Machine Learning

Reinforcement Learning

2017.12.
JY

Agenda

- Machine Learning
  - Definition
  - Machine Learning Algorithm

- Markov Chain

- Markov Reward Processes
  - Discount Factor
  - Markov Chain
  - Total Discounted Factor
  - The Bellman Equation
  - The Bellman Expectation Equation for value function

- Markov Decision Processes
  - Definition
  - Policy
  - The Bellman Expectation Equation
  - Optimal Value Function
  - Optimal State/Active-Value Function
  - Optimal Policy
  - The Bellman Optimality Equation

Agenda

- Dynamic Programming
  - Policy Evaluation
  - Policy Iteration
  - Policy Improvement
  - Generalised Policy Iteration
  - Deterministic Value Iteration
  - Value Iteration
  - Extensions to Dynamic Programming

- Reference

Machine Learning

Machine Learning

Definition of Machine Learning

- Machine Learning is a field of study that give computer ability to learn without being explicitly programmed

Arthur Samuel (1959)

Machine Learning

Reinforcement Learning is one of the three Machine Learning Algorithms :

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

Machines take trial and error → Rule of thumb

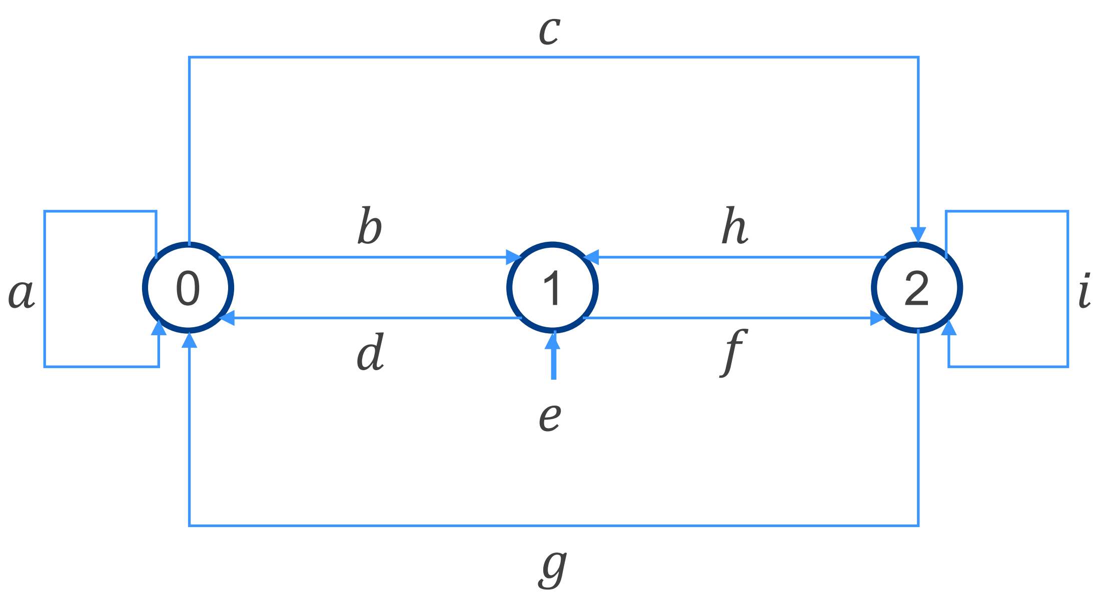
- Learn from an *environment*

Markov Chain

Markov Chain

Brief review of Markov Chain

- A state ONLY depends on the previous state
- State transition probability matrix
- The following is a state diagram
- The sum of each row is 1



State Diagram

$$\begin{array}{c c} & \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{array}{c} 0 \\ 1 \\ 2 \end{array} & \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \end{array}$$

Transition Matrix

MRPs

Markov Reward Processes

List of useful equations for Markov Reward Processes.

- Total discounted reward,

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- Value function,

$$v(s) = E[G_t | S_t = s]$$

※
$$\begin{aligned} v(s) &= E[G_t | S_t = s] \\ &= E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s] \\ &= E[R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \cdots) | S_t = s] \\ &= E[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= E[R_{t+1} + \gamma v(S_{t+1}) | S_t = s] \end{aligned}$$

Immediate reward : R_{t+1} Discounted value of successor state : $\gamma v(S_{t+1})$

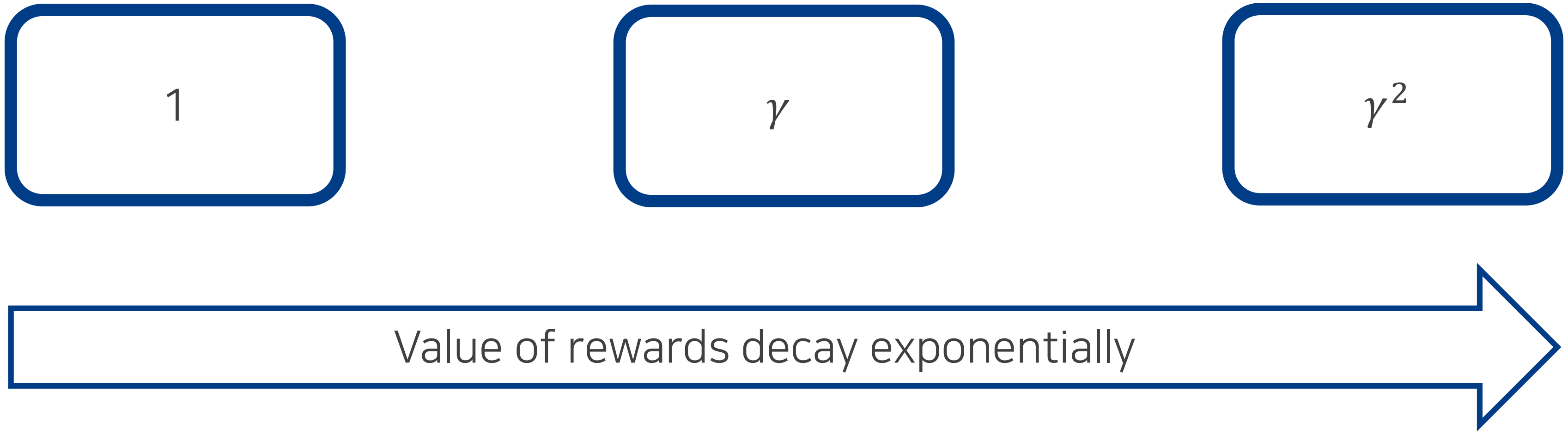
Source : UCL Course on RL

Markov Reward Processes

Discount factor

Def. γ is a discount factor $\gamma \in [0,1]$.

