Reinforcement Learning 01

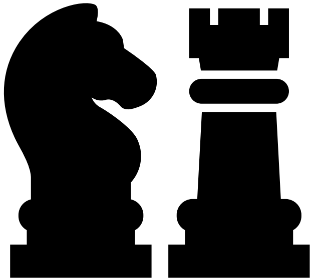
DS Development Presentations

Peter Nicholas S. Onglao | MNL

Background

GOAL:

Make an RL algorithm for chess (>3 presentations away) that can beat me



Contents

Sources

- UCL Course on Reinforcement Leaning, David Silver
  - Video lectures with accompanying slides
  - David Silver: AlphaGo, AlphaZero, AlphaStar (DeepMind)
  - Content is patterned after his lectures
- Reinforcement Learning – An Introduction, Sutton and Barto

1

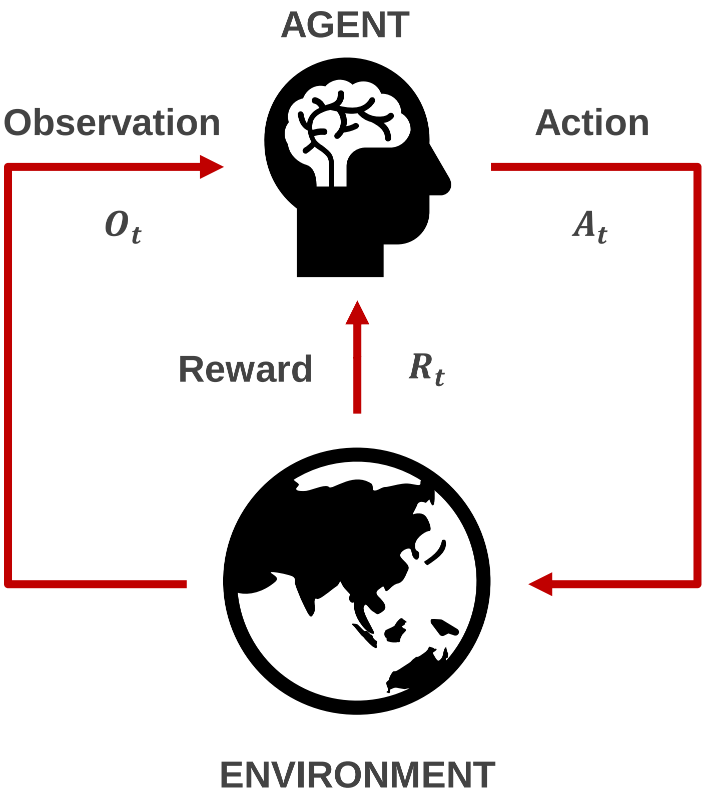
What is RL?

Introduction + basic math requirements for an RL problem.

Introduction to RL

- What makes RL different from supervised/unsupervised learning?
  - No supervisor, only a **reward**
  - Feedback is delayed, not instantaneous
  - **Time really matters**, not i.i.d. data
  - The agent affects the data it receives
- Basic components:
  - Agent
  - Environment
  - Observation
  - Action
  - Reward

RL Components



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

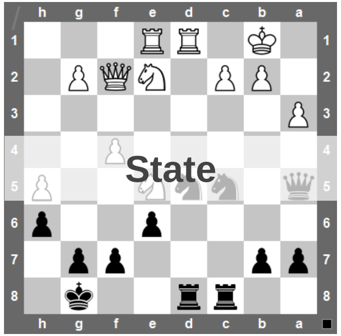
Reward

- The reward R_t is a **SCALAR** feedback signal
- Indicates how well the agent is doing at step t
- The agent's job is to maximize **cumulative reward**
  - Cumulative reward since actions may have long-term consequences, and reward may be delayed
  - Immediate vs. long-term gain
- **Reward hypothesis**
  - All goals can be described by the maximization of expected cumulative reward

History and State

- How will the agent act? History and State
- **History**: the sequence of observations, actions, rewards
  - $H_t = O_1, R_1, A_1, \ldots, A_{t-1}, O_t, R_t$
- All observable variables up to time t
- The future depends on the history
  - Agent: selects action
  - Environment: selects observation/reward
- **State**: a **concise** way to represent history
  - It is a function of history
  - $S_t = f(H_t)$

