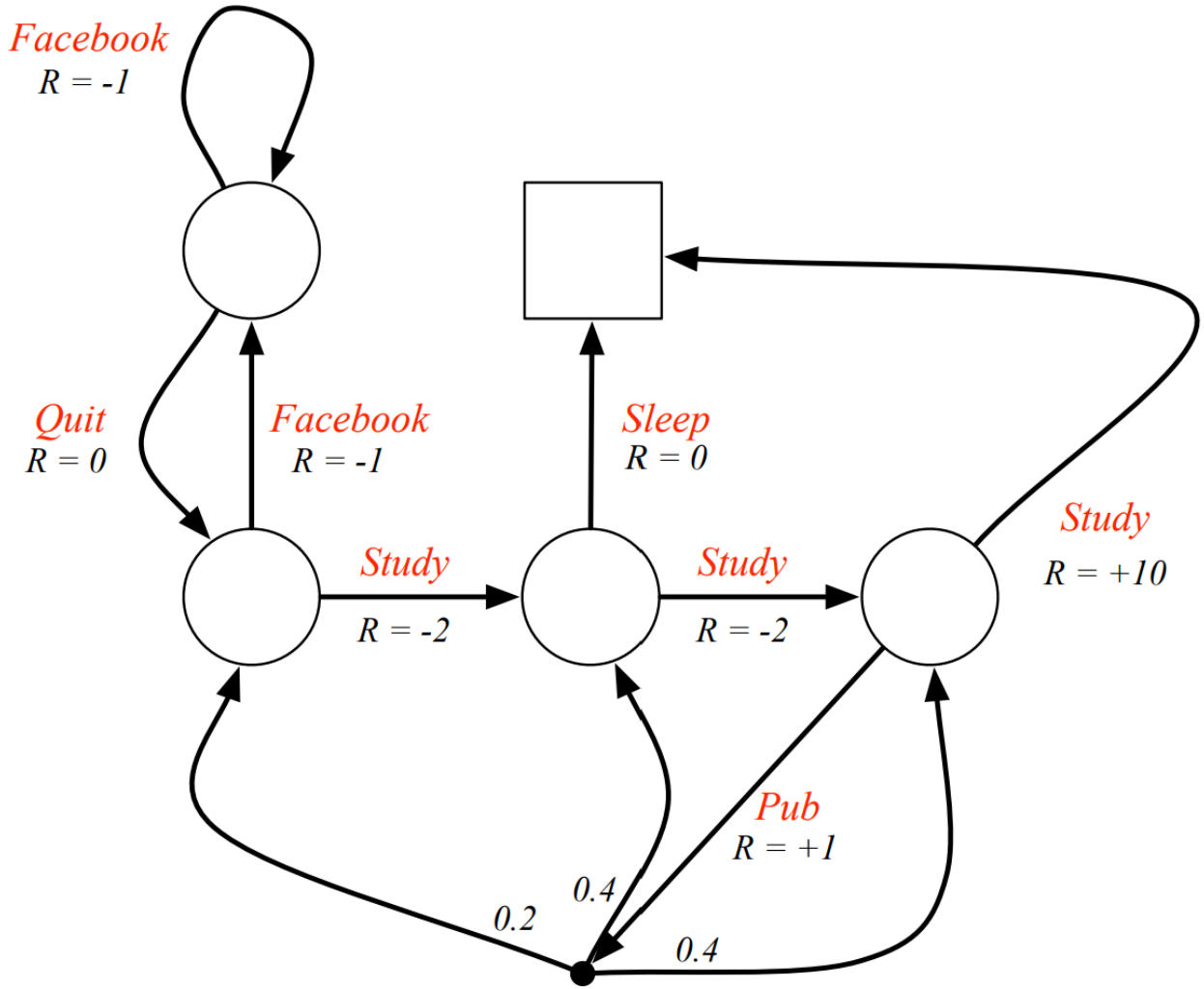
MDP is MRP with **actions**: a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix,

$$p^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

- $\mathcal{R}$ is a reward function, $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
- $\gamma \in [0, 1]$ is a discount factor

