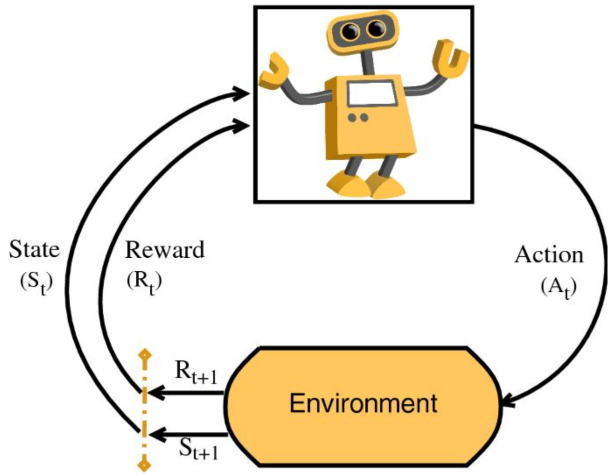
Markov Decision Process



DA344: Jul-Dec 2023

Office: MB 113, AV 106
vidyapradananda@gm.rkmvu.ac.in

RL and MDP

- Reinforcement Learning (RL) problems are modeled as Markov Decision Process (MDP).
- A Markov Decision Process (MDP) model contains:
  - A set of possible states S
  - A set of possible actions A
  - A real valued reward function R(s,a)
  - A transition T of each action’s effects in each state
- It follows the the Markov Property: The effects of an action taken in a state depend only on that state and not on the prior history.
- MDPs are solved through various algorithms, such as, backwards induction, linear programming, policy iteration, q-learning and value iteration
- MDP tool box : https://pymdptoolbox.readthedocs.io/en/latest/

Model-Based RL

- Agent has access to (or learns) a model of the environment. By a model of the environment, we mean a function which predicts state transitions and rewards.(famous example of this approach is **AlphaZero**.)
- it allows the agent to plan by thinking ahead, seeing what would happen for a range of possible choices, and explicitly deciding between its options
- ground-truth model of the environment is usually not available to the agent.it has to learn the model purely from experience
- What to learn?
  - policies, either stochastic or deterministic,
  - action-value functions (reward or Q-functions),
  - value functions, and/or environment models.

https://spinningup.openai.com/en/latest/spinningup/rl_intro.html

Model-Based RL

Model-generated objective of reinforcement learning: learn an optimal policy π that maximizes the expected total reward

$$\underset{\pi}{\text{maximize}} \quad \underset{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim p(\cdot|s_t, a_t)}}{\mathbb{E}} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]$$

To make it more clear: maximize the expected cumulative discounted rewards $r(s_t, a_t)$ from acting according to a policy π in an environment that is governed by system dynamics p

https://bair.berkeley.edu/blog/2019/12/12/mbpo/

https://bair.berkeley.edu/blog/2017/09/12/learning-to-optimize-with-rl/

Issues in Model-Based RL

- In Model based RL we assume a model of the environment and learn it from the interactions with the environment,
- Model-based methods learn with significantly lower sample than model-free RL methods.
- However, learning an accurate model of the environment has proven to be a challenging problem in certain domains. Modelling errors cripple the effectiveness of these algorithms, resulting in ***model-bias***
- How to alleviate the model-bias problem? By characterizing the uncertainty of the learned models with probabilistic models and ensembles

https://arxiv.org/pdf/1907.02057.pdf

Markov Decision Process (MDP)

- A process is observed at time points $t = 0,1, 2 \ldots n$
- At stage t the process is in a state S_t with probability $p(S_t)$, where $S_t \in \boldsymbol{S}$ (assumed finite)
- After observing the state of the process at stage t an action a_t must be chosen, where $a_t \in \boldsymbol{A}$ (assumed finite)

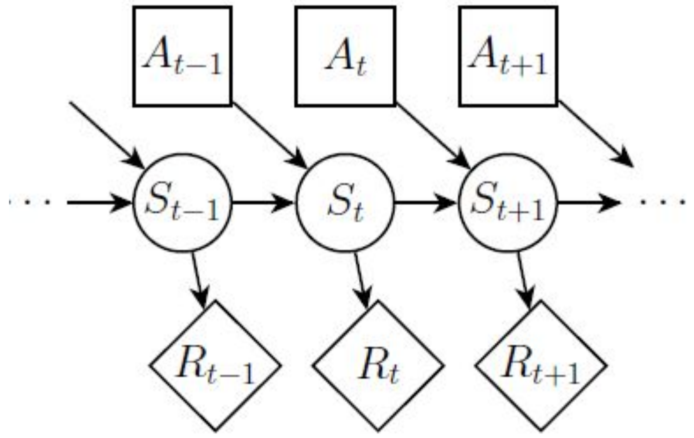
$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \ldots$$

A state S_t is *Markov* if and only if

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

Markov Decision Process (MDP)

- After the action a_t has been taken when state of the process was at state S_t the process goes to state S_{t+1} with probability $p(S_{t+1} \mid a_t , S_t)$



- The reward earned is $R(a_t ,S_t)$ or $R(S_t)$ is earned
- Applications: AI agents in self-driving cars, video games, robot soccer, scheduling energy generation, autonomous flight etc

Markov Decision Process (MDP)

- Both the reward and the transition probabilities are functions only of the last state and the last action taken (Markov property)

$$p(S_{t+1}=s_{t+1} \mid \underbrace{(a_0,s_0), (a_1,s_1), . . , (a_t,s_t)}_{\textit{HISTORY OF STATES AND ACTIONS}}) = p(S_{t+1}=j \mid a_t, s_t)$$

$$R(S_{t,} a_t \mid (a_1,s_1), (a_2,s_2), . \; ..., (a_t, s_t)) = R(S_t, a_t)$$

- The current state captures all relevant information from the history. Once the current state is known, the history may be discarded

Markov Decision Process (MDP)

- For a Markov process having present state *s* and successor state *s'*, the state transition probability is defined by:

$$p_{ss'} = \text{Prob}\,[S_{t+1} = s' \mid S_t = s,\, a_t]$$

- Probability transition matrix (PTM) defines the transition probabilities from all present states to all successor states.

$$\mathcal{P} = \begin{array}{cc} & \textit{to} \\ \textit{from} & \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \end{array}$$

- Each row of the matrix sums to 1.

Markov Process (MP)

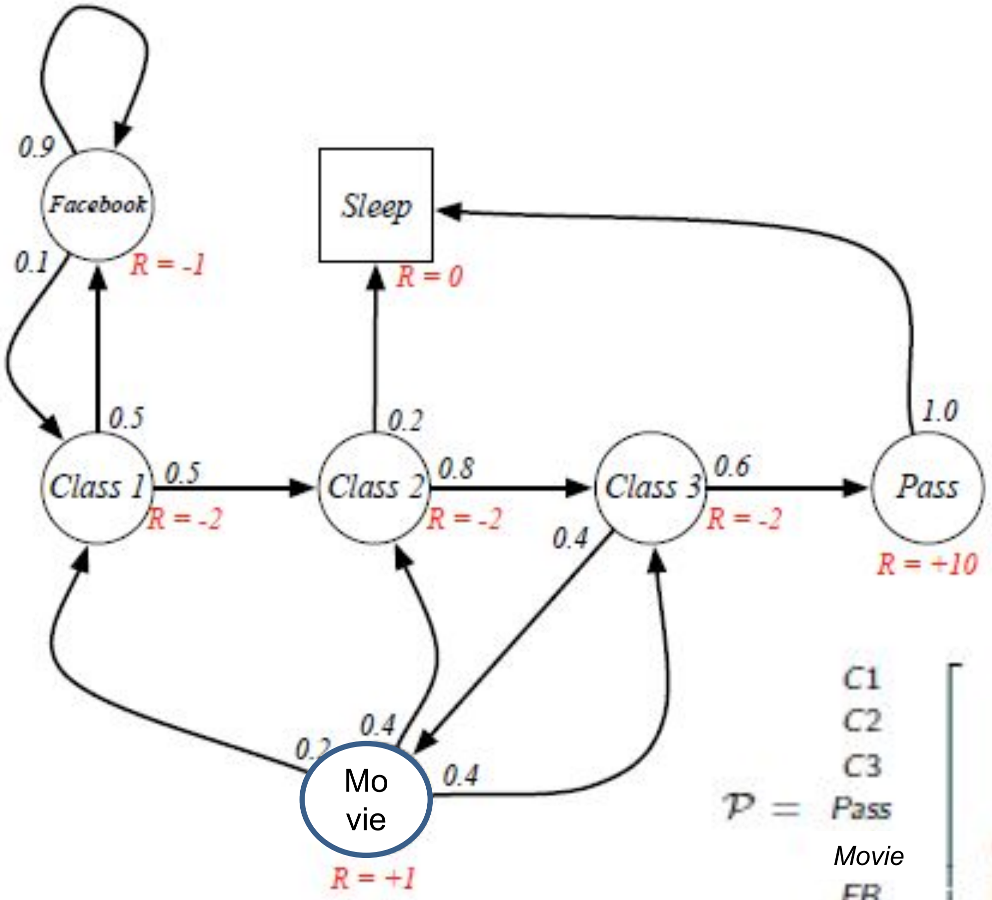
A Markov process is a memoryless random process, i.e. a sequence of random states $S_1, S_2, ...$ with the Markov property.

