Reinforcement Learning

Professor Vwani Roychowdhury

University of California, Los Angeles

July 23, 2022

Overview

Introduction

Reinforcement learning framework

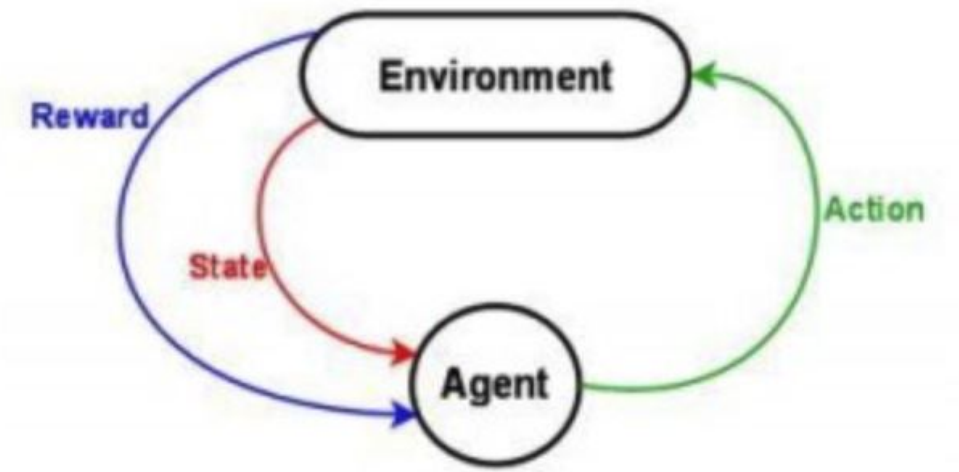
- Goals and Returns
- Model of the environment
- Value functions and Policy
- Optimal Policy

Reinforcement learning algorithms

- Bellman optimality equation
- Iterative solution methods for Bellman optimality equation
  - Value iteration

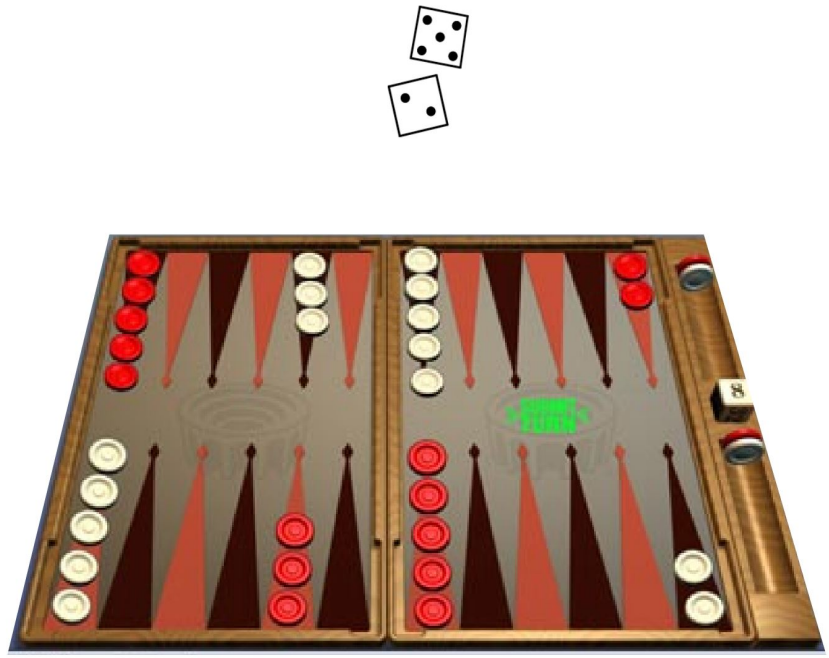
What is Reinforcement Learning?

Reinforcement Learning is the task of learning from interaction to achieve a goal. The learner and the decision maker is called the **agent**. The thing it interacts with, comprising everything outside the agent, is called the **environment**. These interact continually, the agent selecting actions and the environment responding to those actions by presenting rewards and new states.



Backgammon

- States: Configurations of the playing board
- Actions: Moves
- Rewards:
  - win: $+1$
  - lose: -1
  - else: 0



Goal of Reinforcement learning agent

The goal of an agent in a reinforcement learning problem is to maximize the total amount of reward it receives.

- In maze learning there is a reward of -1 for every time step that passes prior to escape. In order to maximize the reward, the agent learns to escape from the maze as quickly as possible.
- In learning to play chess, there is a reward of $+1$ for winning, -1 for losing, and 0 for drawing. In order to maximize the reward, the agent learns the winning strategy.

From the above examples, it can be seen that the reward signal is the way of communicating to the agent what you want it to achieve.