https://mat3e.github.io

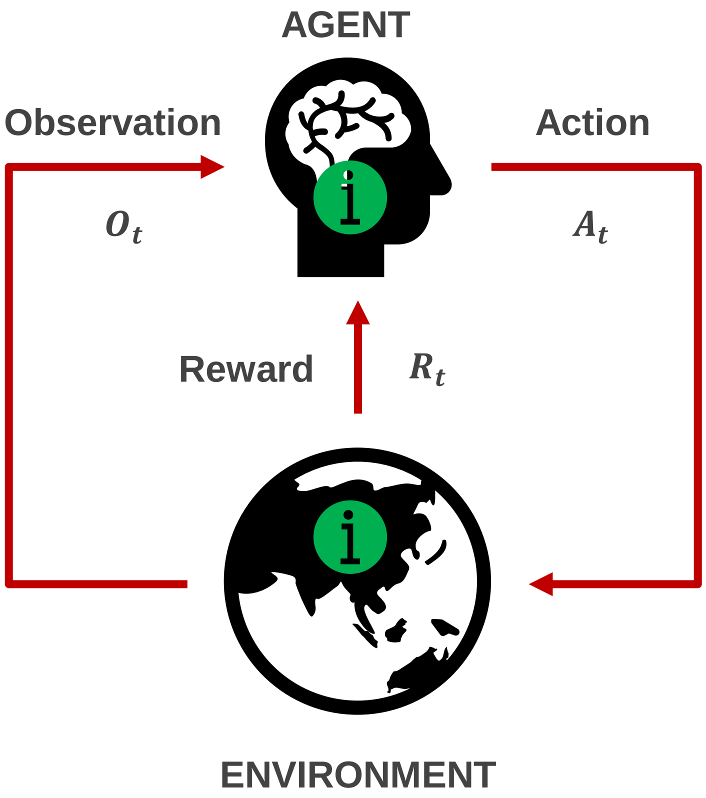


chessfox.com

Environment and Agent States

- **Environment State** (S_t^e)
  - The **internal state** of the environment (some set of numbers)
  - The data the environment uses to pick the next observation/reward
  - Not usually visible to agent
  - If it is visible, it may not be useful to the agent
- **Agent State** (S_t^a)
  - The set of numbers inside the algorithm
  - The information used by the agent to pick the next action
  - **Our role** – choose which information to keep/take

Agent and Environment States



Information/Markov State

- **Markov State**
  - Contains all useful information from the history
  - A state S_t is Markov iff:

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

- "The future is independent of the past given the present"
  - $H_{1:t} \rightarrow S_t \rightarrow H_{t+1:\infty}$
- Once the current state is known, the **history may be thrown away**
- The state is a sufficient statistic of the future
- Environment state S_t^e is Markov
- History H_t is Markov

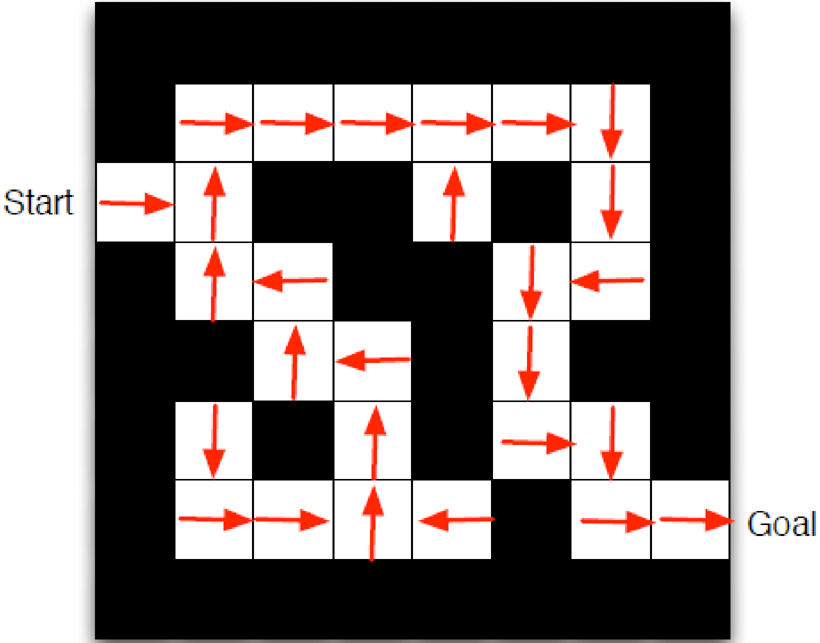
Components of an RL Agent

- An RL agent may include **one or more** of the following:
  - **Policy**: agent's behavior function
  - **Value function**: how good is each state and/or action
  - **Model**: agent's representation of the environment

Components of an RL Agent

- **Policy**: the agent's behavior; a map from state to action
  - Deterministic: $a = \pi(s)$
  - Stochastic: $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$
- **Value function**: prediction of future reward; evaluate the current state
  - $v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+2} + \cdots |S_t = s]$
- **Model**: predicts what the environment will do next
  - $\mathcal{P}$ predicts the **next state**
    - $\mathcal{P}_{ss'}^a = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]$
  - $\mathcal{R}$ predicts the **next reward**
    - $\mathcal{R}_s^a = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$