Definition

A *Markov Reward Process* is 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,
  $\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}\left[R_{t+1} \mid S_t = s\right]$
- γ is a discount factor, $\gamma \in [0, 1]$

Markov Decision Process (MDP)



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

PROBABILITY TRANSITION MATRIX

Markov Decision Process (MDP)

- Upon visiting the sequence of states s_0, s_1, …, with actions a_0, a_1,, …, total payoff is given by

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \dots$$

where, $0 < \gamma < 1$ is a discounting factor

- The goal is to maximize expected total discounted reward

$$\mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots\right]$$

- Note: there could be other criteria, such as average expected reward
- Undiscounted Markov reward process: $\gamma = 1$

Markov Decision Process (MDP)

Most Markov reward and decision processes are discounted because

- Avoids infinite returns in cyclic Markov process
- Uncertainty about the future may not be fully represented if discounting is not allowed
- If the reward is financial, immediate rewards may earn more interest than delayed rewards
- Human behaviour shows preference for immediate reward

Markov Decision Process (MDP)

- A policy is any function mapping from the states to the actions.
- We follow policy *π* if, whenever we are in state *s*, we take action *a* = *π*(*s*).
- Policy defines the behaviour of an agent
- A stationary policy is one which is followed at every stage
- The value function for a policy *π* is the expected sum of discounted rewards upon starting in state *s*, and taking actions according to *π*

$$V^{\pi}(s) = \mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots \mid s_0 = s, \pi\right]$$

Markov Decision Process (MDP)

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

$$v(s) = \mathbb{E}\left[G_t \mid S_t = s\right]$$

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

$$V^{\pi}(s) = \mathrm{E}\big[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots \mid s_0 = s, \pi\big]$$

Markov Decision Process (MDP)

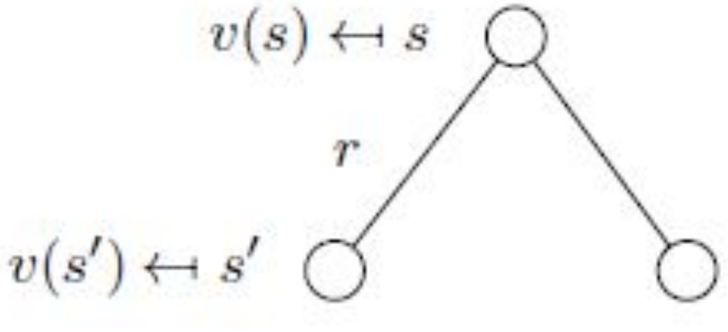
Value function under stationary policy:

- The value function for a stationary policy *π* is the expected sum of discounted rewards upon starting in state *s*, and taking actions according to *π*

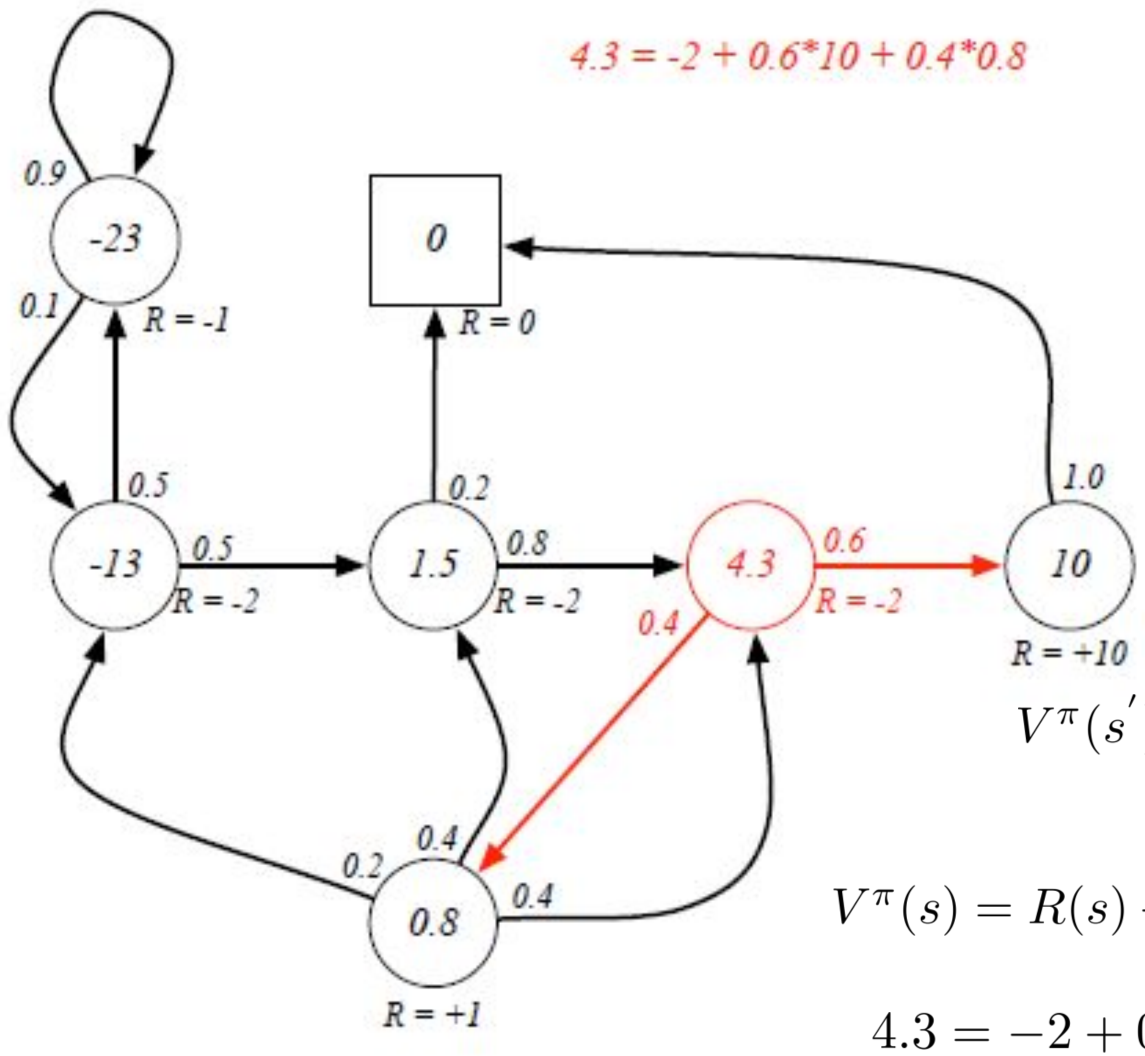
$$V^{\pi}(s) = \mathrm{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots \mid s_0 = s, \pi\right]$$

- The above can be re-written as *Bellman Equations*

$$V^{\pi}(s) = \underbrace{R(s)}_{\text{Immediate reward}} + \gamma \underbrace{\sum_{s' \in S} p_{ss'} V^{\pi}(s')}_{\text{Expected sum Of future discounted rewards}}$$


$v(s) \leftarrow\!\shortmid s$
r
$v(s') \leftarrow\!\shortmid s'$

Markov Decision Process (MDP)



$V^{\pi}(s^{'}) = 10$

$V^{\pi}(s^{'}) = 0.8$

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} p_{ss'} V^{\pi}(s^{'})$$

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

Markov Decision Process (MDP)

- Bellman Equations

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} p_{ss'} V^{\pi}(s')$$

- In matrix form: $V = R + \gamma P V$

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

- Direct Solution: $V = (I - \gamma P)^{-1} R$

v is a column vector with one entry per state, assuming there are 1,2,…, n states

- Complexity is $O(n^3)$ for n states
- For large *n*, iterative methods are used (DP, Monte-Carlo)

Existence of solution

- Direct solution

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

- Computational complexity is O(n^3) for n states
- Direct solution only possible for small MRPs

Existence of solution

- § We need to show that $(\mathbf{I} - \gamma\mathcal{P})$ is invertible and for that we will use the following result from linear algebra - The inverse of a matrix exists if and only if all its eigenvalues are non-zero.
- § For a stochastic matrix (row sum equal to 1 and all entries are ≥ 0), the largest eigenvalue is 1.

Proof

As $\mathcal{P}$ is a stchoastic matrix, $\mathcal{P}\mathbb{1} = \mathbb{1}$ where $\mathbb{1} = [1, 1, \cdots 1]^T$. This means 1 is an eigenvalue of $\mathcal{P}$.
Now, lets suppose $\exists \lambda > 1$ and non-zero x such that $\mathcal{P}\mathrm{x} = \lambda\mathrm{x}$.
Since the rows of $\mathcal{P}$ are non-negative and sum to 1, each element of vector $\mathcal{P}\mathrm{x}$ is a convex combination of the components of the vector x.
A convex combination can't be greater than x_{max}, the largest component of x.
However, as $\lambda > 1$, at least one element (λx_{max}) in the R.H.S. (*i.e.*, in $\lambda\mathrm{x}$) is greater than x_{max}. This is a contradiction and so $\lambda > 1$ is not possible.

- § So the largest eigenvalue of $\mathcal{P}$ is 1.

Existence of solution

For all eigenvalues λ_i of a square matrix $\mathbf{A}$ and corresponding eigenvectors $\mathbf{v}_i$ such that $\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{v}_i$,

$$\text{eig}(\mathbf{I} + \gamma \mathbf{A}) = 1 + \gamma \lambda_i \;\; [\gamma \text{ is any scalar}]$$

Proof:

$$\begin{aligned} \mathbf{A}\mathbf{v}_i &= \lambda_i \mathbf{v}_i \\ \gamma \mathbf{A}\mathbf{v}_i &= \gamma \lambda_i \mathbf{v}_i \\ \mathbf{v}_i + \gamma \mathbf{A}\mathbf{v}_i &= \mathbf{v}_i + \gamma \lambda_i \mathbf{v}_i \\ (\mathbf{I} + \gamma \mathbf{A})\mathbf{v}_i &= (1 + \gamma \lambda_i)\mathbf{v}_i \end{aligned}$$

§ So the smallest eigenvalue of $(\mathbf{I} - \gamma \mathcal{P})$ is $1 - \gamma$. For $\gamma < 1$ which is > 0. And hence, $(\mathbf{I} - \gamma \mathcal{P})$ is invertible.