Return

The return R_t is the total discounted reward from time-step t

$$R_t = r_{t+1} + \gamma r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{1}$$

- r_{t+i} is the reward at $t + i$ time-step
- $\gamma \in [0, 1]$ is the discount factor and is the present value of future rewards
  - Discounting values immediate reward above delayed reward
  - γ close to 0 leads to short-sighted evaluation
  - γ close to 1 leads to far-sighted evaluation
- The goal of the agent is to choose action a_t such that the expected return is maximized

Blank Page

Blank Page

Blank Page

Blank Page

Blank Page

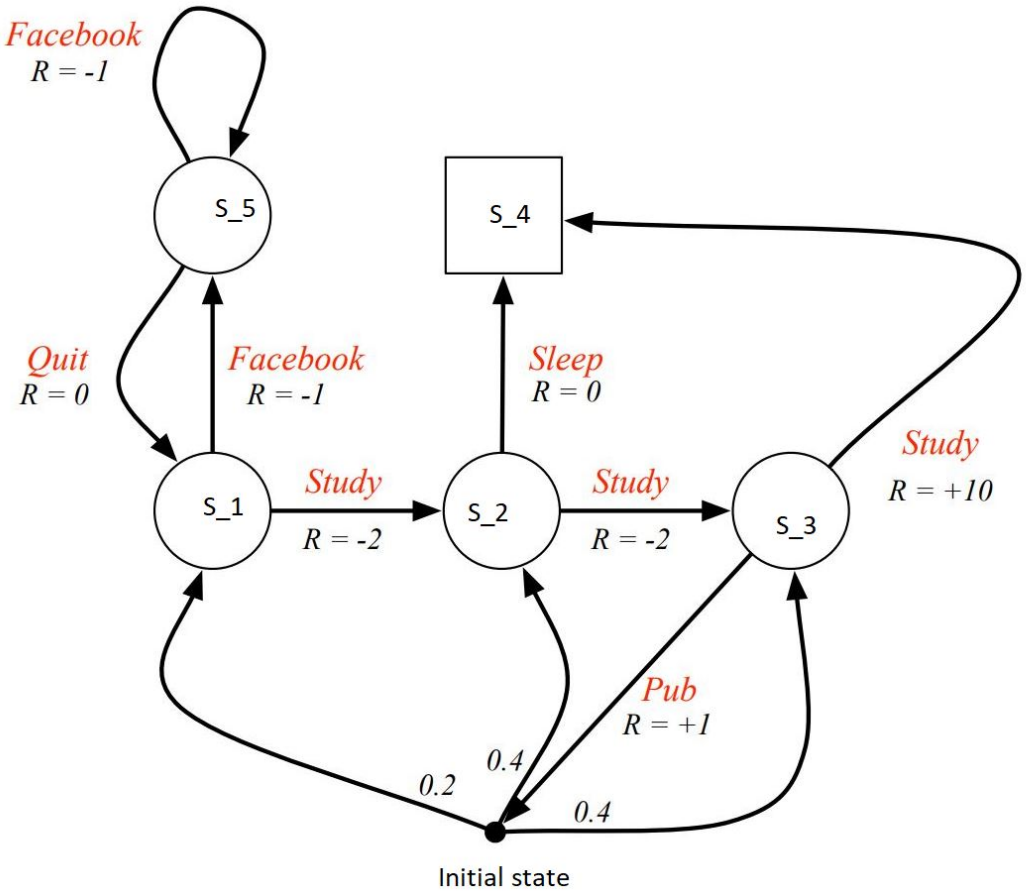
Model of the environment

The environment of the agent is modeled by a Markov Decision Process (MDP). A MDP is defined by its state and action sets and by the one-step dynamics of the environment.

- $\mathcal{S}$ is a set of **states**
- $\mathcal{A}$ is a set of **actions**
- $\mathcal{P}^a_{ss'}$ is a set of **transition probabilities**, where $\mathcal{P}^a_{ss'}$ is the probability of transitioning from state $s \in \mathcal{S}$ to state $s' \in \mathcal{S}$ after taking action $a \in \mathcal{A}$
  - $\mathcal{P}^a_{ss'} = \mathbb{P}(s_{t+1} = s' | s_t = s, a_t = a)$
- Given any current state and action, s and a, together with any next state, s', the expected value of the next reward is $\mathcal{R}^a_{ss'}$
  - $\mathcal{R}^a_{ss'} = \mathbb{E}(r_{t+1} | s_t = s, a_t = a, s_{t+1} = s')$

$\mathcal{P}^a_{ss'}$ and $\mathcal{R}^a_{ss'}$ completely specify the most important aspects of the dynamics of the MDP.