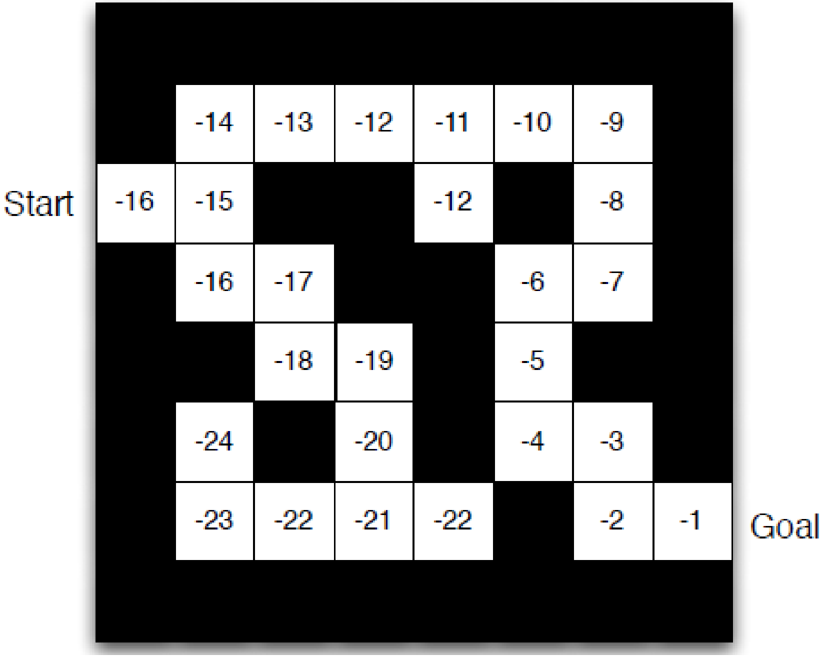
Maze Example: Policy



Credits: David Silver's slides

- Each state s = a position in the maze
- Arrows represent policy $\pi(s)$ for each state s

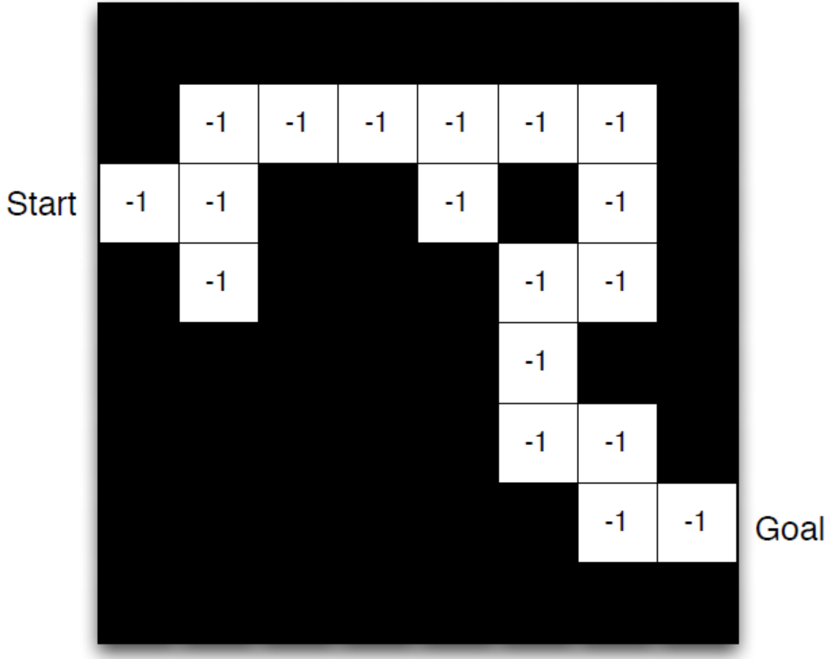
Maze Example: Value Function



Credits: David Silver's slides

- Each state s = a position in the maze
- Reward: -1 per step
- Numbers represent the value function $v_\pi(s)$ of each state s

Maze Example: Model



Credits: David Silver’s slides

- Agent may have an **internal model** of the environment
- Model may be imperfect
- Grid layout represents transition model $\mathcal{P}_{ss'}^{a}$
- Numbers represent immediate reward R_s^a from each state s
  - All the same since the *immediate reward* is the same each step

Categories of RL agents

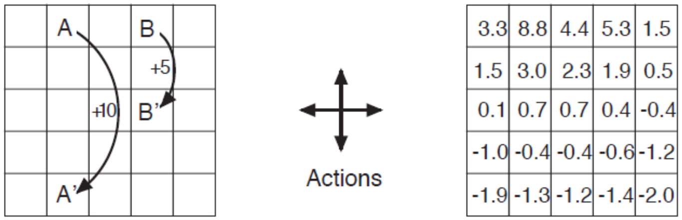
- (1) Depending on Policy & Value Function
  - Value Based: No Policy (Implicit) + Value Function
  - Policy Based: Policy + No Value Function
  - Actor Critic: Policy + Value Function
- (2) Depending on Model
  - Model Free: Policy and/or Value Function + No Model
  - Model Based: Policy and/or Value Function + Model

Differentiating some RL terms

- Learning vs. Planning
  - **Learning**
    - **Environment** is initially **unknown**
    - Agents interacts w/ environment and improves its policy
  - **Planning**
    - A model of the **environment** is **known**
    - Agent does not perform interactions but performs computations, and improves its policy this way

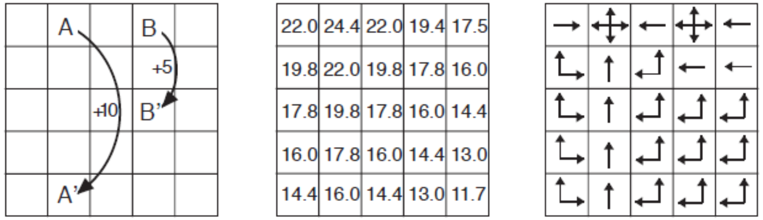
Differentiating some RL terms

- Exploration vs Exploitation
  - **Exploration**: Discover a good policy
  - **Exploitation**: W/o losing too much reward along the way
- Prediction vs Control
  - **Prediction**: evaluate the future given a policy
  - **Control**: optimize the future and find the best policy