We want to maximize the sum of rewards.
We also prefer rewards now to reward later.

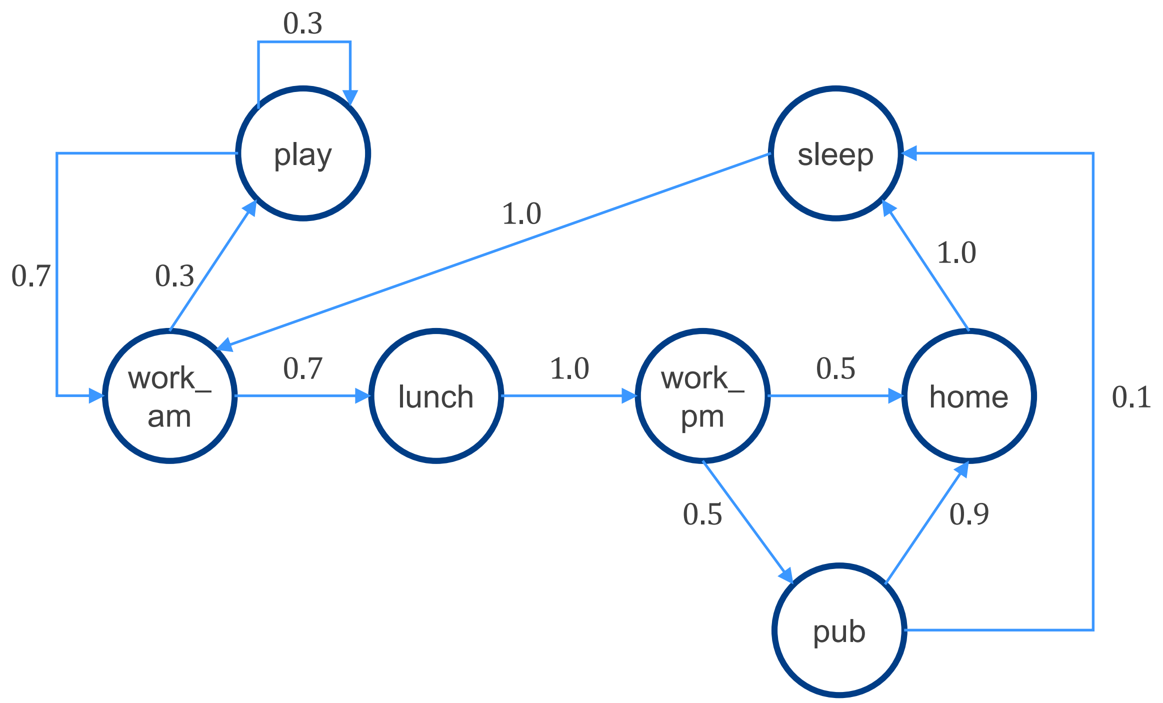


Source : UCL Course on RL

Markov Reward Processes

Markov Chain

- A state ONLY depends on the previous state
- State diagram
- State transition probability matrix
- The sum of each row is 1



Daily Life of an Employee

$$
\begin{array}{c}
 \\ \textit{work\_am} \\ \textit{lunch} \\ \textit{work\_pm} \\ \textit{home} \\ \textit{play} \\ \textit{pub} \\ \textit{sleep}
\end{array}
\begin{array}{c}
\begin{array}{ccccccc} \textit{work\_am} & \textit{lunch} & \textit{work\_pm} & \textit{home} & \textit{play} & \textit{pub} & \textit{sleep} \end{array} \\
\begin{pmatrix}
0 & 0.7 & 0 & 0 & 0.3 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1.0 \\
0.7 & 0 & 0 & 0 & 0.3 & 0 & 0 \\
0 & 0 & 0 & 0.9 & 0 & 0 & 0.1 \\
1.0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\end{array}
$$