Example: Student MDP



This example was taken from `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/MDP.pdf`

Properties of the Student MDP

- Initial state has only one action, that is to start. Given this action, the transition probabilities for the initial state are:
  - $\mathbb{P}(S_1|\textit{Initial state}, \textit{start}) = 0.2$
  - $\mathbb{P}(S_2|\textit{Initial state}, \textit{start}) = 0.4$
  - $\mathbb{P}(S_3|\textit{Initial state}, \textit{start}) = 0.4$
- Transition probabilities for the non-initial state are delta functions
  - $\mathbb{P}(S_2|S_1, \textit{Study}) = 1$
  - $\mathbb{P}(S_5|S_1, \textit{Study}) = 0$
- Possible actions at each of the non-initial state are marked in Red
- Expected rewards for each action are represented by the R values
  - $\mathbb{E}(r_{t+1}|s_t = S_1, a_t = \textit{Study}, s_{t+1} = S_2) = -2$
  - $\mathbb{E}(r_{t+1}|s_t = S_1, a_t = \textit{Facebook}, s_{t+1} = S_5) = -1$

Value functions and policy

RL algorithms are based on estimating **value functions**. Two of the most commonly used value functions are:

- State-value function - Measure of 'how good' it is for the agent to be in a given state
- Action-value function - Measure of 'how good' it is to perform a given action in a given state

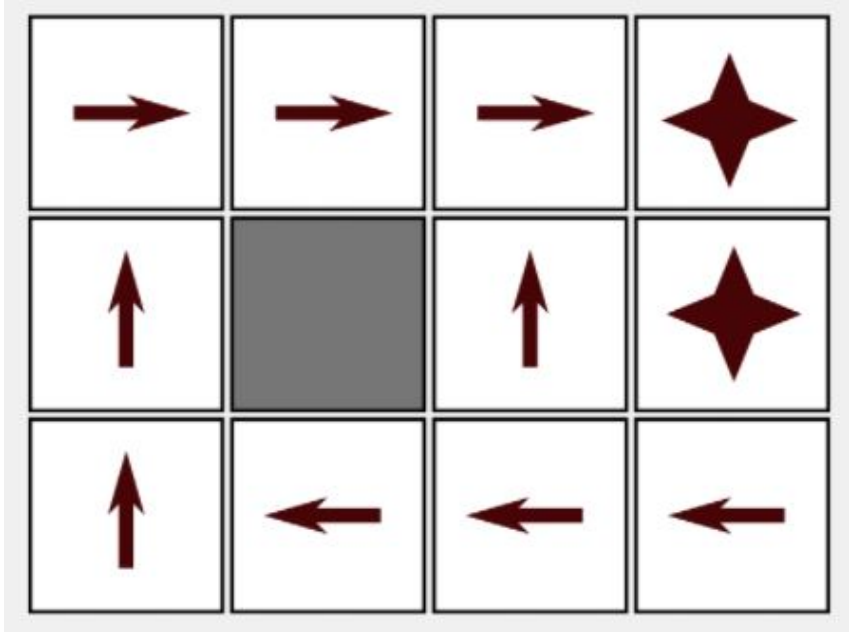
The notion of 'how good' is defined in terms of expected return. The rewards the agent can expect to receive in the future depend on what actions it will take. Therefore, value functions are defined with respect to particular policies.

Policy

A Policy π is a distribution over actions given states

$$\pi(a|s) = \mathbb{P}[a_t = a|s_t = s]$$

- A policy fully defines the behavior of an agent
- In a MDP, policies depend on the current state only
  - $\mathbb{P}[a_t|s_t, s_{t-1}, s_{t-2}, \cdots] = \mathbb{P}[a_t|s_t]$
- Policies are stationary (time-independent)



Blank Page

Blank Page

Blank Page

Blank Page

Blank Page

State-value function

State-value function for policy π, denoted by $V^{\pi}(s)$, is given by

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[R_t \middle| s_t = s\right]$$
$$= \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s\right] \quad (2)$$

From the above expression, we can see that the state-value function of state s, is the total discounted reward an agent is expected to receive in it's lifetime starting from state s and following policy π. This value is a measure of goodness of state s.

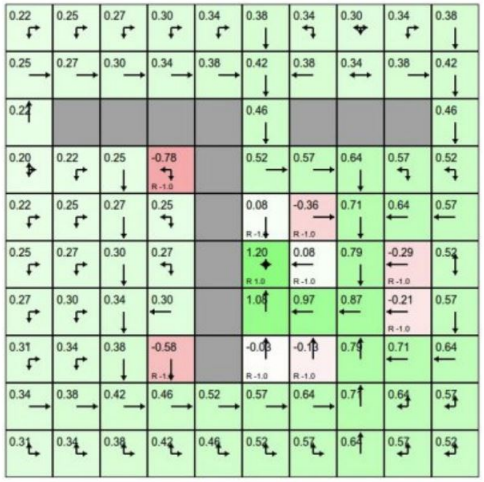
3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Action-value function