(a) (b)

Prediction (policy = random)



a) gridworld b) v_* c) π_*

Control

Credits: David Silver's slides

Summary

- **Components of an RL problem**
  - Agent, Environment
  - Observation, Reward, Action
  - History and State
- **Components of an RL agent**
  - Policy, Value Function, Model
  - Agents can be categorized by presence/absence of those three
- **Differentiating some RL terms**
  - Learning vs Planning
  - Exploration vs Exploitation
  - Prediction vs Control

2

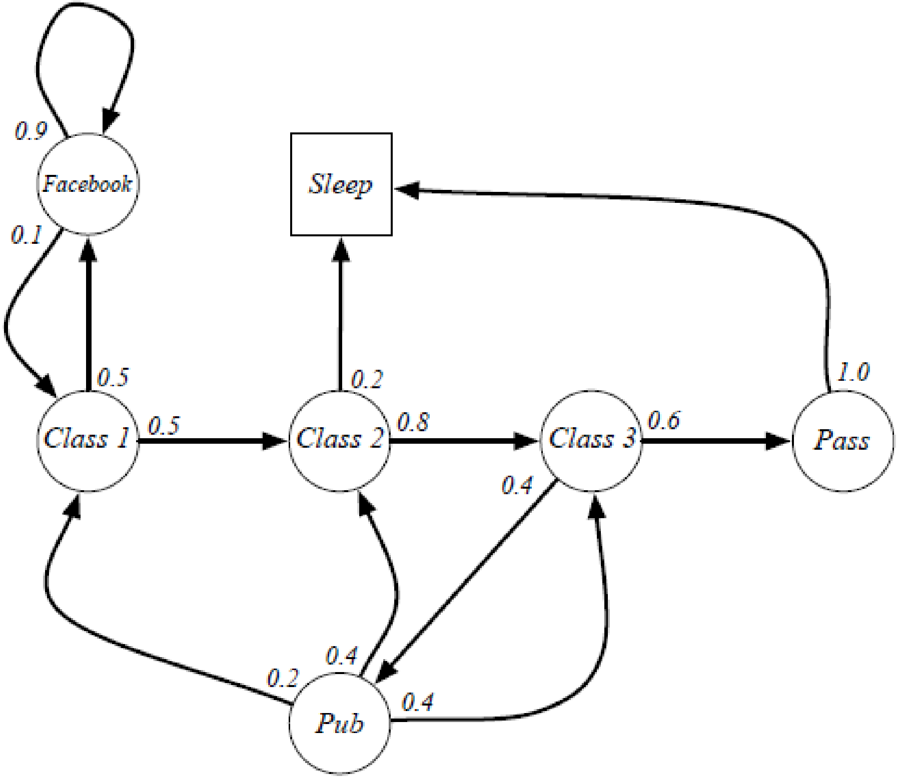
Markov Decision Process

MDPs, MRPs, and the Bellman Equation

Markov Process

- **Markov Process**/Markov Chain
  - A tuple $\langle \mathcal{S}, \mathcal{P} \rangle$
    - $\mathcal{S}$ is a (finite) set of states
    - $\mathcal{P}$ is a state transition probability matrix

Example: Student Markov Chain



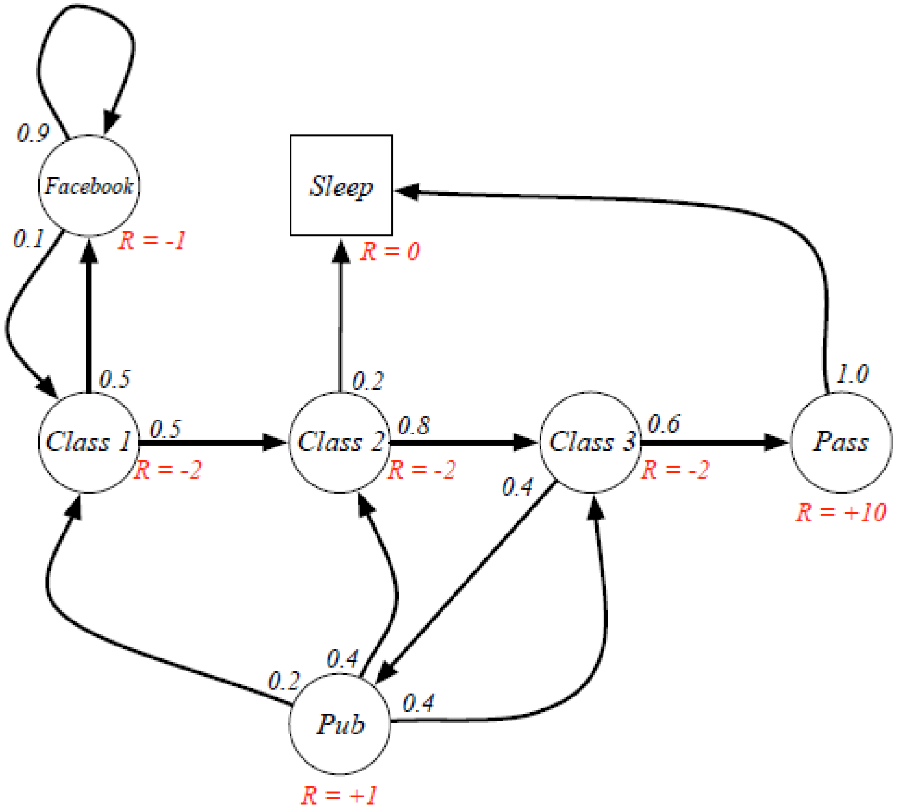
$$
\mathcal{P} = \begin{array}{c} \\ C1 \\ C2 \\ C3 \\ Pass \\ Pub \\ FB \\ Sleep \end{array}
\begin{array}{c}
\begin{array}{ccccccc} C1 & C2 & C3 & Pass & Pub & FB & Sleep \end{array} \\
\left[\begin{array}{ccccccc}
 & 0.5 & & & & 0.5 & \\
 & & 0.8 & & & & 0.2 \\
 & & & 0.6 & 0.4 & & \\
 & & & & & &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