Markov Decision Process (MDP)

Optimal Value function

- The maximum expected sum of discounted rewards upon starting in state *s*, and taking actions according to *π*

$$
\begin{aligned}
V^*(s) &= \max_\pi V^\pi(s) \\
&= \max_\pi R(s) + \gamma \sum_{s' \in S} p_{ss'} V^\pi(s') \\
&= R(s) + \max_\pi \gamma \sum_{s' \in S} p_{ss'} V^\pi(s')
\end{aligned}
$$

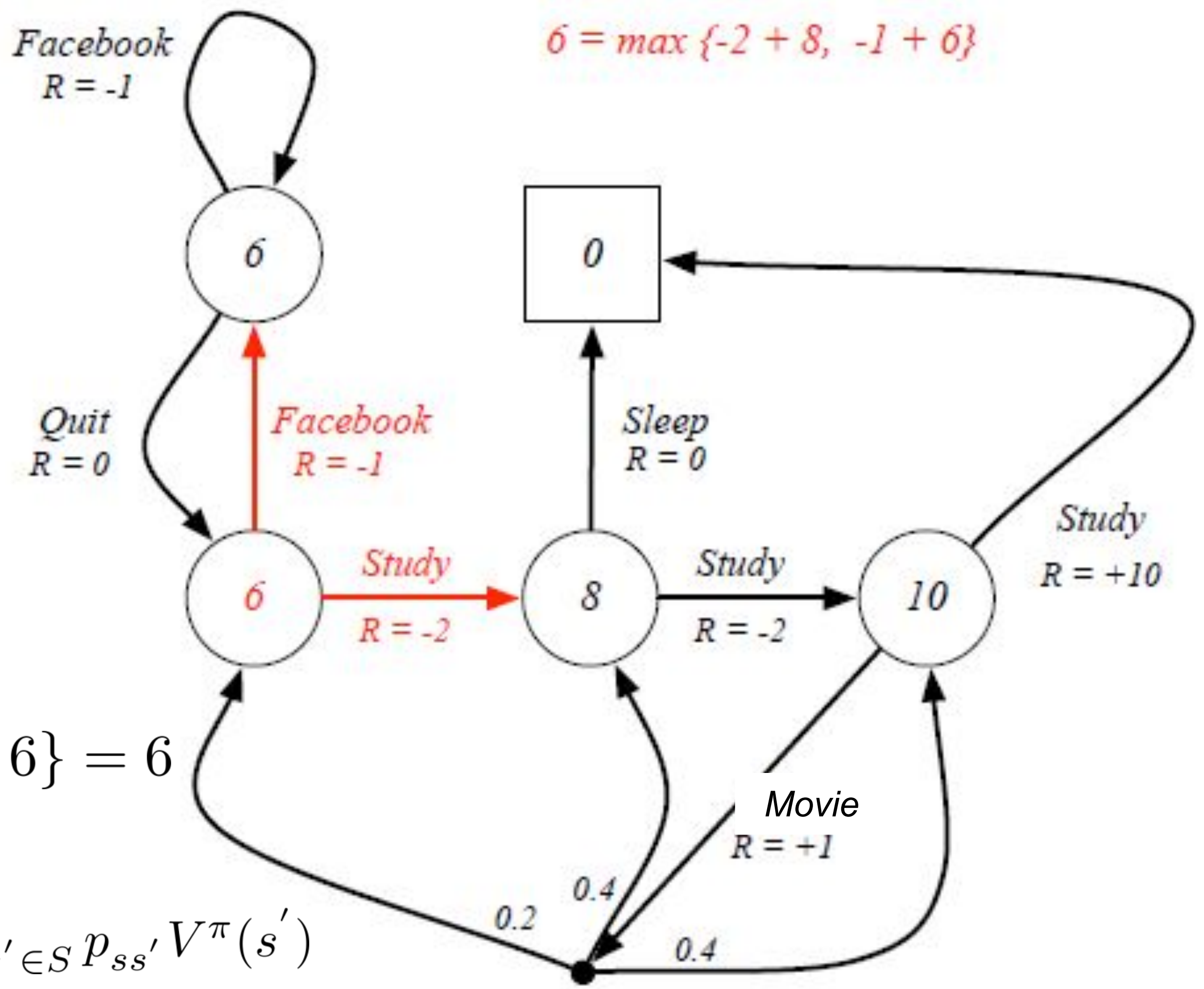
Optimal policy

- The policy function *π* that will maximize expected sum of discounted rewards upon starting in state *s*

$$\pi^*(s) = \arg\max_\pi \gamma \sum_{s' \in S} p_{ss'} V^\pi(s')$$

Note: The maximum value earned is: $V^{\pi^*} = V^*$

Markov Decision Process (MDP)



Optimal Value function

$$V^*(s) = \max_\pi V^\pi(s)$$

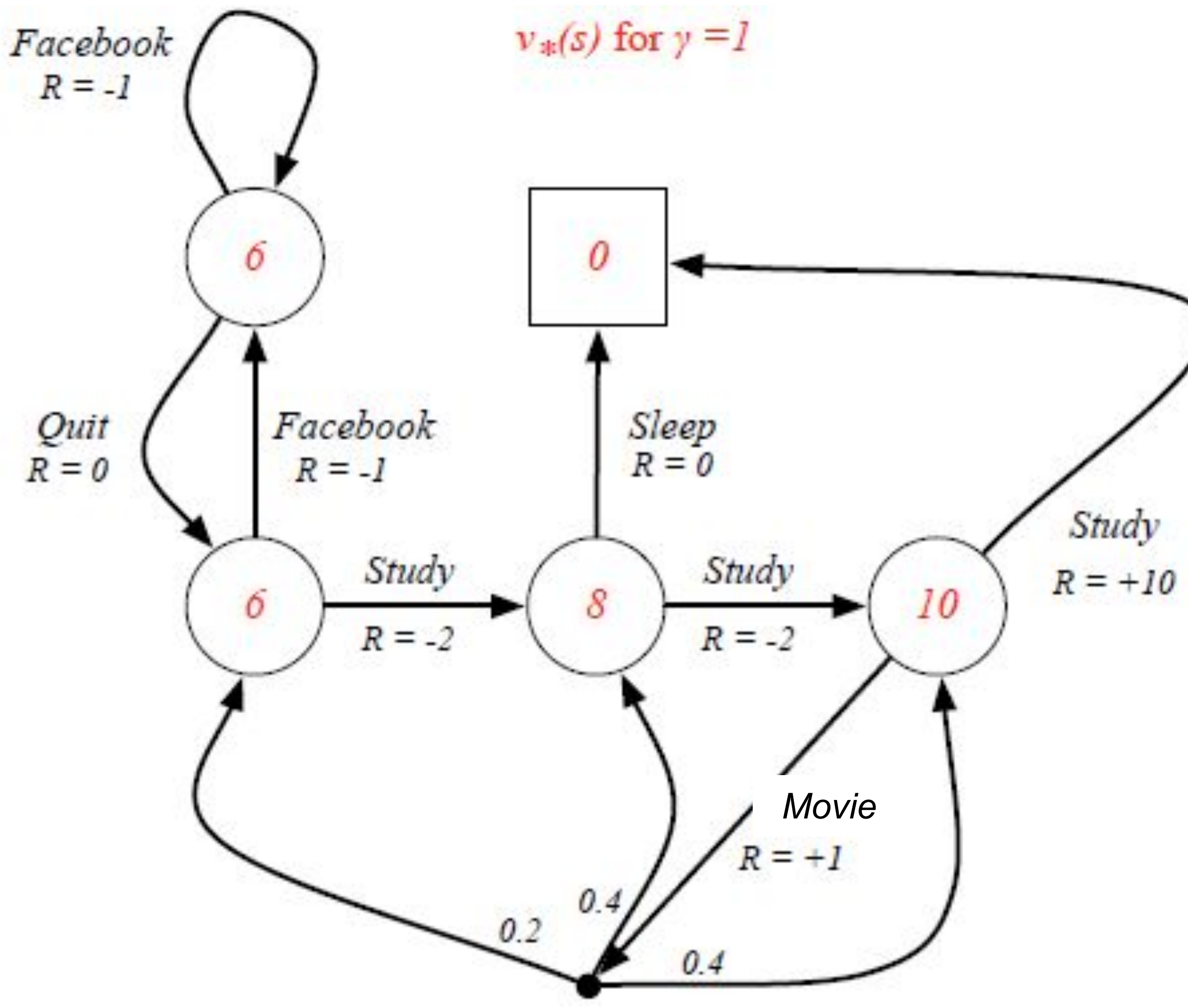
$$V^*(class1)$$
$$= \max\{-2+8, -1+6\} = 6$$

Optimal policy

$$\pi^*(s) = \arg\max_\pi \gamma \sum_{s' \in S} p_{ss'} V^\pi(s')$$
$$\pi^*(class1) = study$$