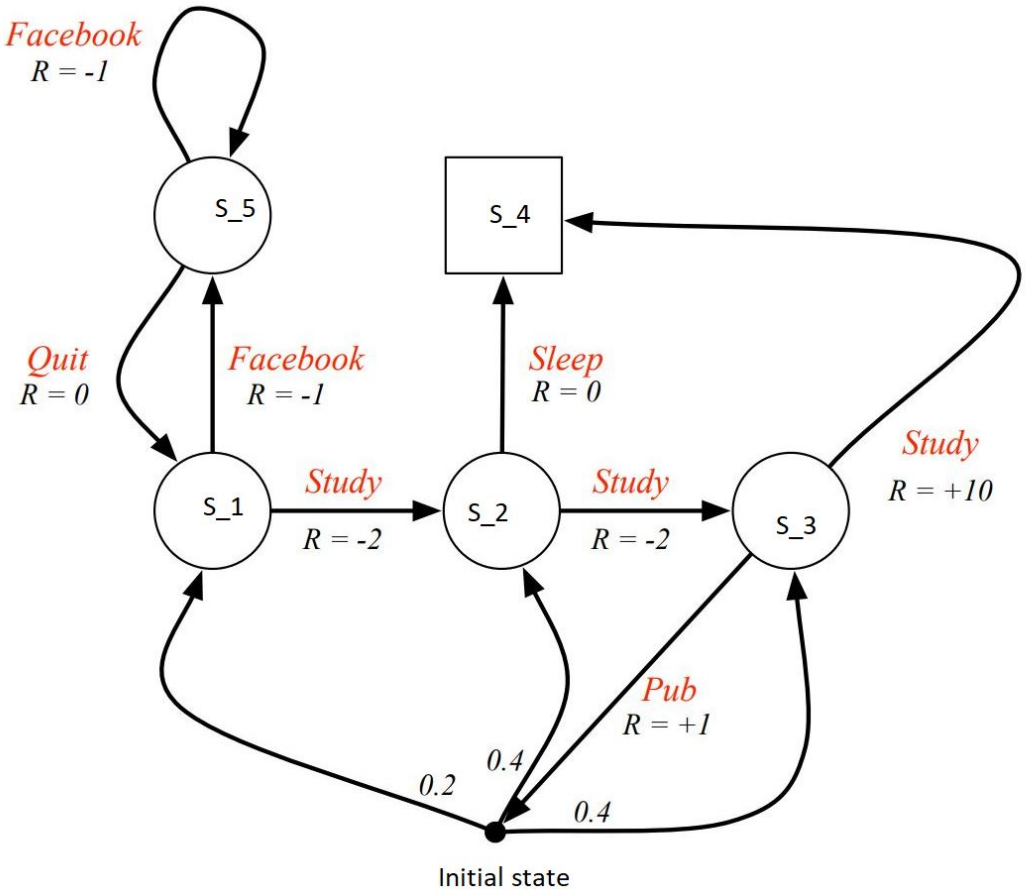
Action-value function for policy π, denoted by $Q^\pi(s, a)$, is given by

$$
\begin{aligned}
Q^\pi(s, a) &= \mathbb{E}_\pi \left[R_t \middle| s_t = s, a_t = a \right] \\
&= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a \right] \qquad (3)
\end{aligned}
$$

From the above expression, we can see that the action-value function is the expected discounted return for taking action a in state s and thereafter following policy π. This value is a measure of goodness of taking action a in state s.