Transition Matrix

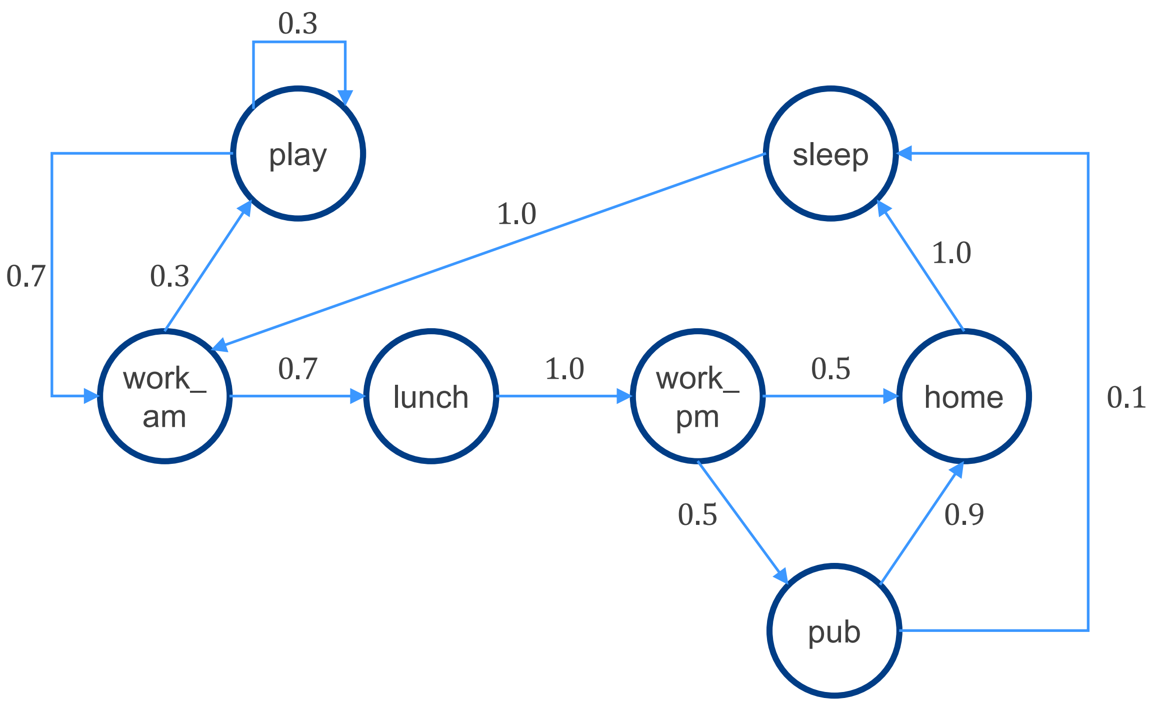
Markov Reward Processes

Total discounted reward

Def. The *return* G_t is the total discounted reward from time-step t,

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

eg. Let $\gamma = 0$.



Daily Life of an Employee

$$
\begin{array}{c}
\\ work\_am \\ lunch \\ work\_pm \\ home \\ play \\ pub \\ sleep
\end{array}
\begin{array}{c}
\begin{array}{ccccccc} work\_am & lunch & work\_pm & home & play & pub & sleep \end{array} \\
\begin{pmatrix}
0 & 0.7 & 0 & 0 & 0.3 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1.0 \\
0.7 & 0 & 0 & 0 & 0.3 & 0 & 0 \\
0 & 0 & 0 & 0.9 & 0 & 0 & 0.1 \\
1.0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\end{array}
$$

Transition Matrix

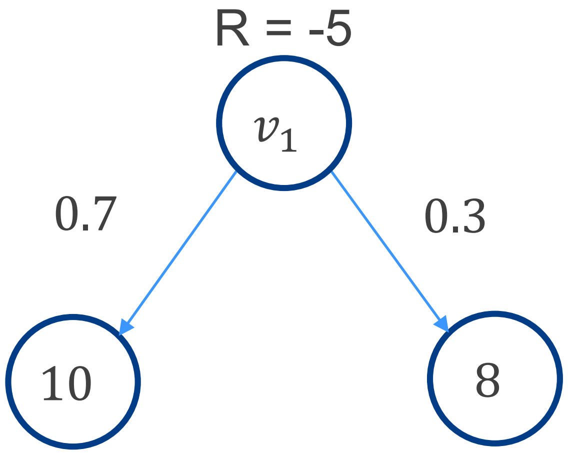
Markov Reward Processes

Total discounted reward

Def. The *return* G_t is the total discounted reward from time-step t,

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

eg. Let $\gamma = 1$.



Daily Life of an Employee

How do we find value function v_1?

$$v_1 = -5 + 0.7 \times 10 + 0.3 \times 8 = 4.4$$

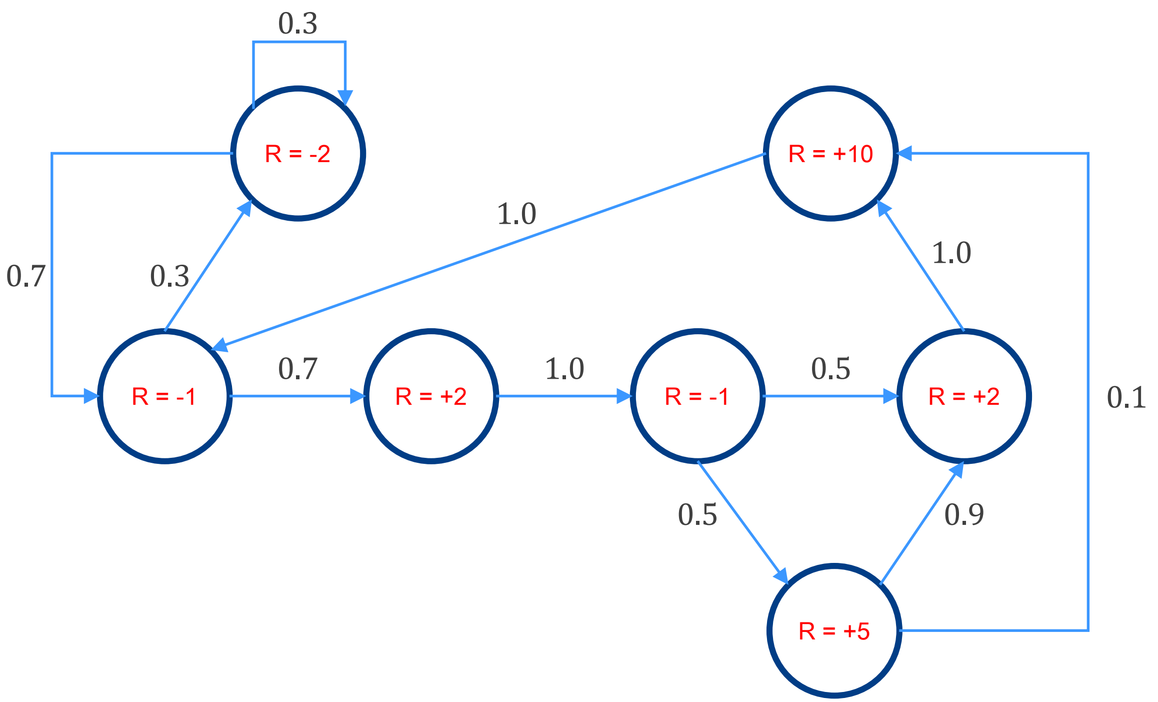
Markov Reward Processes

Total discounted reward

Def. The *return* G_t is the total discounted reward from time-step t,

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

eg. Let $\gamma = 0$.



Daily Life of an Employee

Since $\gamma = 0$, we only care about the immediate reward, which means we only get the reward from the next state.

Other states are irrelevant!