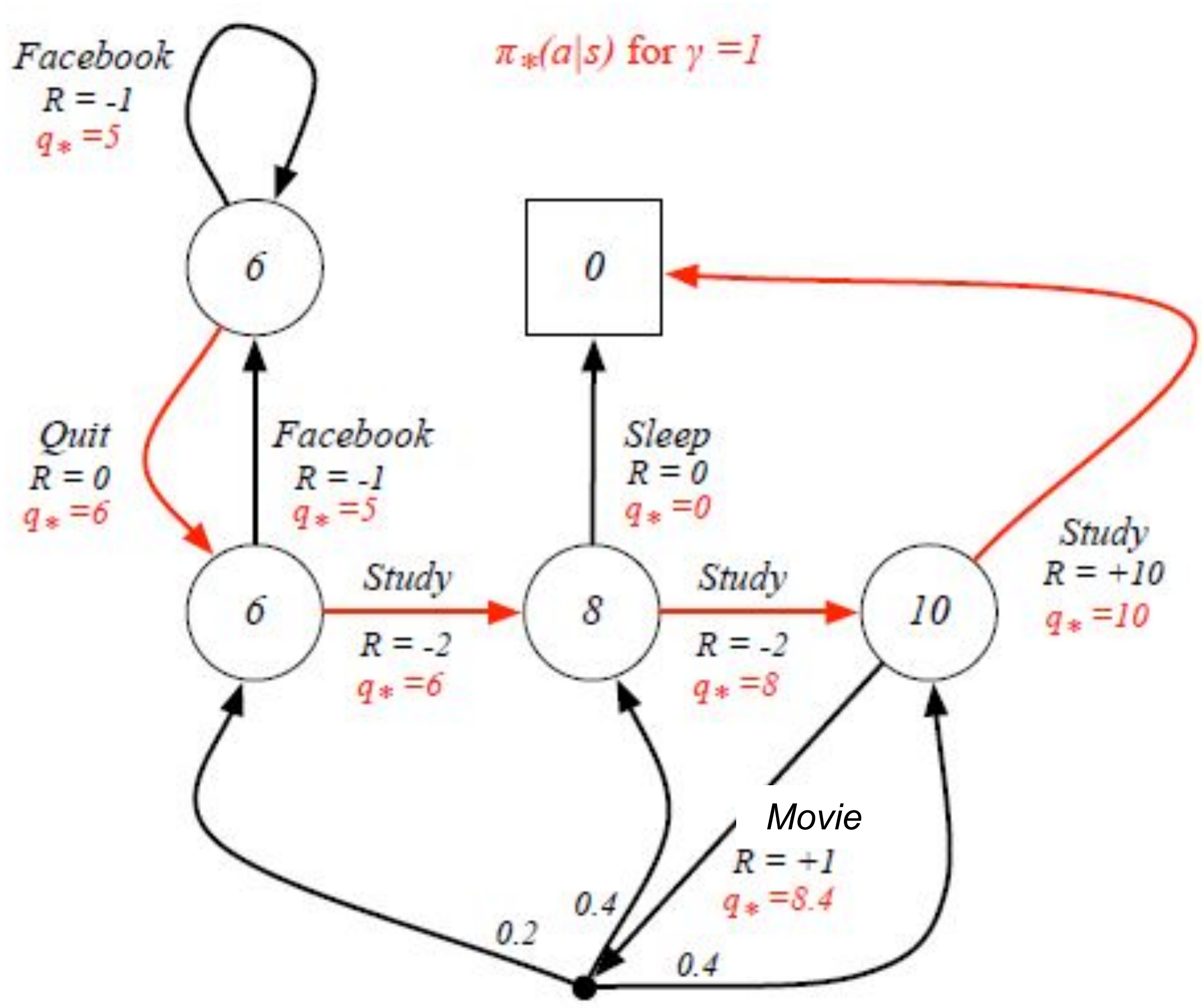
Markov Decision Process (MDP)

Optimal Value functions for each state



Markov Decision Process (MDP)

Optimal policy for each state



Value Function

By value function of a state, we mean the expected return if we start in that state or state-action pair, and then act according to a particular policy forever after

$$V^{\pi}(s) = \underset{\tau \sim \pi}{\mathrm{E}} \left[R(\tau) \,|\, s_0 = s \right]$$

$$V(s) := R(s) + \max_{\pi} \gamma \sum_{s' \in S} p_{ss'} V(s')$$

https://spinningup.openai.com/en/latest/spinningup/rl_intro.html#bellman-equations

Solution of MDPs

Value iteration:

Consider only MDPs with finite state and action spaces. The value iteration algorithm.

- For each state s, initialize *V*(*s*):=0
- Repeat until convergence:

$$V(s) := R(s) + \max_{\pi} \gamma \sum_{s' \in S} p_{ss'} V(s')$$

- Value iteration will cause *V* to converge to *V* *
- Having found *V** we can find optimal policy

$$\pi^*(s) = \arg\max_{\pi} \gamma \sum_{s' \in S} p_{ss'} V^*(s')$$

Example: Sequential Decision

Suppose that an agent is situated in the 4×3 grid. Beginning in the start state, it must choose an action at each time step. The actions terminate when the agent reaches one of the goal states, which has rewards +1 or –1. All other states have a reward of –0.04.

3	-0.4	-0.4	-0.4	+1
2	-0.4	**forbi den**	-0.4	-1
1	**start**	-0.4	-0.4	-0.4
	1	2	3	4

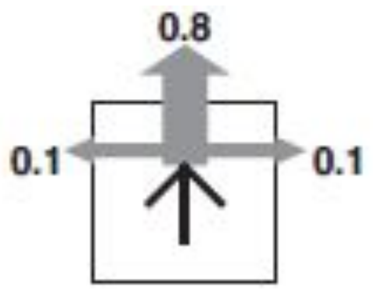
The actions available to the agent in each state are given by

A(s)= { *Up*, *Down*, *Left*, and *Right* }

Example: Sequential Decision

An action moves the agent to intended direction with probability 0.8, and with probability 0.2 moves at right angles to the intended direction. From the start square (1,1), the action *Up* moves to (1,2) with probability 0.8, but with probability 0.1, it moves right to (2,1), and with probability 0.1, it moves left, bumps into the wall, and stays in (1,1).

-0.4	-0.4	-0.4	+1
-0.4	**forbi den**	-0.4	-1
start	-0.4	-0.4	-0.4



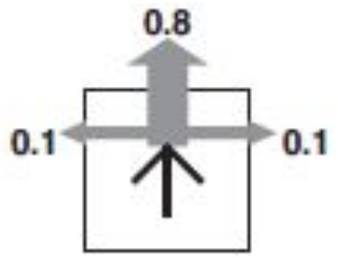
Transition model: P(s′ | s, a) denotes the probability of reaching state s′ if action a is done in state s.

Example: Sequential Decision

Solution :

- Any fixed action sequence (fixed policy) wouldn't work
- A solution must specify what the agent should do for *any* state that the agent might reach - specify policy π(s).
- An **optimal policy** is a policy that yields the highest expected value. We use π^* to denote an optimal policy

3	-0.4	-0.4	-0.4	+1
2	-0.4	forbi den	-0.4	-1
1	start	-0.4	-0.4	-0.4
	1	2	3	4



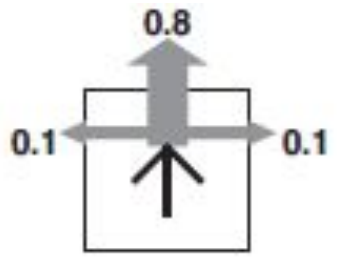
Example: Sequential Decision

Solution :

- Suppose agent is in state (1,1). What is the optimal policy?

$$
\begin{aligned}
U(1,1) = -0.04 + \gamma \max[\ & 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), && (Up) \\
& 0.9U(1,1) + 0.1U(1,2), && (Left) \\
& 0.9U(1,1) + 0.1U(2,1), && (Down) \\
& 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)\ ]. && (Right)
\end{aligned}
$$

3	-0.4	-0.4	-0.4	+1
2	-0.4	forbi den	-0.4	-1
1	start	-0.4	-0.4	-0.4
	1	2	3	4



Value iteration

Idea: Repeatedly update an estimate of the optimal value function according to Bellman optimality equation

Initialize an estimate for the value function arbitrarily

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

Repeat, for each state ***s*** update:

$$\hat{V}(s) \leftarrow \underbrace{R(s)}_{\text{Immediate Reward in state s}} + \underbrace{\max_{a \in A}}_{\text{All possible actions in state s}} \gamma \sum_{s' \in S} \underbrace{p_{ss'}}_{\text{Elements of Probability transition matrix}} \hat{V}(s')$$

Stop when: $|\hat{V}^{k+1}(s) - \hat{V}^{k}(s)| < \epsilon$

Value iteration Algorithm

function VALUE-IT (.) **returns** a utility function

Inputs: states S, actions A(s), transition model P(s′ | s, a), rewards R(s), discount γ, maximum error allowed in the utility of any state ε, maximum change in the utility of any state in an iteration δ

```
Initialize: $V(s) \leftarrow 0, \forall s$
Repeat
        $V(s) \leftarrow \hat{V}(s); \delta = 0$
    For each state s in S, do
        $\hat{V}(s) \leftarrow R(s) + \max_{a \in A} \gamma \sum_{s' \in S} p_{ss'} V(s')$
    If  $|\hat{V}(s) - V(s)| > \delta$, then  $\delta \leftarrow |\hat{V}(s) - V(s)|$
until  $\delta < \epsilon(1-\gamma)/\gamma$
Return $V$
```

Convergence of Value iteration

Theorem: Value iteration converges to optimal value $\hat{V} \to V^*$

Proof: For any estimate of the value function $\hat{V}$, we define the Bellman backup operator $B : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$

$$B\hat{V}(s) = R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)\hat{V}(s')$$

Bellman operator is a *contraction*. For any value function estimates V_1, V_2

$$\max_{s \in \mathcal{S}} |BV_1(s) - BV_2(s)| \le \gamma \max_{s \in \mathcal{S}} |V_1(s) - V_2(s)|$$

Since $BV^\star = V^\star$ (the contraction property also implies existence and uniqueness of this fixed point), we have:

$$\max_{s \in \mathcal{S}} \left| B\hat{V}(s) - V^\star(s) \right| \le \gamma \max_{s \in \mathcal{S}} \left| \hat{V}(s) - V^\star(s) \right| \implies \hat{V} \to V^\star$$

Convergence of Value iteration

Proof of contraction property:

$$
\begin{aligned}
&|BV_1(s) - BV_2(s)| \\
&= \gamma \left| \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a) V_1(s') - \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a) V_2(s') \right| \\
&\leq \max_{a \in \mathcal{A}} \left| \sum_{s' \in \mathcal{S}} P(s'|s,a) V_1(s') - \sum_{s' \in \mathcal{S}} P(s'|s,a) V_2(s') \right| \\
&= \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a) \, |V_1(s') - V_2(s')| \\
&\leq \gamma \max_{s \in \mathcal{S}} |V_1(s) - V_2(s)|
\end{aligned}
$$

(because *P*(*s'* |*s; a*) are non-negative and sum to one)

Using the property:

$$
|\max_x f(x) - \max_x g(x)| \leq \max_x |f(x) - g(x)|
$$

More on Value iteration

How many iterations will it take to find optimal policy?