Example: Student MDP

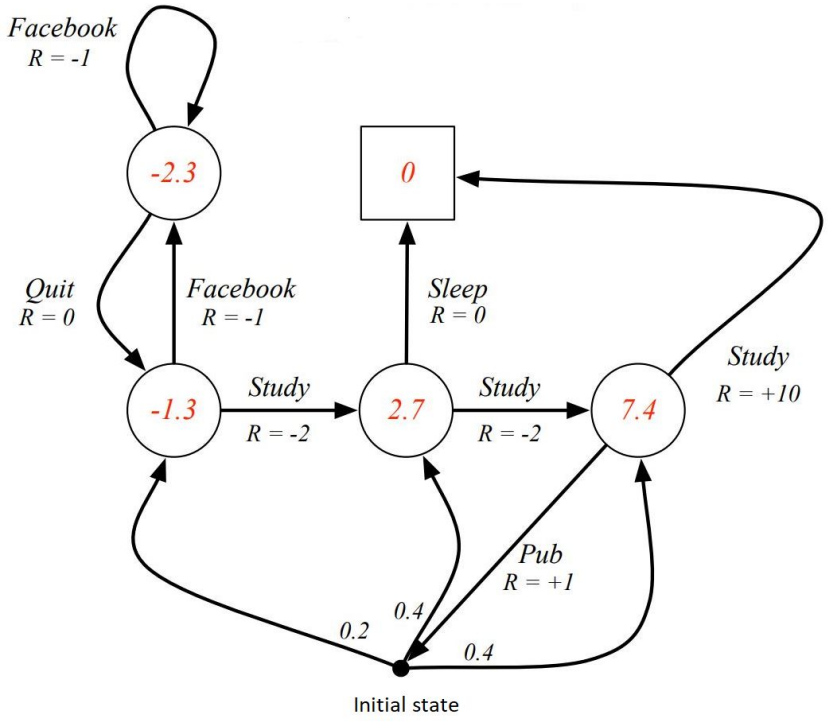


This example was taken from `http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/MDP.pdf`

Properties of the Student MDP

- Initial state has only one action, that is to start. Given this action, the transition probabilities for the initial state are:
  - $\mathbb{P}(S_1|\textit{Initial state}, \textit{start}) = 0.2$
  - $\mathbb{P}(S_2|\textit{Initial state}, \textit{start}) = 0.4$
  - $\mathbb{P}(S_3|\textit{Initial state}, \textit{start}) = 0.4$
- Transition probabilities for the non-initial state are delta functions
  - $\mathbb{P}(S_2|S_1, \textit{Study}) = 1$
  - $\mathbb{P}(S_5|S_1, \textit{Study}) = 0$
- Possible actions at each of the non-initial state are marked in Red
- Expected rewards for each action are represented by the R values
  - $\mathbb{E}(r_{t+1}|s_t = S_1, a_t = \textit{Study}, s_{t+1} = S_2) = -2$
  - $\mathbb{E}(r_{t+1}|s_t = S_1, a_t = \textit{Facebook}, s_{t+1} = S_5) = -1$

State-value of the Student MDP



The value of each state ($V^{\pi}(s)$) under equiprobable random policy, computed using equation 2, is marked in red.

- Since for each state, we have two possible actions, so under equiprobable random policy we have $\pi(a|s) = 0.5$
- Discount factor, $\gamma = 1$, was used

Bellman expectation equation

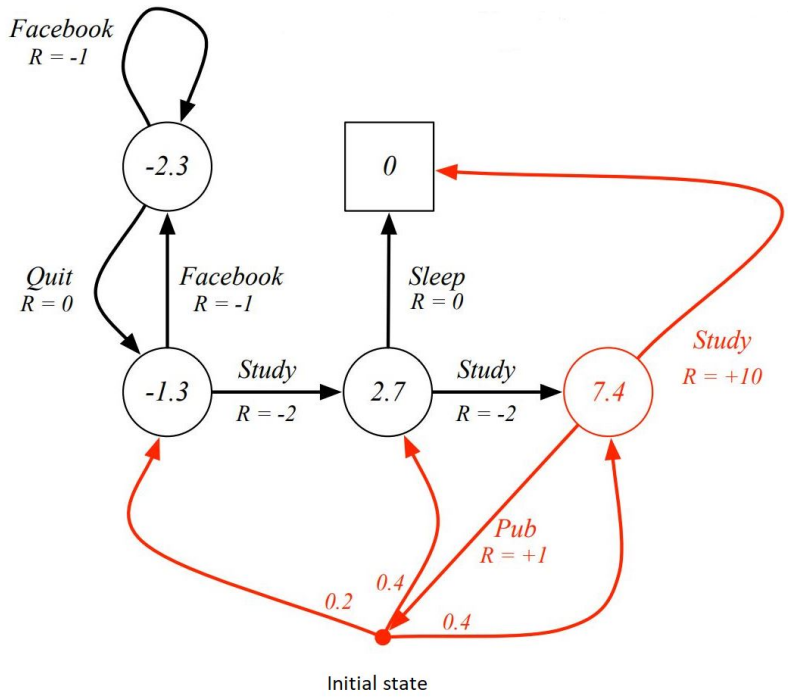
A fundamental property of value functions are that they satisfy particular recursive relationship. For any policy π and any state s, the following consistency condition holds between the value of s and the value of it's possible successor states:

$$
\begin{aligned}
V^{\pi}(s) &= \mathbb{E}_{\pi}[R_t | s_t = s] \\
&= \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s \right] \\
&= \mathbb{E}_{\pi}\left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \middle| s_t = s \right] \\
&= \sum_{s} \pi(a|s) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right] \qquad (4)
\end{aligned}
$$

Bellman expectation equation continued

Equation 4 is the Bellman equation for V^{π}, and it expresses a relationship between the values of a state and the values of it's successor states. V^{π} is the unique solution to it's Bellman equation and this equation forms the basis of a number of ways to estimate the optimal values of states and learn the optimal policy.