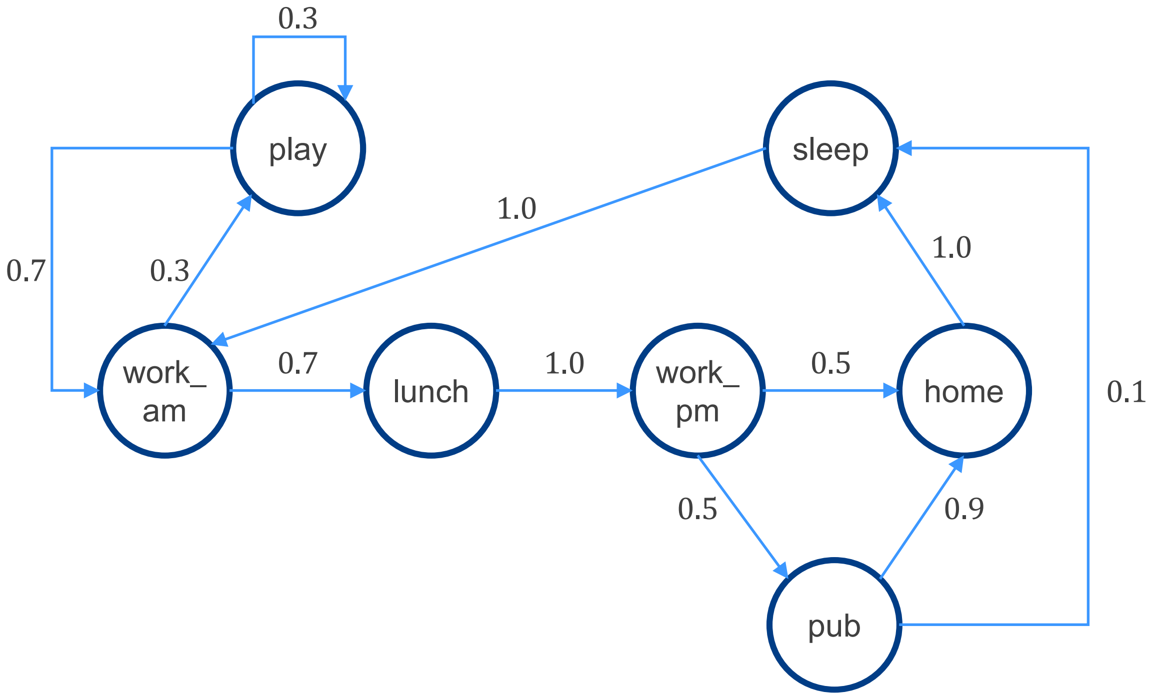
Markov Reward Processes

Total discounted reward

Def. The *return* G_t is the total discounted reward from time-step t,

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

eg. Let $\gamma = \frac{1}{2}$.



Daily Life of an Employee

$$
\begin{array}{r c}
 & \begin{array}{ccccccc} \textit{work\_am} & \textit{lunch} & \textit{work\_pm} & \textit{home} & \textit{play} & \textit{pub} & \textit{sleep} \end{array} \\
\begin{array}{r} \textit{work\_am} \\ \textit{lunch} \\ \textit{work\_pm} \\ \textit{home} \\ \textit{play} \\ \textit{pub} \\ \textit{sleep} \end{array} &
\begin{pmatrix}
0 & 0.7 & 0 & 0 & 0.3 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1.0 \\
0.7 & 0 & 0 & 0 & 0.3 & 0 & 0 \\
0 & 0 & 0 & 0.9 & 0 & 0 & 0.1 \\
1.0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\end{array}
$$

Transition Matrix

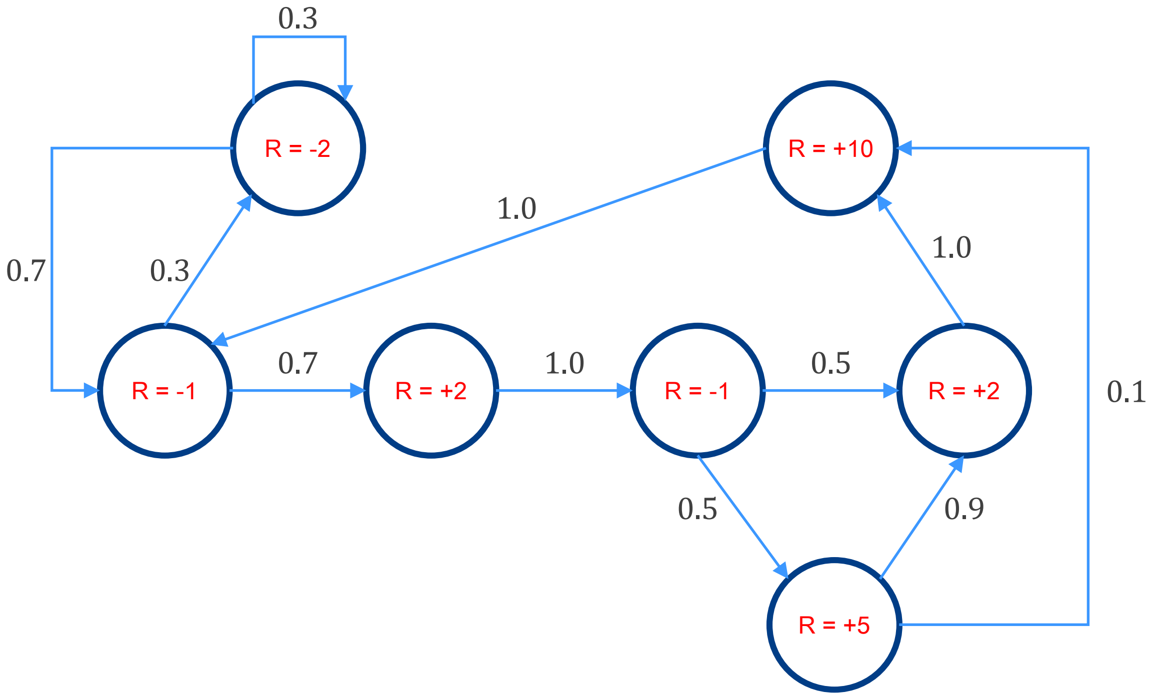
Markov Reward Processes

Total discounted reward

Def. The *return* G_t is the total discounted reward from time-step t,

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

eg. Let $\gamma = \frac{1}{2}$.