Assume rewards in [0 ; $R_{\max}$], then

$$V^\star(s) \le \sum_{t=1}^{\infty} \gamma^t R_{\max} = \frac{R_{\max}}{1-\gamma}$$

Because: $V^\pi(s) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right]$

Let V^k be value after *k*-th iteration

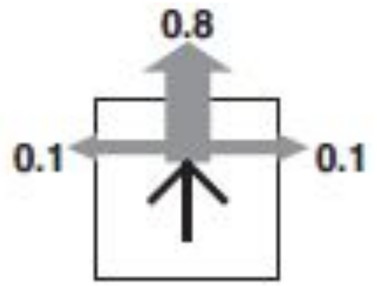
$$\max_{s \in \mathcal{S}} |V^k(s) - V^\star(s)| \le \frac{\gamma^k R_{\max}}{1-\gamma}$$

Thus we have linear convergence to optimal value function

Implementation of Value iteration

Example:

(1,3) -0.4	(2,3) -0.4	(3,3) -0.4	(4,3) +1
(1,2) -0.4	**forbi den**	(3,2) -0.4	(4,2) -1
(1,1) -0.4	(2,1) -0.4	(3,1) -0.4	(4,1) -0.4



Upon taking an action, the Agent moves toward intended direction with probability 0.8, and with probability 0.2 moves at right angles to the intended direction.

- Define *states* as a 2D *grid,*
- Define actions as A(s)= {*Up*, *Down*, *Left*, *Right* }
- Define *policy* as a dictionary of {state:action} pairs,
- Define *value* function V as a dictionary of {state:number} pairs
- Define *transition* model *T(s, a)* return a list of *(p, s') pairs.*
- *Alternately*, define *transition* model as *T(s, a, s')* for each *state/action/state triplet*
- Define *rewards* as a 2D matrix *R :* R[x, y] = grid[x][y]

Implementation of Value iteration

```
def value_iteration( R, T, gamma,  epsilon=0.001):
V1 = dict([(s, 0) for s in states])
while True:
        V = V1.copy()
        delta = 0
        for s in states:
            V1[s] = R(s) + gamma * max([sum([p * V[s1] for (p, s1) in T(s, a)])
                                        for a in A(s)])
            delta = max(delta, abs(V1[s] - V[s]))
        if delta < epsilon * (1 - gamma) / gamma:
             return V
```

```
Grid ([[-0.04, -0.04, -0.04, +1],
       [-0.04, None,  -0.04, -1],
       [-0.04, -0.04, -0.04, -0.04]],
      terminals=[(3, 2), (3, 1)])
```

def best_policy(V) : $\pi^*(s) = \arg\max_\pi \gamma \sum_{s' \in S} p_{ss'} V^*(s')$

```
#Given an MDP and a utility function V, determine the best policy
   pi = {}
   for s in states:
       pi[s] = argmax(A(s), a: gamma * sum([p * V[s1] for (p, s1) in T(s, a)]))
   return pi
```

Policy iteration

The policy iteration algorithm alternates the following two steps, beginning from some initial policy π_0:

(i) **Policy evaluation:** given policy π, calculate $V := V^{\pi}$ (utility of each state if π were to be executed)

(ii) **Policy improvement:** Calculate a new policy using

$$\pi^*(s) := \arg\max_{a \in A} \gamma \sum_{s' \in S} P(s'|s,a) V^*(s')$$

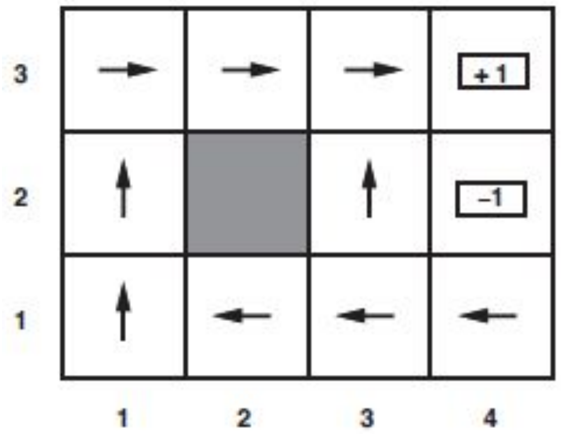
The algorithm terminates when the policy improvement step yields no change in the utilities.

Policy iteration: $\pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 \rightarrow V^{\pi_1} \rightarrow \pi_2 \rightarrow V^{\pi_2} \rightarrow \pi_3 \rightarrow V^{\pi_3} \rightarrow \ldots \rightarrow \pi^*$

Note: The second step repeatedly computes the value function for the current policy, and then updates the policy using the current value function (Greedy update)

Policy iteration

Example: Suppose π is the policy as given below



π(1, 1)=Up, π(1, 2)=Up, π(1, 2)=Up
π(1, 3)=Right, π(2, 3)=Right, π(3, 3)=Right,
π(2, 1)=Left, π(3, 1)=Left, π(4, 1)=Left

$$U_i(1,1) = -0.04 + 0.8U_i(1,2) + 0.1U_i(1,1) + 0.1U_i(2,1)\,,$$
$$U_i(1,2) = -0.04 + 0.8U_i(1,3) + 0.2U_i(1,2)\,,$$

Policy evaluation: given policy π, calculate

$$V(s) = R(s) + \gamma \sum_{s' \in S} p_{ss'} V(s') = R(s) + \gamma \sum_{s' \in S} P(s'|s,a) V(s')$$

Note: This is a system of *linear equations* with *n* states and *n* unknowns, because the “max” operator has been removed.

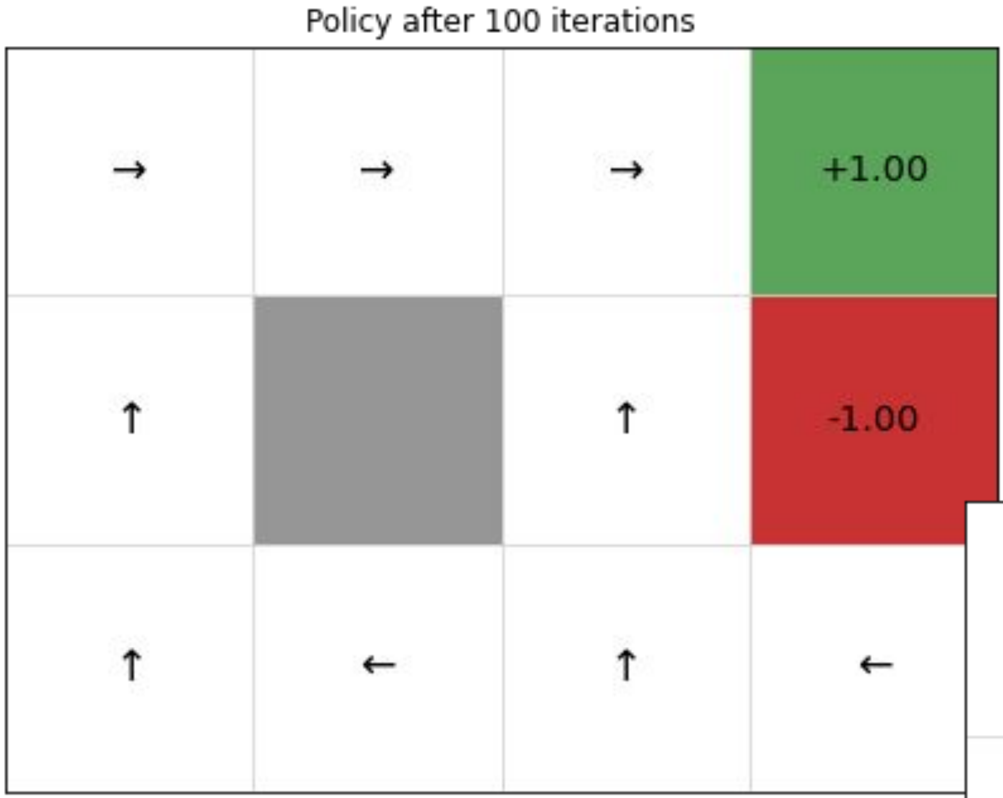
Policy iteration Algorithm

```
function POLICY-IT ( . ) returns a policy
Inputs: states S, actions A(s), transition model P(s′|s,a), rewards R(s), discount factor γ,
Local variable: V : stores utilities for states in S
                π : policy vector indexed by state
Initialize: $V(s) = 0, \forall s$ , Randomize $\pi(s)$
Repeat
        V ← POLICY-EVALUATION(π)
        unchanged?←true
    For each state s in S, do
        If   $\max_{a \in A(s)} \sum_{s'} P(s'|s,a)V(s') > \sum_{s'} P(s'|s,a)V(s')$
            then $\pi^*(s) \leftarrow \arg\max_{a \in A(s)} \sum_{s' \in S} P(s'|s,a)V(s')$
            unchanged?←false
Until unchanged?
Return π
```