Credits: David Silver's slides

Markov Reward Process (MRP)

- A Markov chain with **values**
- **Markov Reward Process**
  - A tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
    - $\mathcal{S}$ is a (finite) set of states
    - $\mathcal{P}$ is a state transition probability matrix
    - $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}[R_{t+1} | S_t = s]$
    - γ is a discount factor, $\gamma \in [0,1]$

Example: Student MRP



Credits: David Silver's slides

Return

- Return $\boldsymbol{G_t}$ is the **total discounted reward** from time-step t

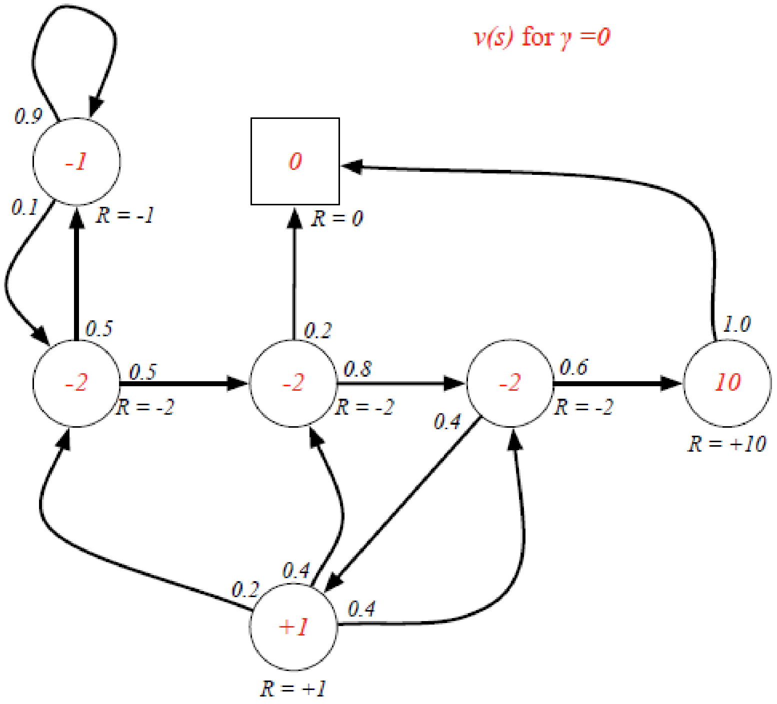
$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

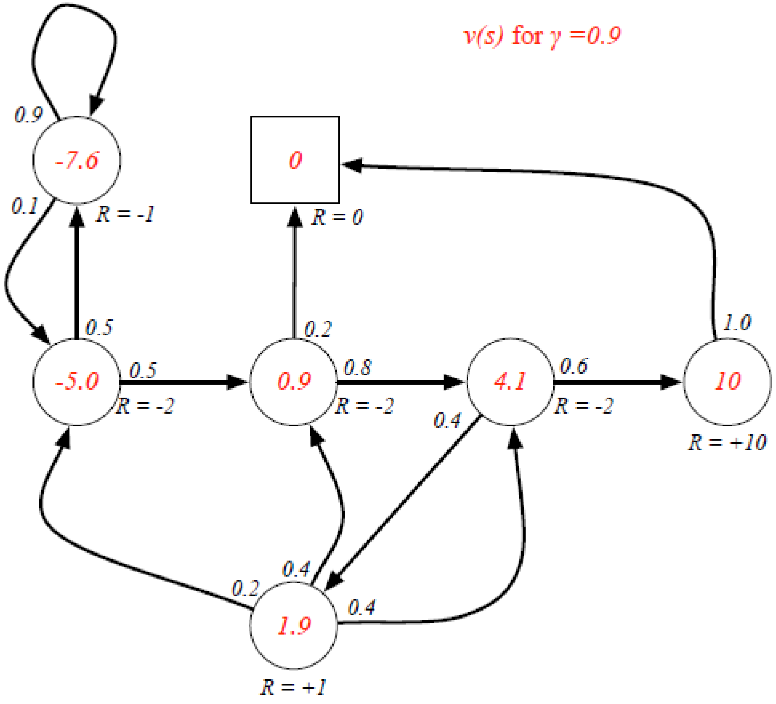
  - Discount γ is the **present value of future rewards**
  - Value immediate reward vs. delayed reward
    - γ near 0: “myopic”
    - γ near 1: “far-sighted”