Example: student MDP



Policy function

policy π is a distribution over actions given states

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

- fully describes agent's behavior
- depends on the current state only (not previous states)
- stationary (time-independent), i.e. $A_t \sim \pi(\cdot|S_t)$, $\forall t$

Value function

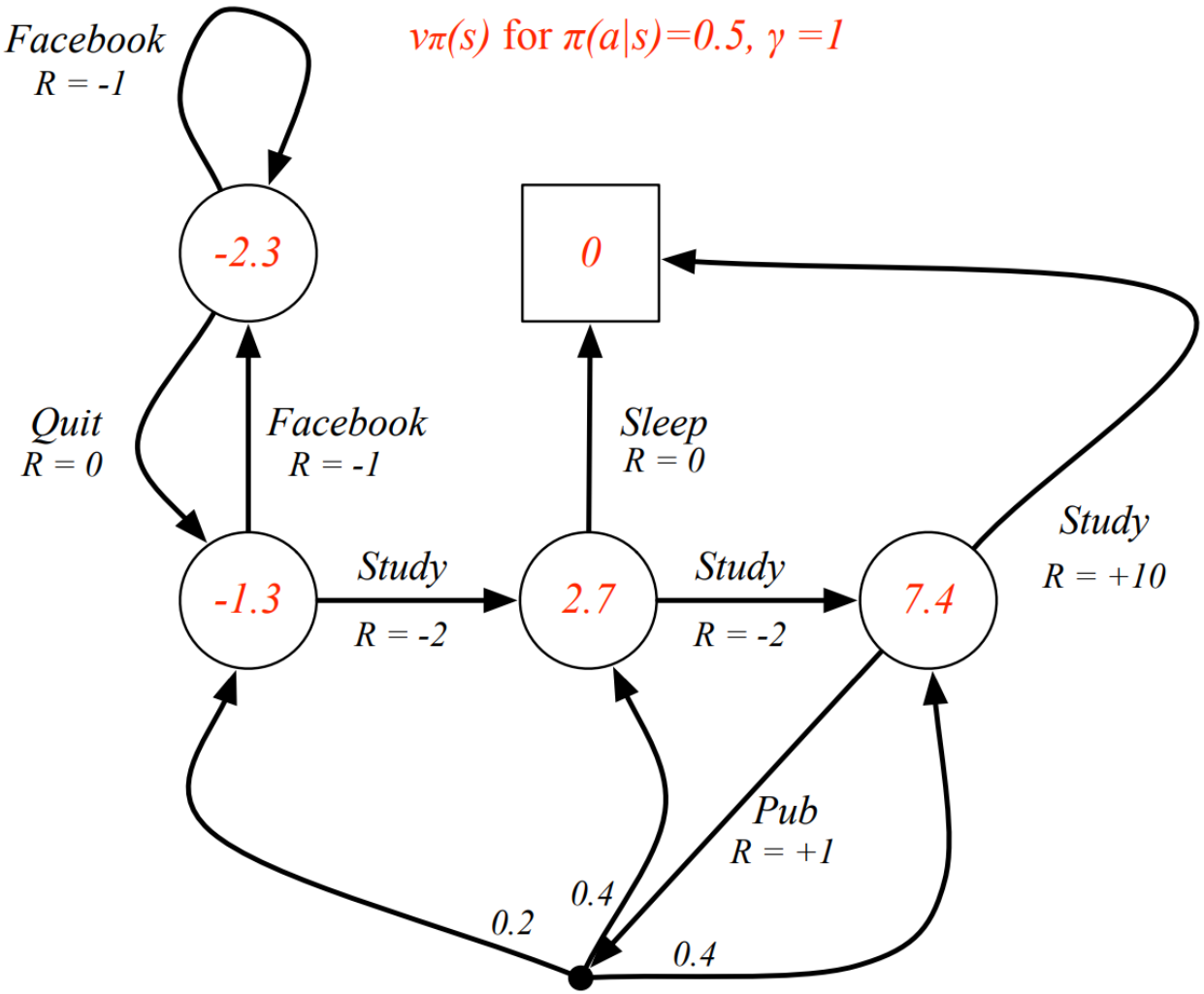
- **state-value function** $v_\pi(s)$ is the expected return from following the policy π in state s:

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

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

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

Example: state-value function for student MDP



Bellman equation

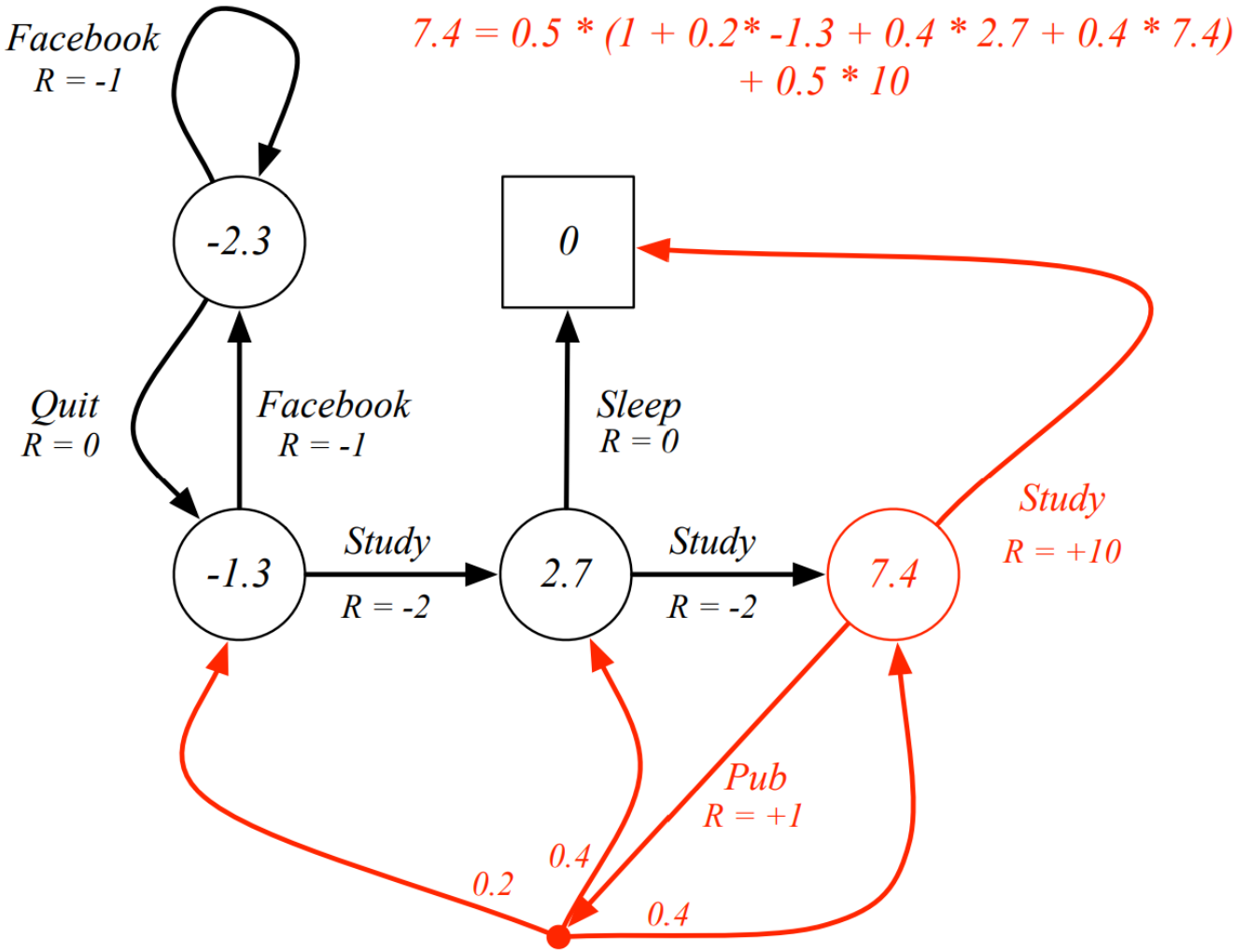
- **state-value** function can be decomposed into immediate reward plus discounted value of successor state:

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

- **action-value** function can similarly be decomposed:

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

Example: Bellman equation



Exercise: value function

MDP:

- States: $S = \{A, B\}$
- Actions: $A = \{\text{Left}, \text{right}\}$
- Rewards:
  - $R(A, \text{Left}, A) = 1$
  - $R(A, \text{right}, B) = 0$
  - $R(B, \text{Left}, A) = 0$
  - $R(B, \text{right}, B) = 2$

Exercise: value function

- transition probabilities:
  - $P(A|A, \text{Left}) = 1$
  - $P(B|A, \text{right}) = 1$
  - $P(A|B, \text{Left}) = 1$
  - $P(B|B, \text{right}) = 1$
- policy:
  - $\pi(A) = \text{Left}$
  - $\pi(B) = \text{right}$
- discount factor $\gamma = 0.5$

Solution: value function

- calculate the value function using Bellman's equation:
- state A:

$$\begin{aligned} V^{\pi}(A) &= P(A|A, Left)\left[R(A, Left, A) + \gamma V^{\pi}(A)\right] \\ &= 1 \cdot (1 + 0.5V^{\pi}(A)) \end{aligned}$$

- state B:

$$\begin{aligned} V^{\pi}(B) &= P(B|B, Right)\left[R(B, Right, B) + \gamma V^{\pi}(B)\right] \\ &= 1 \cdot (2 + 0.5V^{\pi}(B)) \end{aligned}$$

- solving the equations yields $V^{\pi}(A) = 2$, $V^{\pi}(B) = 4$

Optimal value function

- **optimal state-value function** $v_*(s)$ is max over all policies:

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

- **optimal action-value function** $q_*(s, a)$ is max over all policies:

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

- How to obtain the optimal policy?

Value iteration

- idea: iteratively update the value of each state to reflect the best possible reward that can be obtained starting from that state
- value of a state is updated based on the rewards and values of the states that can be reached from it
- this process converges to the optimal value function

Value iteration

Update rule for the value function $V(s)$ for each state s:

$$V_{k+1}(s) = \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_k(s') \right\},$$

where

- $V_k(s)$ is the value of state s at iteration k
- $R(s,a)$ is the reward received after taking action a in state s
- γ is the discount factor
- $P(s'|s,a)$ is the transition probability

Policy iteration

- Two steps:
  - policy **evaluation**
  - policy **improvement**
- alternate between these two steps until the policy no longer changes
- intuition: by gradually improving the policy based on its current performance, we eventually reach the best possible policy

Policy iteration

- policy **evaluation** (for a given policy π):

$$V^{\pi}(s) \leftarrow R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) V^{\pi}(s')$$

- policy **improvement**:

$$\pi(s) \leftarrow \text{argmax}_{a \in \mathcal{A}} \left\{ R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^{\pi}(s') \right\}$$

Temporal Difference (TD) Learning

- combines ideas from Monte Carlo and dynamic programming
- **model-free**
  (needs no model for environment, reward, transition probabilities)
- learns directly from raw experience without a model of the environment's dynamics
- updates estimates of the value of a state based on observed reward and current estimate, **without waiting for a final outcome**

Temporal Difference (TD) Learning

- **TD(0)** update for the state-value function:

$$V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s))$$

- Q-**learning** update for the action-value function:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

- **SARSA** update for the action-value function:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$$

Policy gradient methods

- directly optimize policy by adjusting parameters of policy in the direction of increase of expected return
- reinforcement:

$$\theta = \theta + \alpha \sum_t G_t \cdot \nabla_\theta \log \pi(a_t|s_t, \theta)$$

- **actor** update:

$$\theta \leftarrow \theta + \alpha \cdot \delta \cdot \nabla_\theta \log \pi(a_t|s_t, \theta)$$

- **critic** update:

$$V_\phi(s) \leftarrow V_\phi(s) + \beta\delta$$

- TD error:

$$\delta = r + \gamma V_\phi(s') - V_\phi(s)$$