Implementation of Policy iteration

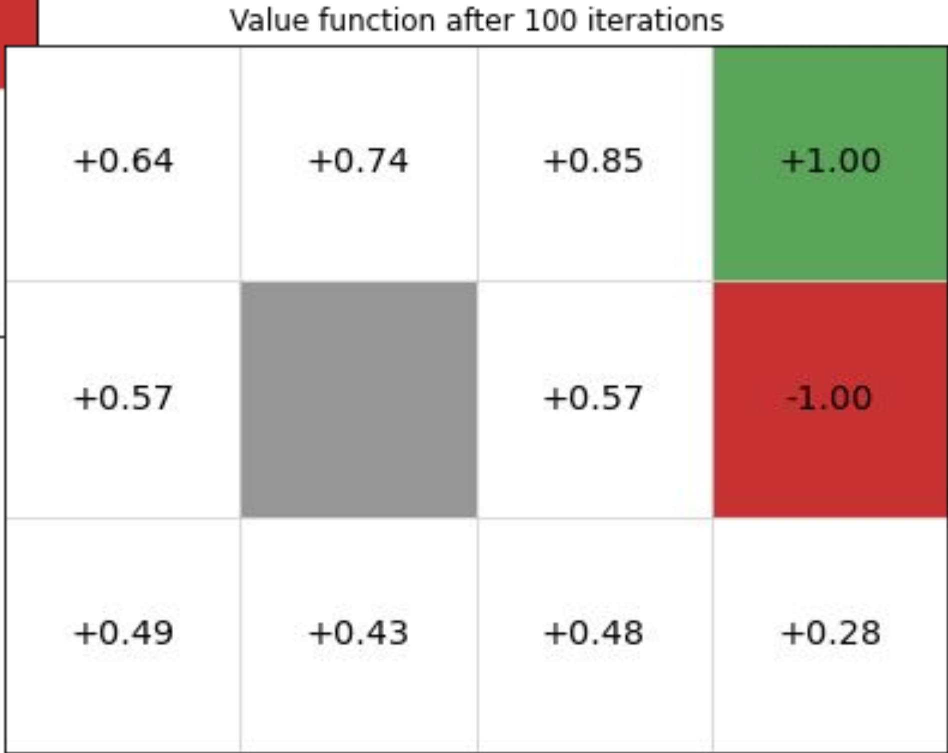
```
def policy_iteration(states, R, T, gamma):
    V = dict([(s, 0) for s in states])
    pi = dict([(s, random.choice(A(s))) for s in states])
    while True:
        V = policy_evaluation(pi, V)
        unchanged = True
        for s in states:
            a = argmax(A(s), a:  gamma * sum([p * V[s1] for (p, s1) in T(s, a)]))
            if a != pi[s]:
                pi[s] = a
                unchanged = False
        if unchanged:
            return pi

def policy_evaluation(pi, V, k=30):
  for i in range(k):
        for s in states:
            V[s] = R(s) + gamma * sum([p * V[s] for (p, s1) in T(s, pi[s])])
    return V
```

Solution to Grid world problem



Policy after 100 iterations

→	→	→	+1.00
↑		↑	-1.00
↑	←	↑	←



Value function after 100 iterations

+0.64	+0.74	+0.85	+1.00
+0.57		+0.57	-1.00
+0.49	+0.43	+0.48	+0.28

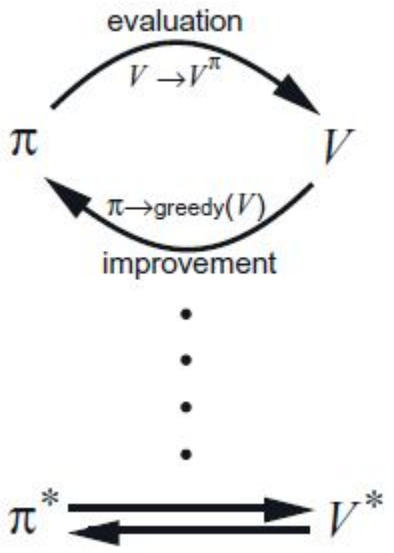
Value vs Policy iteration

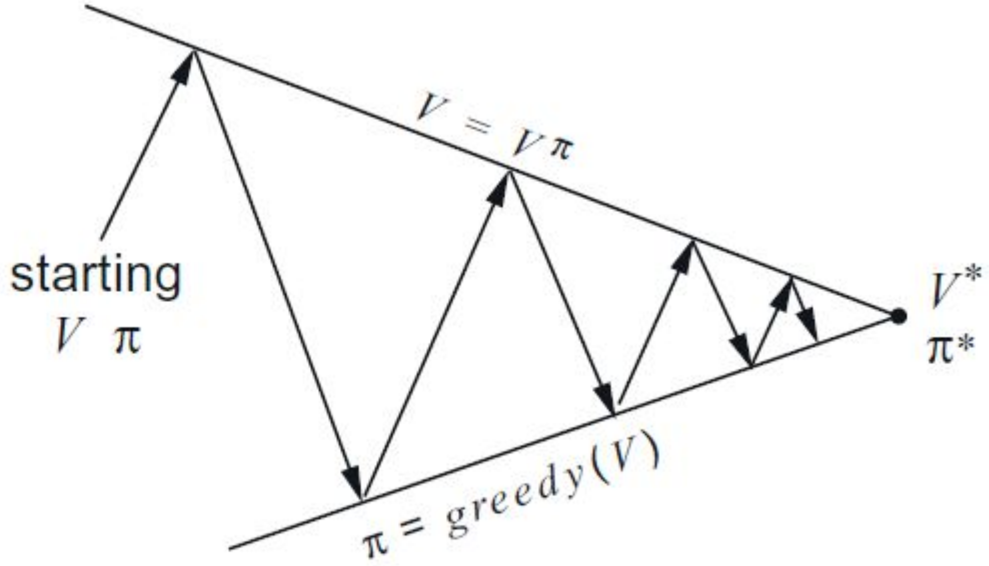
- Value iteration works by producing successive approximations of the optimal value function.
- Each iteration can be performed in $O(|A||S|^2)$ steps (A = actions set, S = states)
- The *number* of iterations in Value iteration can grow exponentially in the discount factor, because a larger γ implies that a longer sequence of future rewards has to be taken into account
- The complexity of value iteration is linear in number of actions, and quadratic in the number of states

Value vs Policy iteration

- In practice policy iteration converges much faster, but each evaluation step is expensive.
- Each iteration can be performed in $O(|A||S|^2 + |S|^3)$steps
- The *number* of iterations in Policy iteration can grow extremely large when the problem size grows

Value vs Policy iteration





The *policy evaluation* step estimates V^π , the policy's performance.
The *policy improvement* step improves the policy π based on the estimates in V^π

Modified policy iteration (MPI) strikes a middle ground between value and policy iteration

(state)Value-function

$V^{\pi}(s)$, where s is a state, is the expected value of following
policy π in state s.

The **Optimal Value Function**, gives the expected return if you start in state and always act according to the *optimal* policy in the environment:

$$V^*(s) = \max_{\pi} \mathop{\mathrm{E}}_{\tau \sim \pi} [R(\tau) \,|s_0 = s]$$

Action-Value-function

$Q^{\pi}(s, a)$, where a is an action and s is a state, is the expected value of doing a in state s, then following policy π.

The *Optimal action-value function* $Q^{\pi}(S,A)$ gives optimal the expected return if you start in state s, take an arbitrary action a (which may not have come from the policy), and then forever after act according to optimal policy π:

$$Q^{*}(s,a) = \max_{\pi} \mathop{\mathrm{E}}_{\tau \sim \pi} \left[R(\tau) \,|\, s_0 = s, a_0 = a \right]$$

Bellman equations

The Bellman equations for the on-policy value functions are

$$V^{\pi}(s) = \mathop{\mathrm{E}}_{\substack{a\sim\pi \\ s'\sim P}} \left[r(s,a) + \gamma V^{\pi}(s') \right],$$

$$Q^{\pi}(s,a) = \mathop{\mathrm{E}}_{s'\sim P} \left[r(s,a) + \gamma \mathop{\mathrm{E}}_{a'\sim\pi} \left[Q^{\pi}(s',a') \right] \right],$$

This is the expected return if you start in state s and always act according to optimal policy π:

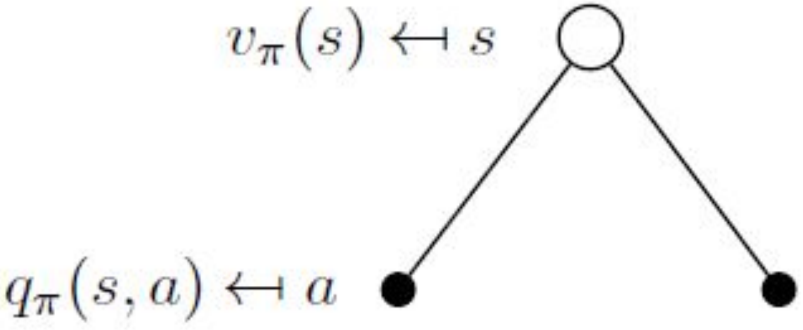
Q-function & V-function

Q^π and V^π can be defined mutually recursively:

$$
\begin{aligned}
V^\pi(s) &= Q^\pi(s, \pi(s)) \\
Q^\pi(s,a) &= \sum_{s'} P(s'|a,s)\left(r(s,a,s') + \gamma V^\pi(s')\right)
\end{aligned}
$$