- The state value function $\boldsymbol{v(s)}$ gives the expected return of state s

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

State-Value Function for Student MRP





Credits: David Silver's slides

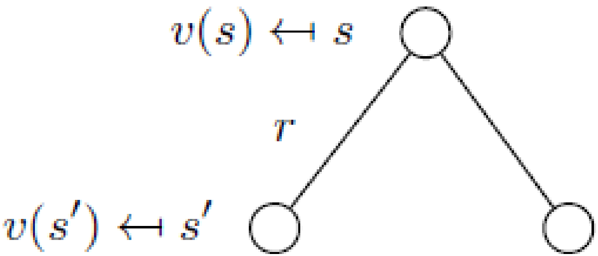
Bellman Equation for MRPs

- The value function can be decomposed into **two parts**:
  - Immediate reward
  - Discounted value of successor state

$$
\begin{aligned}
v(s) &= \mathbb{E}[G_t | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma (R_{t+2} + \gamma^2 R_{t+3} + \cdots) | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] \\
&= \mathbb{E}[\underbrace{R_{t+1}}_{\text{Immediate reward}} + \underbrace{\gamma v(S_{t+1})}_{\text{Discounted value of next state}} | S_t = s]
\end{aligned}
$$

Bellman Equation for MRPs

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

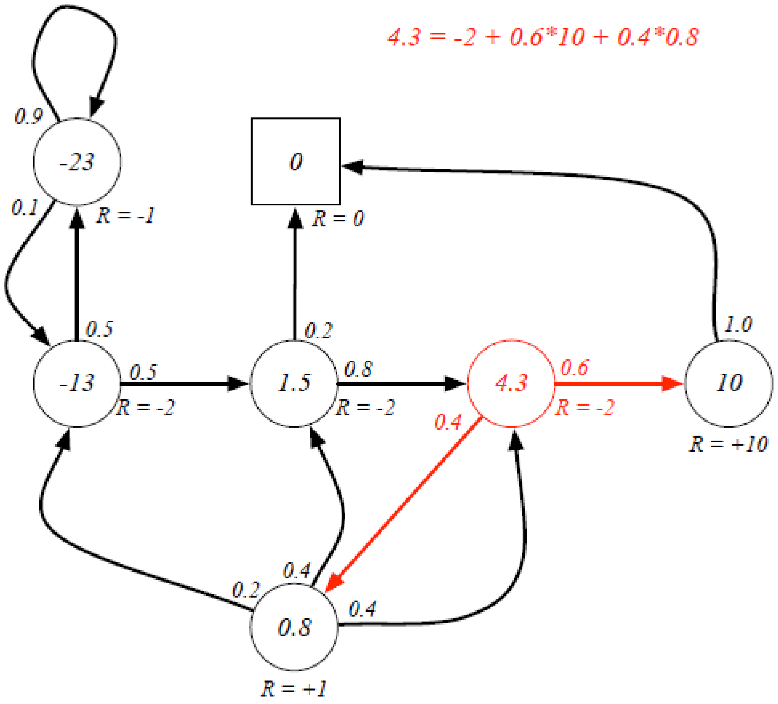


$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} P_{ss'} v(s')$$

Credits: David Silver's slides

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix, with elements $P_{ss'}$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}[R_{t+1} | S_t = s]$

Bellman Equation for Student MRP



Credits: David Silver's slides

Bellman Equation: Matrix Form

$$\begin{bmatrix} \mathrm{v}(1) \\ \vdots \\ \mathrm{v}(\mathrm{n}) \end{bmatrix} = \begin{bmatrix} \mathcal{R}(1) \\ \vdots \\ \mathcal{R}(n) \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \cdots & \mathcal{P}_{1n} \\ \vdots & \ddots & \vdots \\ \mathcal{P}_{n1} & \cdots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

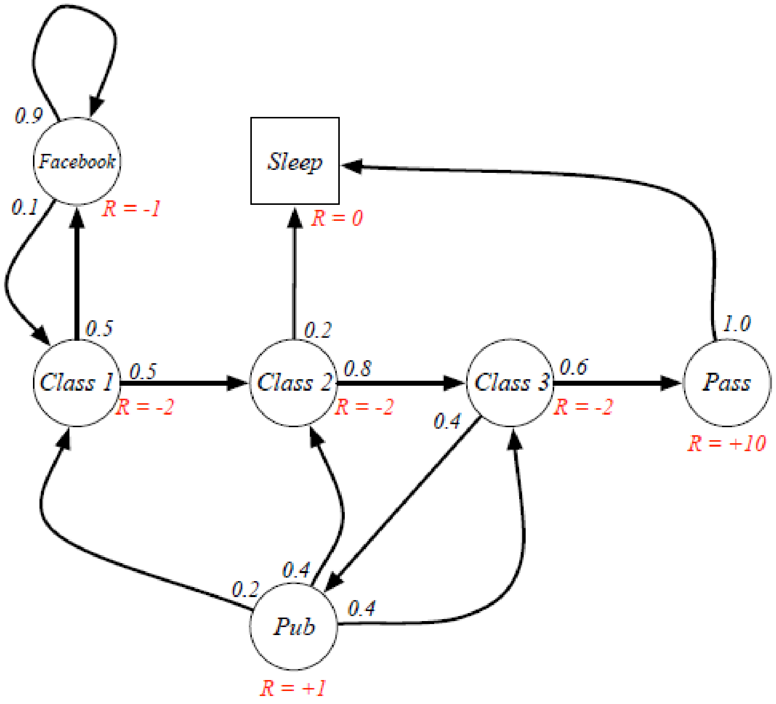
$$\begin{aligned} \mathrm{v}(s) &= \mathcal{R} + \gamma \mathcal{P} \mathrm{v} \\ (I - \gamma \mathcal{P})\mathrm{v} &= \mathcal{R} \\ \mathrm{v} &= (I - \gamma \mathcal{P})^{-1} \mathcal{R} \end{aligned}$$