Bellman equation for Student MDP



The Bellman expectation equation (equation 4) for the state (S_3) marked in red is given below:

$$V^{\pi}(S_3) = \sum_{s} \pi(a|s) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

$$7.4 = 0.5 \times [1 + 0.2 \times (-1.3) + 0.4 \times 2.7 + 0.4 \times 7.4] + 0.5 \times 10$$

Optimal Policy

A policy π is defined to be better than or equal to a policy π' if it's expected return is greater than or equal to that of π' for all states

$$\pi \geq \pi' \iff V^{\pi}(s) \geq V^{\pi'}(s), \quad \forall s \in \mathcal{S}$$

For any MDP, we have the following:

- 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 state-value function, $V^{\pi^*}(s) = V^*(s)$
- All optimal policies achieve the optimal action-value function, $Q^{\pi^*}(s, a) = Q^*(s, a)$

Optimal state-value and action-value functions

Optimal state-value function, denoted by V^*, is defined as

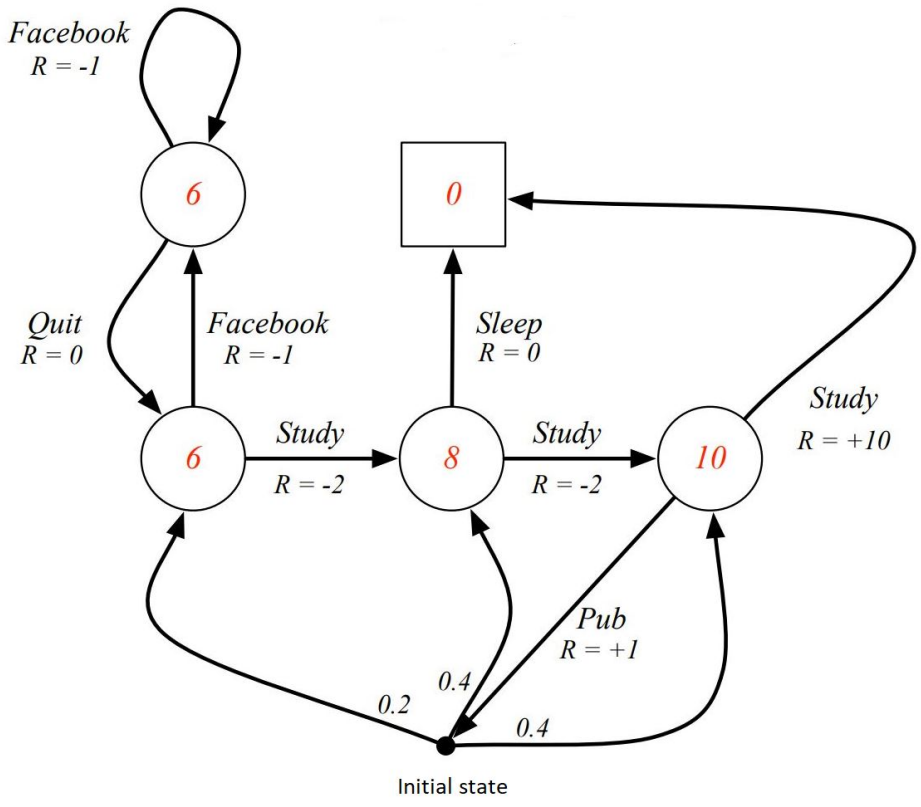
$$V^*(s) = \underset{\pi}{\text{maximize}} \quad V^{\pi}(s) \tag{5}$$

From equation 5, it can be observed that $V^*(s)$ is the maximum value function over all policies. Optimal action-value function, denoted by $Q_*(s, a)$, is defined as