Daily Life of an Employee

work_am → lunch → work_pm → home → sleep

$= -1 + 2 \times \frac{1}{2} - 1 \times \frac{1}{4} + 2 \times \frac{1}{8} + 10 \times \frac{1}{16} = 0.625$

work_am → play → lunch → work_pm → home

$= -1 - 2 \times \frac{1}{2} + 2 \times \frac{1}{4} - 1 \times \frac{1}{8} + 2 \times \frac{1}{16} = -1.5$

work_am → work_pm → home → sleep

$= -1 - 1 \times \frac{1}{2} + 2 \times \frac{1}{4} + 10 \times \frac{1}{8} = 0.25$

work_am → lunch → pub → home → sleep

$= -1 + 2 \times \frac{1}{2} + 5 \times \frac{1}{4} + 2 \times \frac{1}{8} + 10 \times \frac{1}{16} = 2.125$

…

We need to average these return G_t…

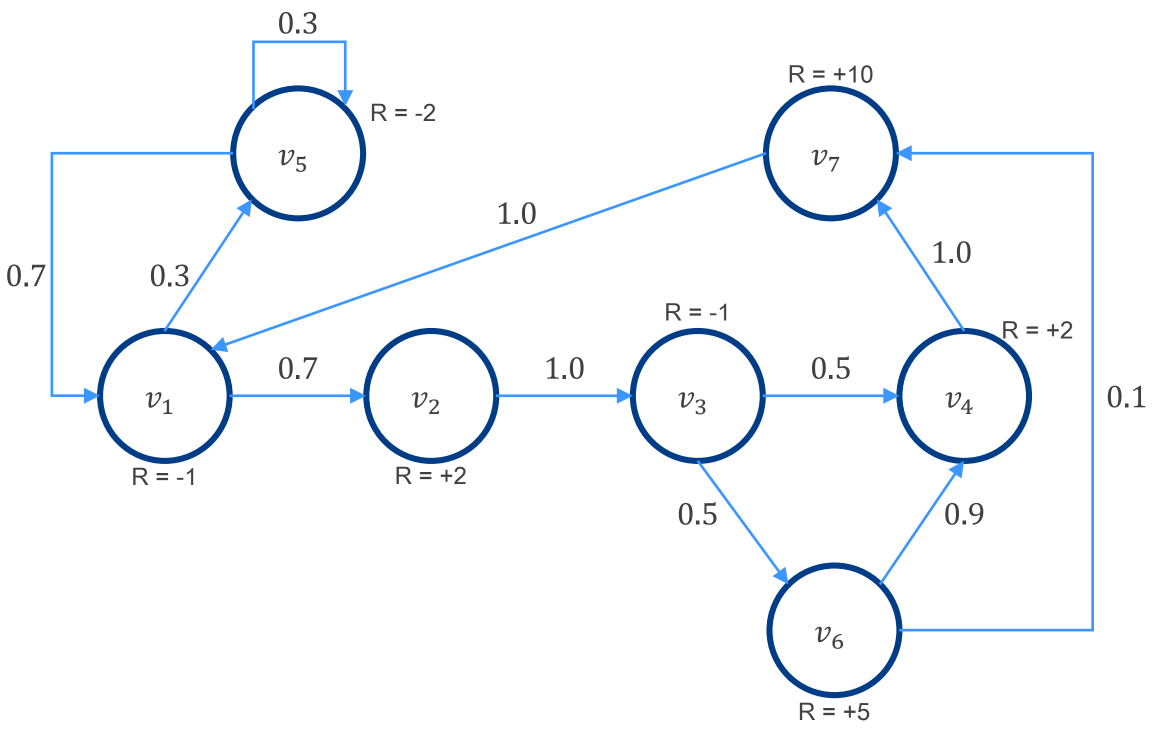
Markov Reward Processes

Value Function

Def. The *return* G_t is the total discounted reward from time-step t,

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

eg. Let $\gamma = \frac{1}{2}$.



Daily Life of an Employee

$$v(s) = E[G_t | S_t = s]$$

How do we find value function?

$v_1 = ???$
$v_2 = ???$
$v_3 = ???$
$v_4 = ???$
$v_5 = ???$
$v_6 = ???$
$v_7 = ???$

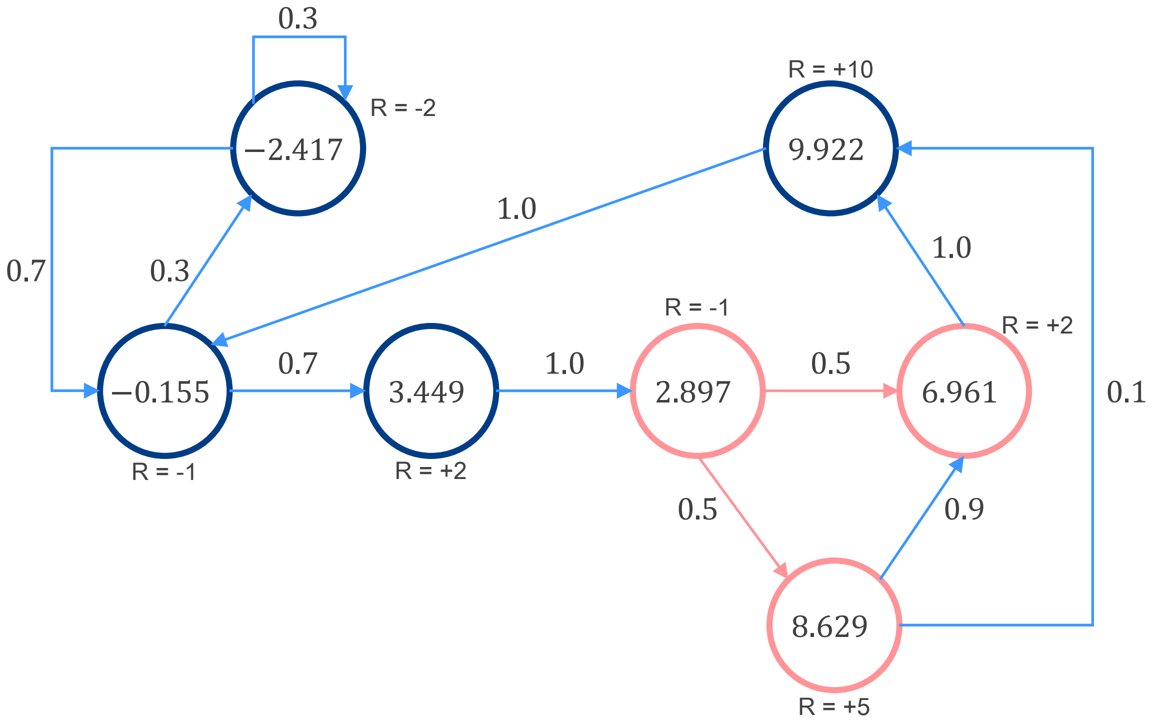
Markov Reward Processes

Value Function

Def. The *return* G_t is the total discounted reward from time-step t,

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

eg. Let $\gamma = \frac{1}{2}$.