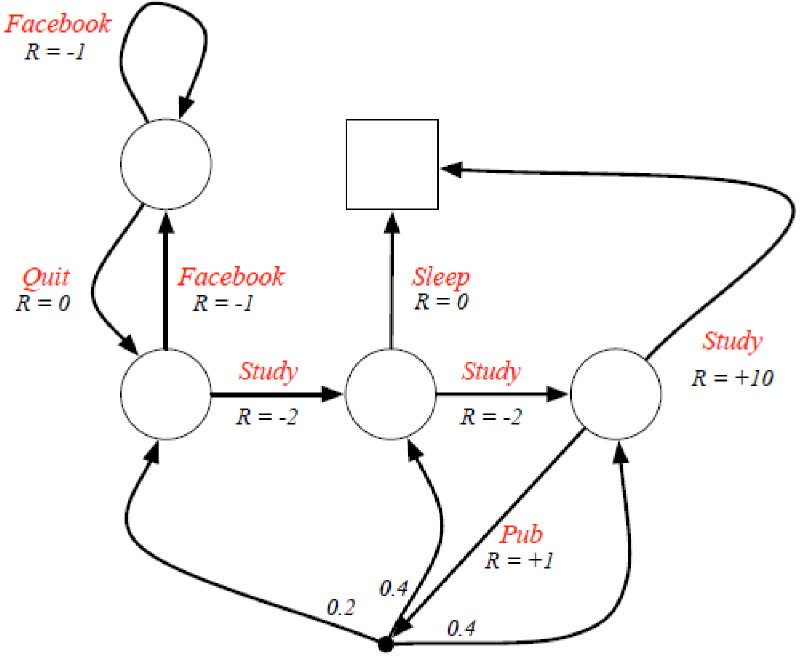
- We can solve the Bellman equation! But it's $\boldsymbol{O(n^3)}$ **for $\boldsymbol{n}$ states**.
- For larger n there are iterative solutions:
  - Monte-Carlo evaluation, Temporal-Difference learning
  - Dynamic Programming

Markov Decision Process (MDP)

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

Student MRP vs Student MDP





Credits: David Silver's slides

Policies for MDP

- **Policy** (π): a distribution of actions given states
  - $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$
  - Fully defines the behavior of an agent
  - MDP policies depend on the current state, not the history
  - *Stationary* (time-independent), since you're considering all possible states

Value Functions for MDP

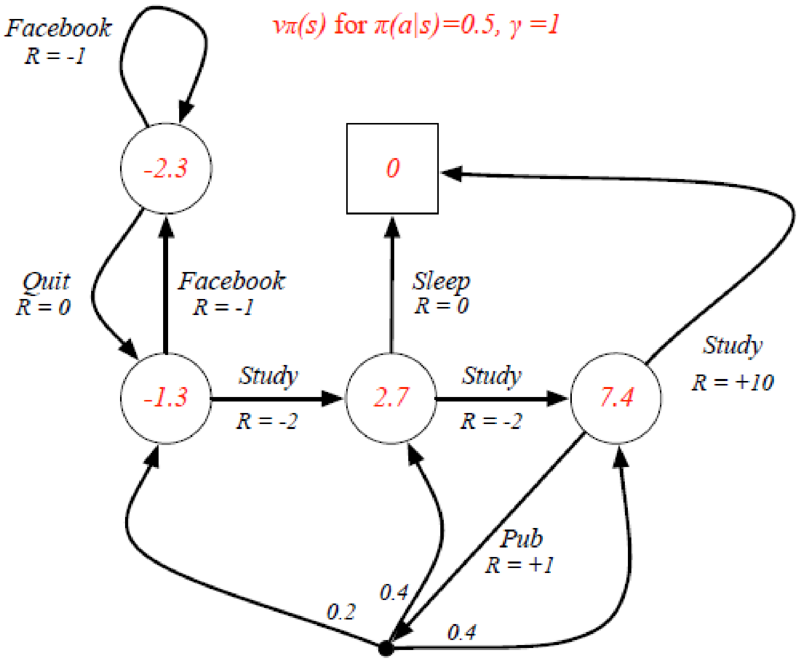
- **State-value function** ($v_\pi(s)$): the expected return following policy π, from state s