$$Q^*(s, a) = \underset{\pi}{\text{maximize}} \quad Q^{\pi}(s, a) \tag{6}$$

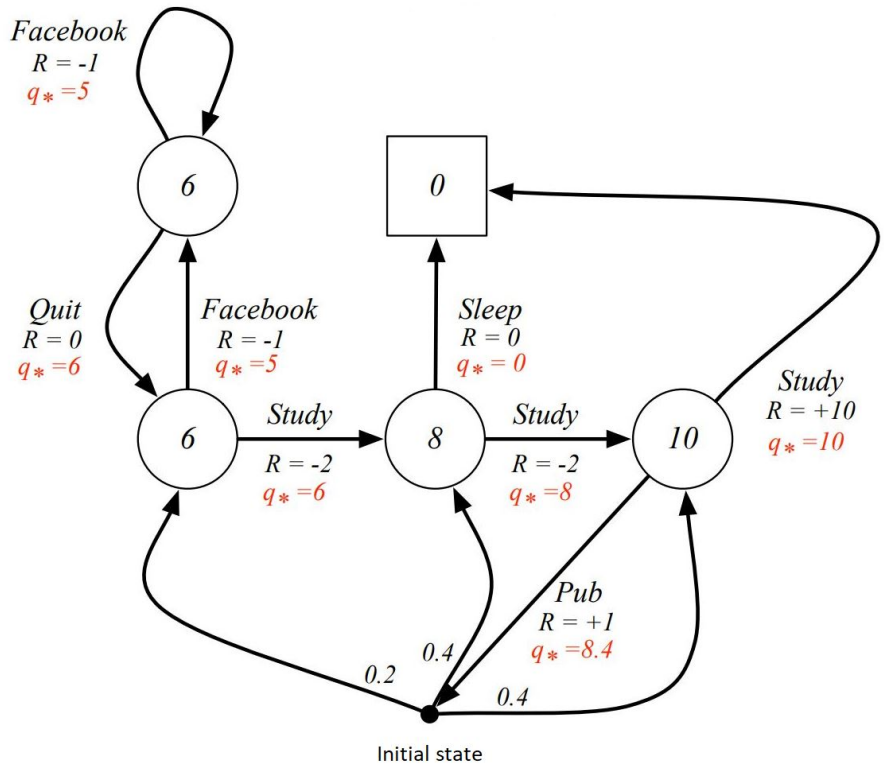
From equation 6, it can be observed that $Q^*(s, a)$ is the maximum action-value function over all policies.

Optimal state-value function for the Student MDP



The optimal state-value function ($V^*(s)$) for the Student MDP, computed using value iteration algorithm, is displayed in red.

Optimal action-value function for the Student MDP



The optimal action-value function for the Student MDP ($Q^*(s, a)$), computed using value iteration algorithm, is displayed in red.

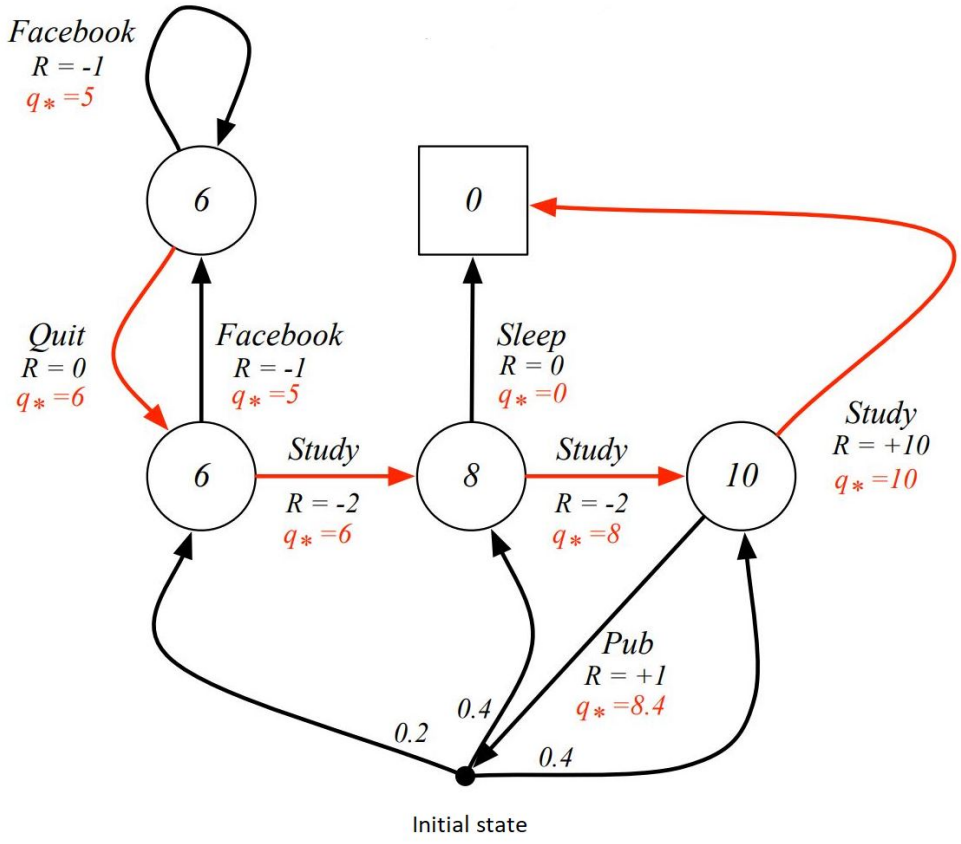
Finding optimal policy

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

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

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

Optimal policy of the Student MDP



The optimal deterministic policy of the Student MDP, that is the optimal action to take at each state, is marked in red.