Daily Life of an Employee

How do we find value function?

$$v_3 = -1 + 0.5(0.5 \times 6.961 + 0.5 \times 8.629)$$
$$= 2.8975$$

Markov Reward Processes

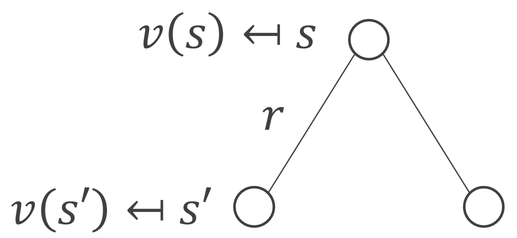
The Bellman Expectation Equation for value function

The two parts of value function :

- Immediate reward R_{t+1}
- Discounted value of successor state $\gamma v(S_{t+1})$

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

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



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

Source : UCL Course on RL

Markov Reward Processes

The Bellman Equation in matrix form

The Bellman Equation can be expressed as the following using matrices,

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

where v is a column vector with one entry per state,

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} R_1 \\ \vdots \\ R_n \end{bmatrix} + \gamma \begin{bmatrix} P_{11} & \cdots & P_{1n} \\ \vdots & \ddots & \vdots \\ P_{n1} & \cdots & P_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

Thank you Bellman Equation!! You are a linear equation!!
We can solve for v now.

Source : UCL Course on RL

Markov Reward Processes

The Bellman Equation in matrix form

The Bellman Equation can be expressed as the following using matrices,

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

where v is a column vector with one entry per state,

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

Source : UCL Course on RL

MDPs

Markov Decision Processes

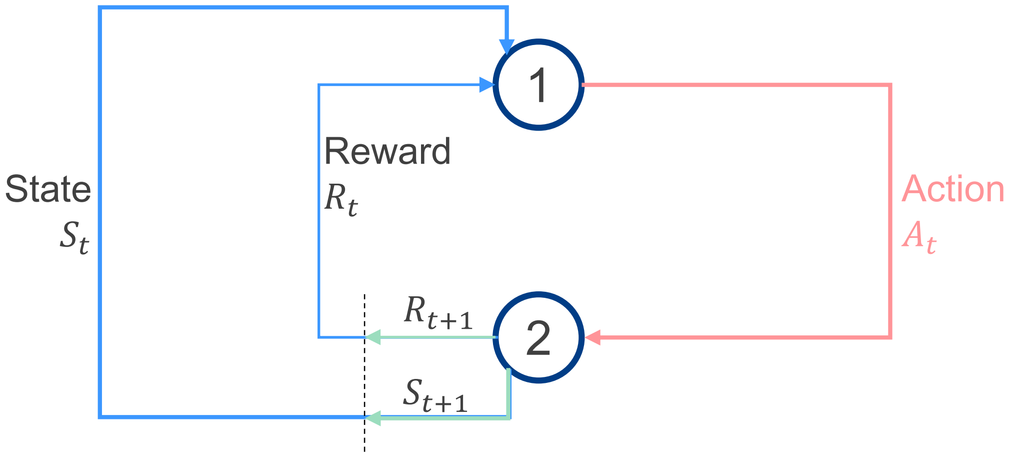
Def. A Markov Decision Processes is a tuple $\langle S, A, P, R, \gamma \rangle$.

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

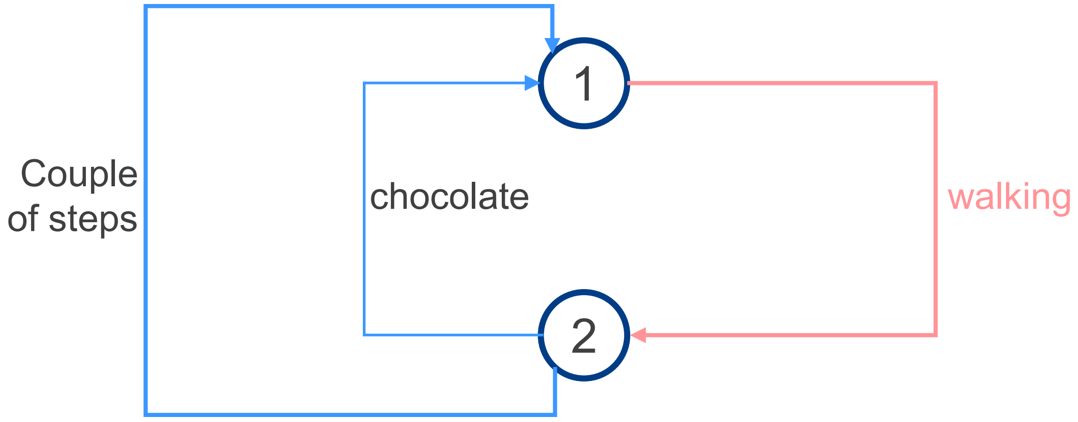
→ Markov property

gets *reward* (ALL the rewards **over time**) by taking action from state s

value decrease over time



1 = agent, 2 = environment



Source : Simple Beginner's guide to Reinforcement Learning & its implementation, UCL Course on RL

Markov Decision Processes

List of useful equations for Markov Decision Processes.