Example

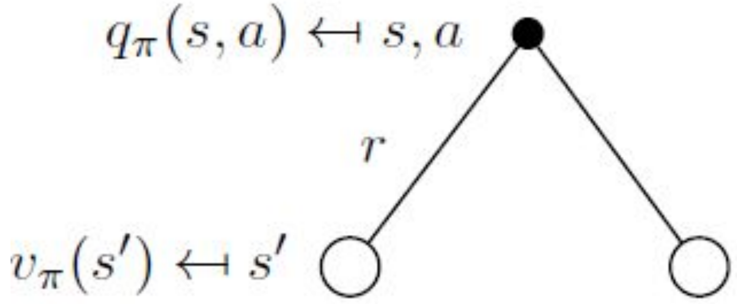
for V^π



$v_\pi(s) \leftarrow\!\shortmid s$
$q_\pi(s,a) \leftarrow\!\shortmid a$

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

for Q^π



$q_\pi(s,a) \leftarrow\!\shortmid s,a$
r
$v_\pi(s') \leftarrow\!\shortmid s'$

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

Q-function

There are two key connections between the (state) value function and the action-value function

$$V^\pi(s) = \mathop{\mathrm{E}}_{a\sim\pi}\left[Q^\pi(s,a)\right],$$

and

$$V^*(s) = \max_a Q^*(s,a).$$

Proof: put $Q^\pi(s,a) = \mathop{\mathrm{E}}_{\tau\sim\pi}\left[R(\tau)\,|s_0 = s, a_0 = a\right]$

In $V^\pi(s) = \mathop{\mathrm{E}}_{a\sim\pi}\left[Q^\pi(s,a)\right],$

and we get $Q^*(s,a) = \max_\pi \mathop{\mathrm{E}}_{\tau\sim\pi}\left[R(\tau)\,|s_0 = s, a_0 = a\right]$

Optimal Q-Function and Optimal Action

The *Optimal action-value function* $Q^{\pi}(S,A)$ gives optimal the expected return if you start in state s, take an arbitrary action a (which may not have come from the policy), and then forever after act according to optimal policy π:

The optimal policy in s will select whichever action maximizes the expected return from starting in s. As a result, if we have Q*, we can directly obtain the optimal action, a*(s), via

$$a^*(s) = \arg\max_a Q^*(s,a).$$

Bellman equations

The Bellman equations for the optimal value functions are

$$V^*(s) = \max_a \mathop{\mathrm{E}}_{s' \sim P} \left[r(s,a) + \gamma V^*(s') \right],$$
$$Q^*(s,a) = \mathop{\mathrm{E}}_{s' \sim P} \left[r(s,a) + \gamma \max_{a'} Q^*(s',a') \right].$$

The crucial difference between the Bellman equations for the on-policy value functions and the optimal value functions, is the absence or presence of the \max over actions

Bellman equations

Q* and V* can be defined mutually recursively

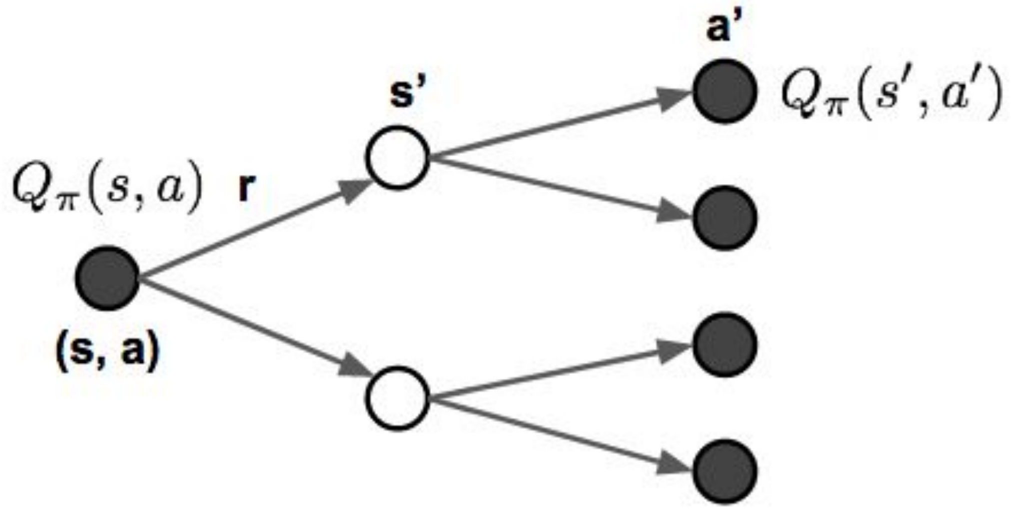
$$\begin{aligned} Q^*(s,a) &= \sum_{s'} P(s'|a,s)\left(r(s,a,s') + \gamma V^*(s')\right) \\ V^*(s) &= \max_a Q^*(s,a) \\ \pi^*(s) &= \arg\max_a Q^*(s,a) \end{aligned}$$

The crucial difference between the Bellman equations for the on-policy value functions and the optimal value functions, is the absence or presence of the \max over actions

Bellman equations

Q* and V* can be defined mutually recursively

$$
\begin{aligned}
Q^*(s,a) &= \sum_{s'} P(s'|a,s)\left(r(s,a,s') + \gamma V^*(s')\right) \\
V^*(s) &= \max_a Q^*(s,a) \\
\pi^*(s) &= \arg\max_a Q^*(s,a)
\end{aligned}
$$



Bellman equations

The recursive update process can be further decomposed to be equations built on both state-value and action-value functions. As we go further in future action steps, we extend V* and Q* alternatively by following the policy π

.

$$V_*(s) = \max_{a \in \mathcal{A}} Q_*(s, a)$$

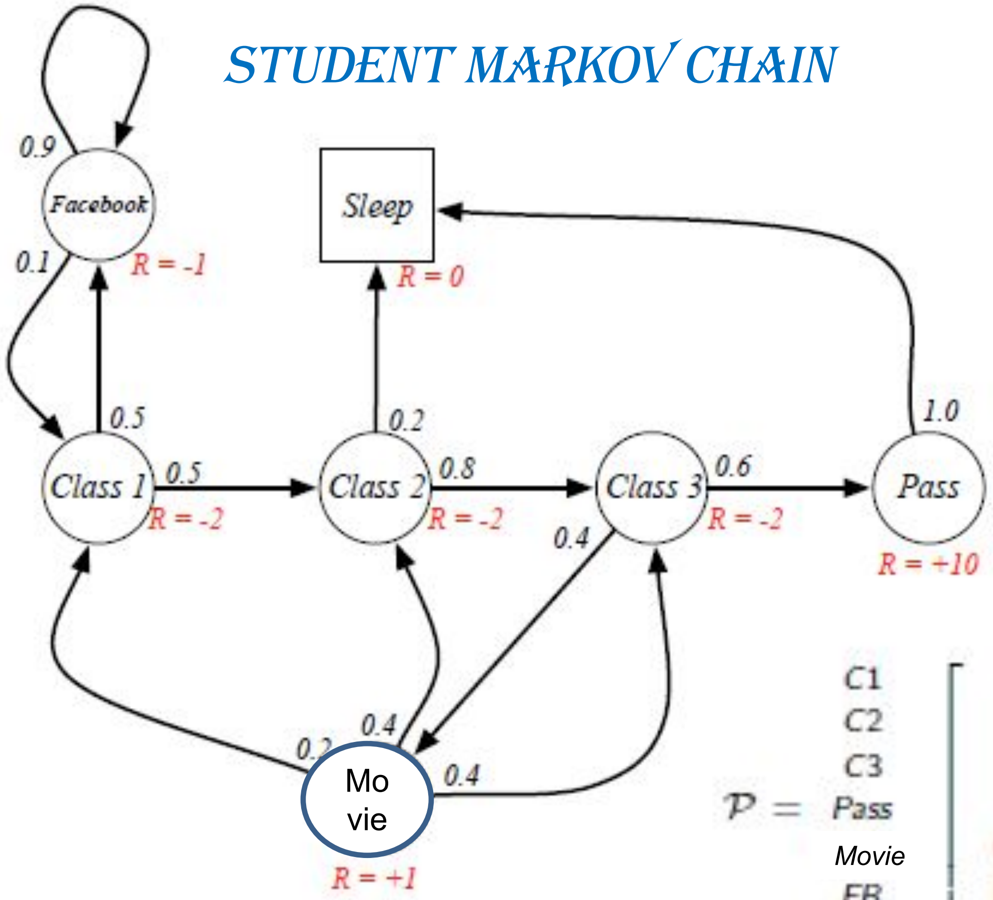
$$Q_*(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P^a_{ss'} V_*(s')$$

$$V_*(s) = \max_{a \in \mathcal{A}} \left(R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P^a_{ss'} V_*(s')\right)$$

$$Q_*(s, a) = R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P^a_{ss'} \max_{a' \in \mathcal{A}} Q_*(s', a')$$

Example:

STUDENT MARKOV CHAIN

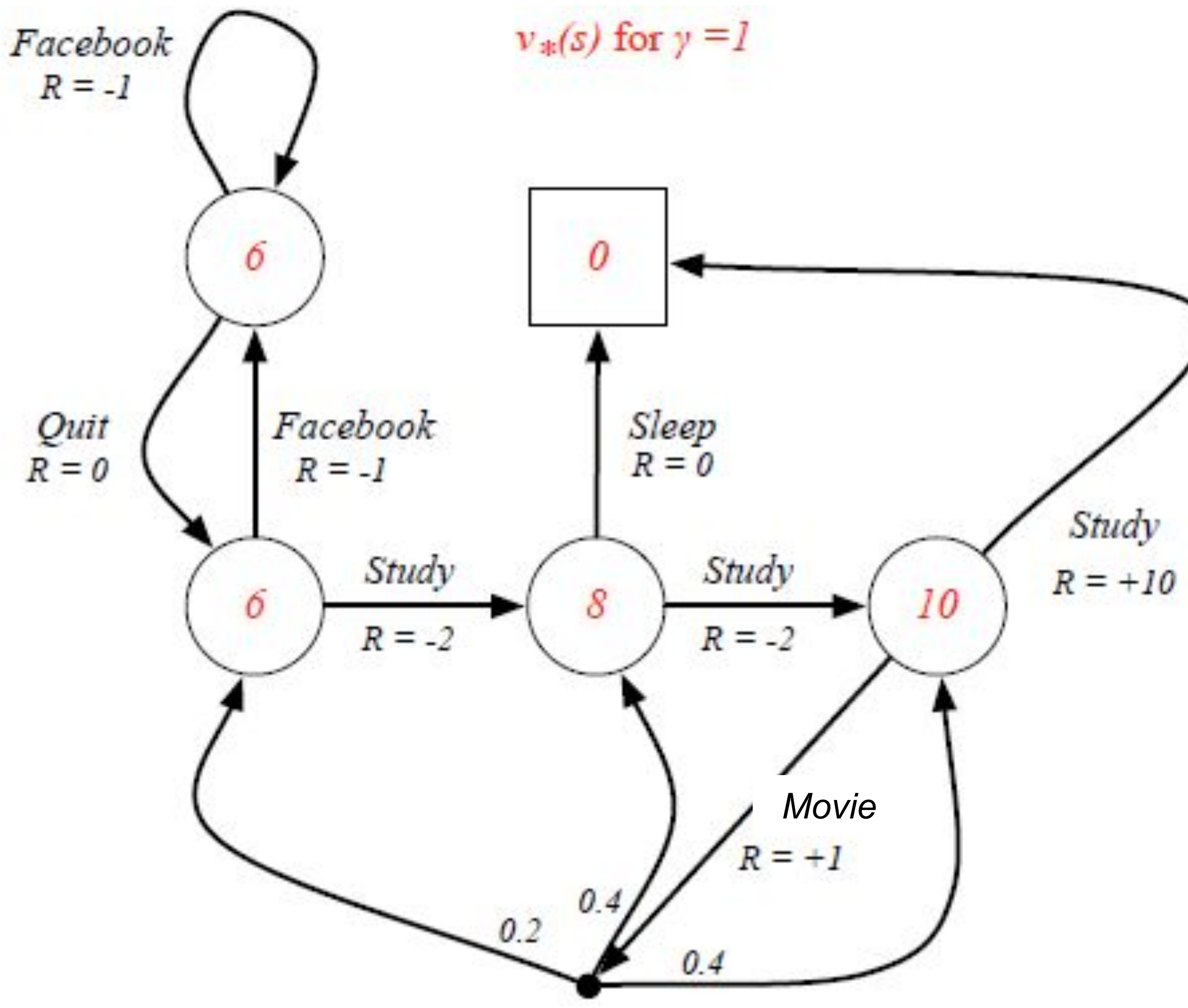


$$\mathcal{P} = \begin{array}{c|ccccccc} & C1 & C2 & C3 & Pass & Movie & FB & Sleep \\ \hline C1 & & 0.5 & & & & 0.5 & \\ C2 & & & 0.8 & & & & 0.2 \\ C3 & & & & 0.6 & 0.4 & & \\ Pass & & & & & & & 1.0 \\ Movie & 0.2 & 0.4 & 0.4 & & & & \\ FB & 0.1 & & & & & 0.9 & \\ Sleep & & & & & & & 1 \end{array}$$

PROBABILITY TRANSITION MATRIX

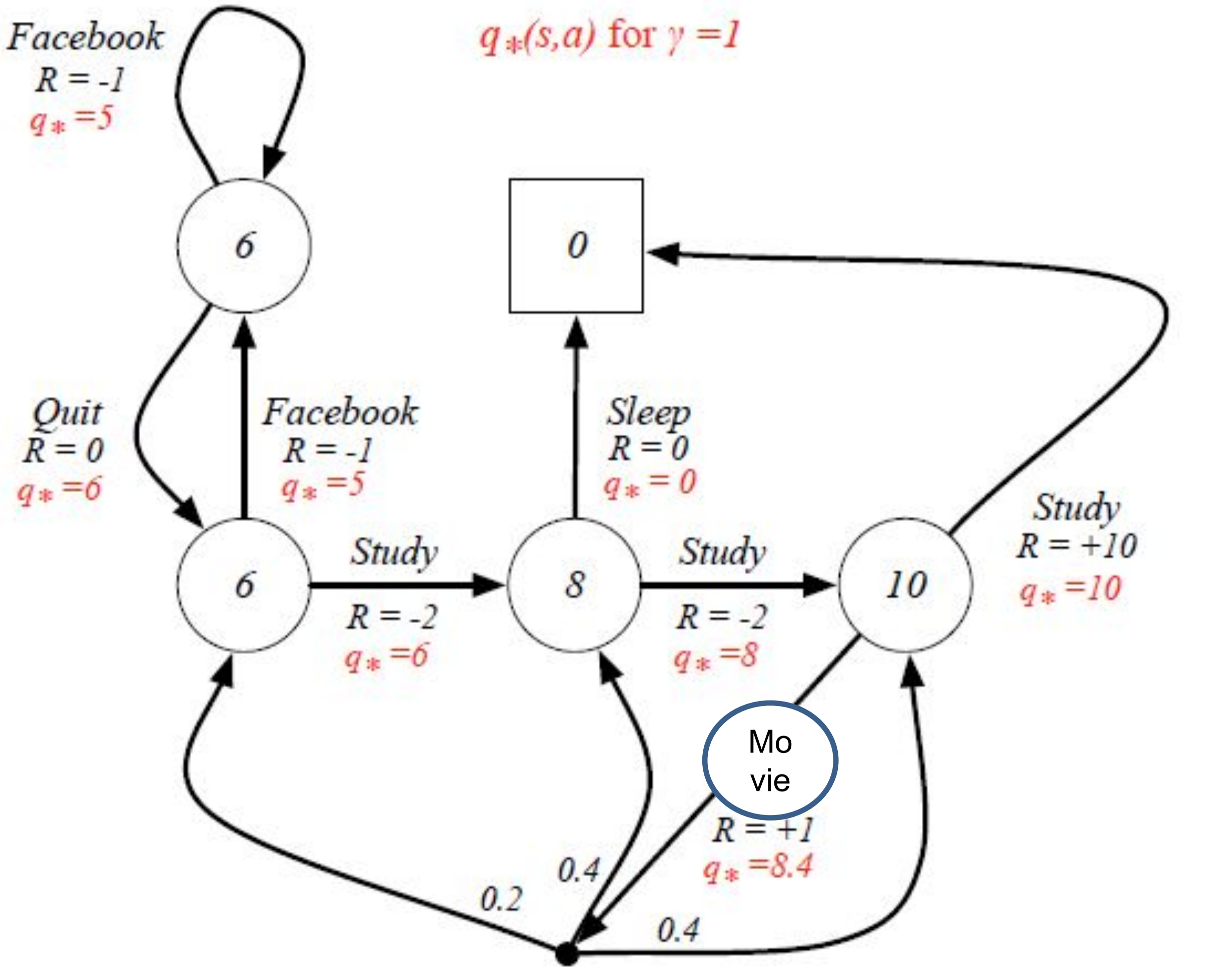
Example (MDP)

Optimal (state) Value functions for each state



Example (MDP)

Student Markov Chain: Optimal Action-Value



Dynamic programming

When the model is fully known, following Bellman equations, we can use Dynamic Programming (DP)to iteratively evaluate value functions and improve policy.

Policy Evaluation

$$V_{t+1}(s) = \mathbb{E}_\pi[r + \gamma V_t(s') | S_t = s] = \sum_a \pi(a|s) \sum_{s',r} P(s', r|s, a)(r + \gamma V_t(s'))$$

Policy Improvement

Policy Improvement generates a better policy $\pi' \geq \pi$

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

$$= \sum_{s',r} P(s', r|s, a)(r + \gamma V_\pi(s'))$$

Dynamic programming

Policy Iteration

$$\pi_0 \xrightarrow{\text{evaluation}} V_{\pi_0} \xrightarrow{\text{improve}} \pi_1 \xrightarrow{\text{evaluation}} V_{\pi_1} \xrightarrow{\text{improve}} \pi_2 \xrightarrow{\text{evaluation}} \dots \xrightarrow{\text{improve}} \pi_* \xrightarrow{\text{evaluation}} V_*$$

Say, we have a policy π and then generate an improved version π' by greedily taking actions, The value of this improved π' is guaranteed to be better because

$$\begin{aligned} Q_\pi(s, \pi'(s)) &= Q_\pi(s, \arg\max_{a \in \mathcal{A}} Q_\pi(s, a)) \\ &= \max_{a \in \mathcal{A}} Q_\pi(s, a) \geq Q_\pi(s, \pi(s)) = V_\pi(s) \end{aligned}$$

Q-function approximation

https://gibberblot.github.io/rl-notes/single-agent/function-approximation.html