Steps in RL algorithms

The main steps in RL algorithm are:

- Find optimal state-value or action-value functions
- Use the optimal state-value or action-value functions to determine the optimal policy

As shown in the previous slide, it is relatively straightforward to determine the optimal policy once we have the optimal state-value or action-value functions. We can find the optimal value functions using the Bellman optimality equation

Bellman optimality equation (in words)

Since V^* is the value function for a policy, so it must satisfy the Bellman consistency equation given by equation 4. However, since V^* is the optimal value function, so it satisfies a special form of the Bellman equation known as the Bellman optimality equation. Intuitively, the Bellman optimality equation expresses the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state.

Bellman optimality equation

$$
\begin{aligned}
V^*(s) &= \max_{a \in \mathcal{A}} Q^{\pi^*}(s, a) \\
&= \max_{a \in \mathcal{A}} \mathbb{E}_{\pi^*} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a \right] \\
&= \max_{a \in \mathcal{A}} \mathbb{E}_{\pi^*} \left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \middle| s_t = s, a_t = a \right] \\
&= \max_{a \in \mathcal{A}} \mathbb{E} \left[r_{t+1} + \gamma V^*(s_{t+1}) \middle| s_t = s, a_t = a \right] \qquad (7) \\
&= \max_{a \in \mathcal{A}} \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V^*(s')] \qquad (8)
\end{aligned}
$$

Equations 7 and 8 are two forms of the Bellman optimality equation for V^*.

Solving the Bellman optimality equation

- Bellman optimality equation is system of $|\mathcal{S}|$ non-linear equations
- In general, there do not exist any closed form solution to the Bellman optimality equation
- There exist many iterative solution methods
  - Value iteration
  - Policy iteration
  - Q-learning
  - Sarsa

Value iteration

Value iteration algorithm can be divided into 3 steps:

1. Initialization of the state-value function
2. Estimation of the optimal state-value function
3. Computation of the optimal deterministic policy

For the initialization step, the algorithm initializes all the state values to zero

$$V(s) = 0, \quad \forall s \in \mathcal{S}$$

For the estimation step, the value update is obtained simply by turning the Bellman optimality equations (given by equations 7 and 8) into an update rule

$$\begin{aligned} V_{k+1}(s) &= \max_{a \in \mathcal{A}} \mathbb{E}\left[r_{t+1} + \gamma V_k(s_{t+1}) \middle| s_t = s, a_t = a \right] \\ &= \max_{a \in \mathcal{A}} \sum_{s'} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V_k(s')] \end{aligned}$$

Value iteration (continued)

For the computation step, the optimal policy in any given state is the action that is expected to lead to the highest-valued state

$$\pi(s) = \arg \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} [\mathcal{R}^a_{ss'} + \gamma V(s')]$$

Value iteration formally requires an infinite number of iterations to converge exactly to V^*. In value iteration algorithm, we stop once the value function changes by only a small amount

$$\Delta = max(\Delta, |V_k(s) - V_{k+1}(s)|)$$

The algorithm terminates if $\Delta < \epsilon$, where ϵ is a small number.

Value iteration algorithm

procedure VALUE ITERATION($\mathcal{P}^a_{ss'}$, $\mathcal{R}^a_{ss'}$, $\mathcal{S}$, $\mathcal{A}$, γ):
  for all $s \in \mathcal{S}$ **do** ▷ Initialization
    $V(s) \leftarrow 0$
  end for
  $\Delta \leftarrow \infty$
  while $\Delta > \epsilon$ **do** ▷ Estimation
    $\Delta \leftarrow 0$
    for all $s \in \mathcal{S}$ **do**
      $v \leftarrow V(s)$;
      $V(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$;
      $\Delta \leftarrow max(\Delta, |v - V(s)|)$;
    end for
  end while
  for all $s \in \mathcal{S}$ **do** ▷ Computation
    $\pi(s) \leftarrow \arg \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V(s')]$;
  end for
end procedure return π

Limitations of value iteration algorithm

- Value iteration algorithm requires complete knowledge of the environment
  - Takes as input the MDP ($\mathcal{P}^a_{ss'}$, $\mathcal{R}^a_{ss'}$, $\mathcal{S}$, $\mathcal{A}$, γ)
- Value iteration algorithm involve operations over the entire state set of the MDP
  - In the estimation step we loop through the entire state set $\mathcal{S}$

Requirement of complete knowledge of the environment restricts the application of the algorithm to real world scenarios where we only have access to sample sequences from simulated interaction with the environment. The exhaustive sweep of the algorithm makes it prohibitively expensive for large state set. For example in the game of Backgammon we have around 10^{20} states and even if we could estimate value of million states per second, it would take over a million years for the algorithm to complete.

A word on other iterative solution methods

There exists other iterative solution methods for the Bellman optimality equations:

- Policy Iteration
- Q-learning
- SARSA

In this set of lecture slides, we will not cover these algorithms due to time-constraint. In project 3, you will implement the value-iteration algorithm to compute the optimal policy of an agent navigating in a 2-D environment.