- Total discounted reward,

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

- State-value function,

$$v_\pi(s) = E_\pi[G_t | S_t = s]$$

- Policy,

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

- Action-value function,

$$q_\pi(s,a) = E_\pi[G_t | S_t = s, A_t = a]$$

Source : UCL Course on RL

Markov Decision Processes

Policy

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

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

Determining an action in a certain state on a specific time is called “Policy”.

$$A_t \sim \pi(\cdot\,|S_t), \forall\, t > 0$$

★ $$P^{\pi}_{ss'} = \sum_{a \in A} \pi(a|s) P^{a}_{ss'}$$

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

Source : UCL Course on RL

Markov Decision Processes

Policy

Value function

- State-value function for policy

Def. The *state-value function* $v_\pi(s)$ of an Markov Decision Processes is the expected return starting from state s, and then following policy π,

$$v_\pi(s) = E_\pi[G_t | S_t = s]$$

The state-value function can be varied for each policy.
Since we need to find the policy which maximizes its value function, state-value function plays an important role in reinforcement learning.

The state-value function can again be decomposed into immediate reward plus discounted value of successor state,

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

Source : UCL Course on RL

Markov Decision Processes

Policy

Value function

- Action-value function for policy

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

$$q_\pi(s, a) = E_\pi[G_t | S_t = s, A_t = a]$$

This is the expected value of return when an action is taken in a certain state.

The action-value function can similarly be decomposed,

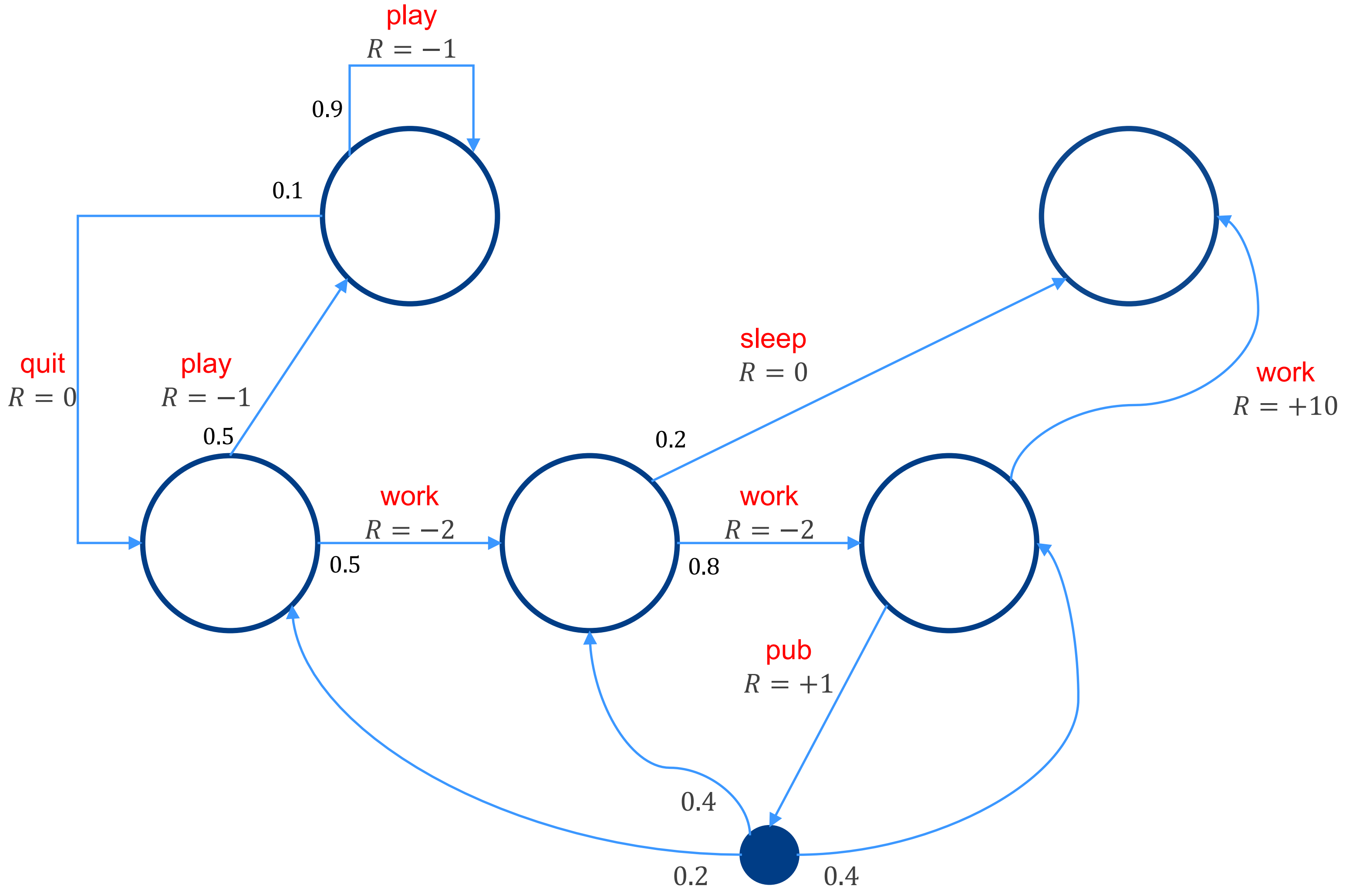
$$q_\pi(s, a) = E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