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

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

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

State-Value Function for Student MDP



Credits: David Silver’s slides

Bellman Expectation Equations

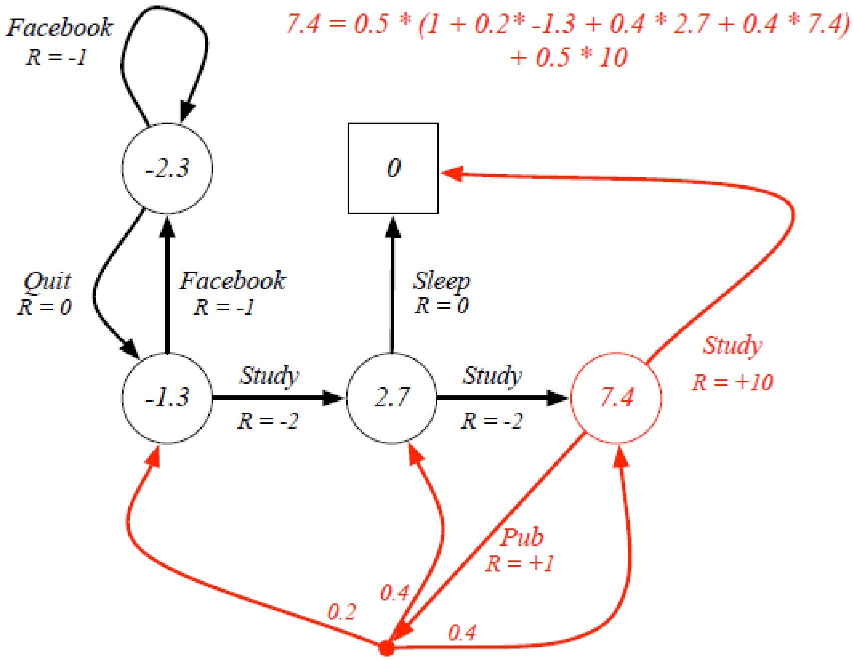
- **State-value function** ($v_\pi(s)$): the expected return following policy π, from state s

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

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

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

Bellman Expectation Equation in Stud. MDP



Credits: David Silver's slides

Finding an Optimal Value Function

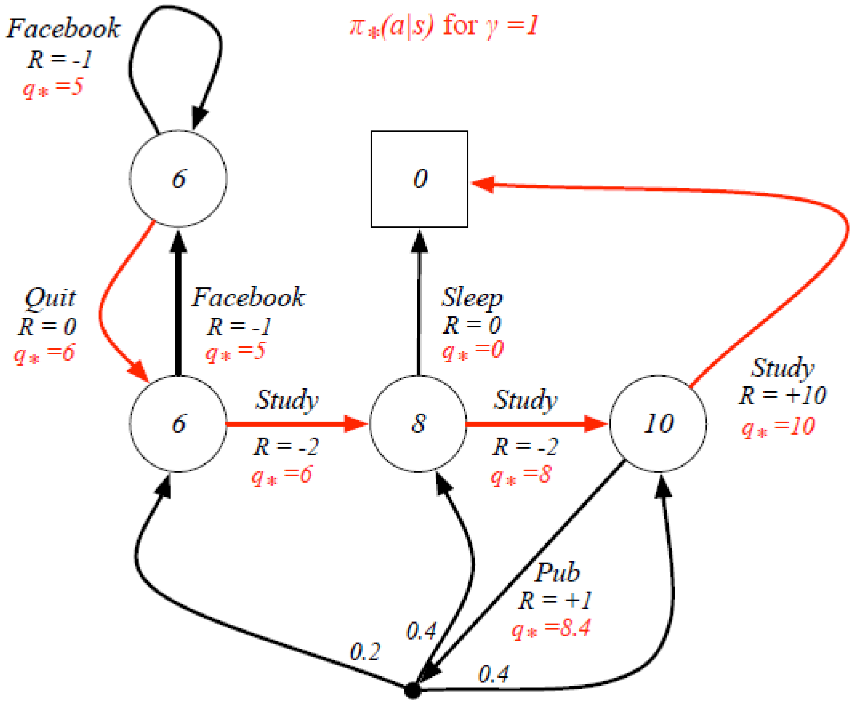
- **Optimal state-value function** ($v_*(s)$): the maximum state-value function over all policies
$$v_*(s) = \max_\pi v_\pi(s)$$
- **Optimal action-value function** ($q_*(s,a)$): the maximum action-value function over all policies
$$q_*(s,a) = \max_\pi q_\pi(s,a)$$
- Theorem:
  - There exists an optimal policy π_* better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$
  - All optimal policies achieve the optimal state-value and action-value functions

Finding an Optimal Policy

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

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

Optimal Policy for Student MDP



Credits: David Silver's slides

Bellman Optimality Equation

- The Bellman equation for the state and action-value functions

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

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

Solving the Bellman Optimality Equation

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

Thank you!

Sources

- UCL Course on Reinforcement Leaning, David Silver
- Reinforcement Learning – An Introduction, Sutton and Barto

CREDITS

Special thanks to:

- Presentation template by SlidesCarnival
- Photographs by Startup Stock Photos