Source : UCL Course on RL

Markov Decision Processes

Policy

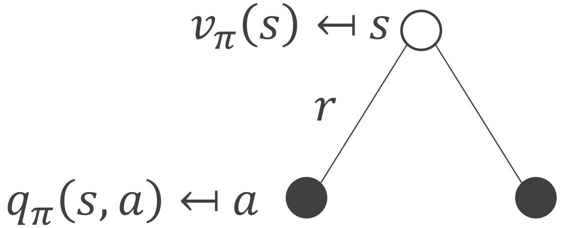
- State Diagram

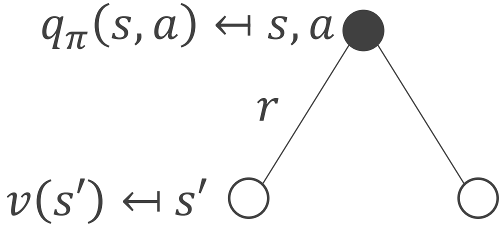


Daily Life of an Employee

Markov Decision Processes

The Bellman Expectation Equation for V^{π} and Q^{π}





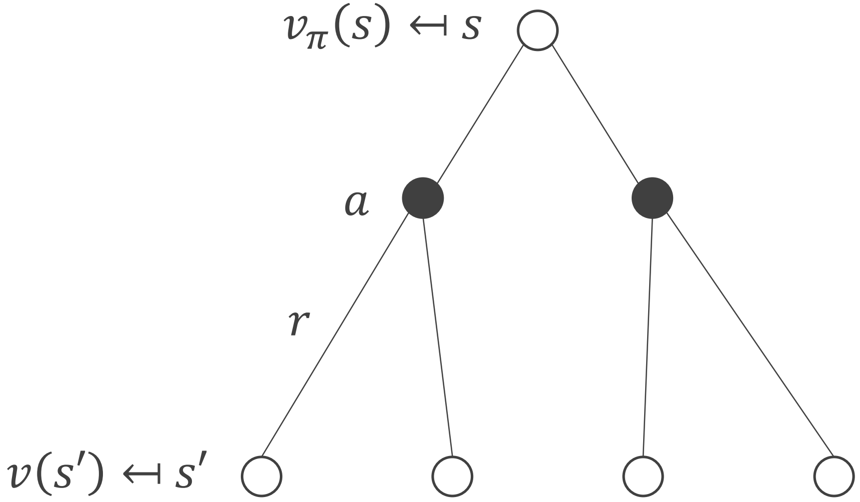
$$v_\pi(s) = \sum_{a \in A} \pi(a|s) q_\pi(s,a)$$

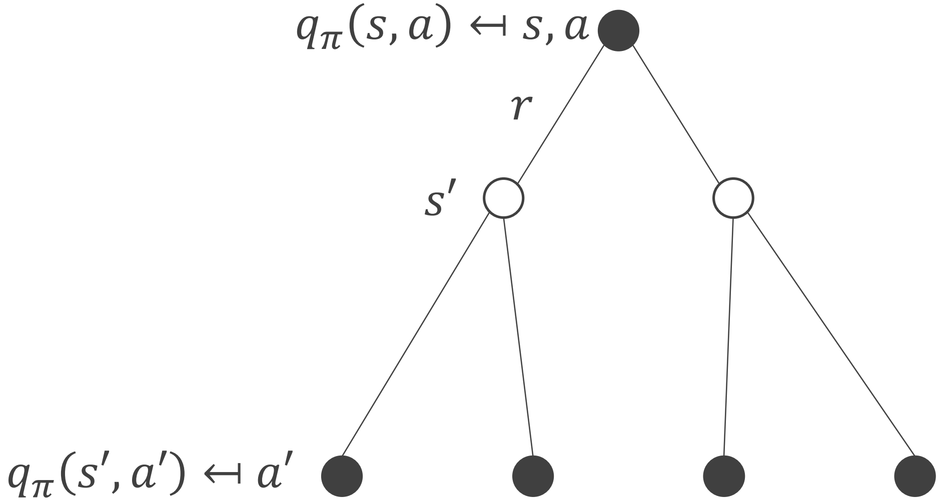
$$q_\pi(s,a) = R_S^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s')$$

Source : UCL Course on RL

Markov Decision Processes

The Bellman Expectation Equation for V^π and Q^π





$$v_\pi(s) = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s') \right)$$

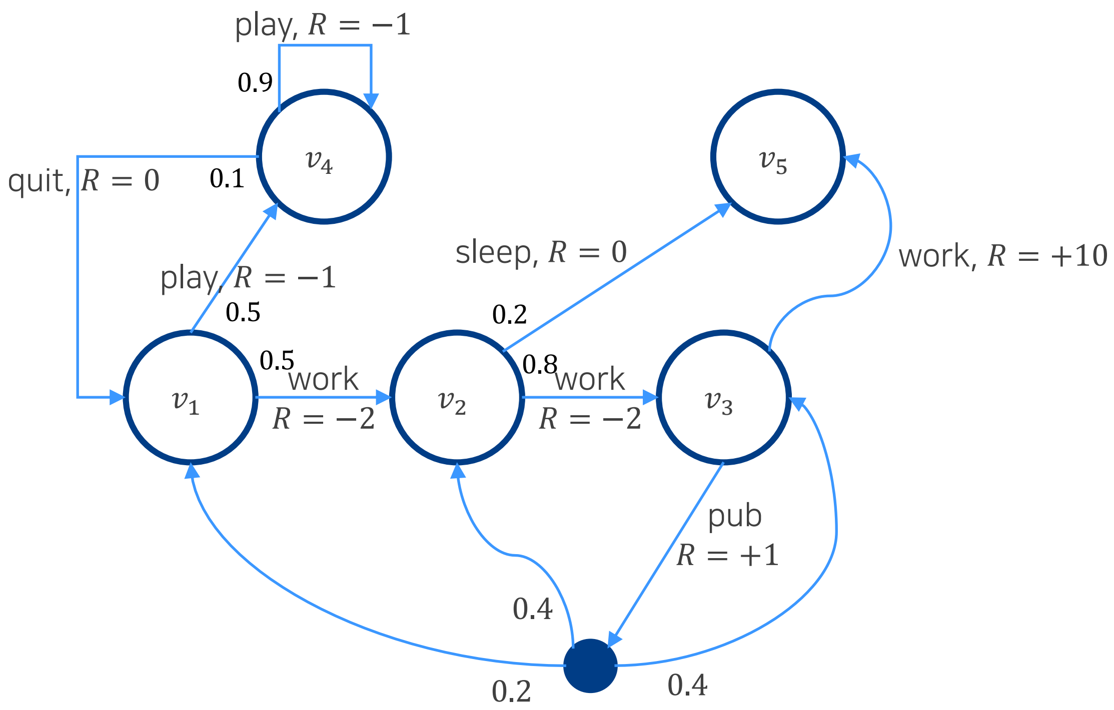
$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a'|s') q_\pi(s',a')$$

Source : UCL Course on RL

Markov Decision Processes

The Bellman Expectation Equation

eg. Let $\pi(a|s) = 0.5, \gamma = 1.$



Daily Life of an Employee

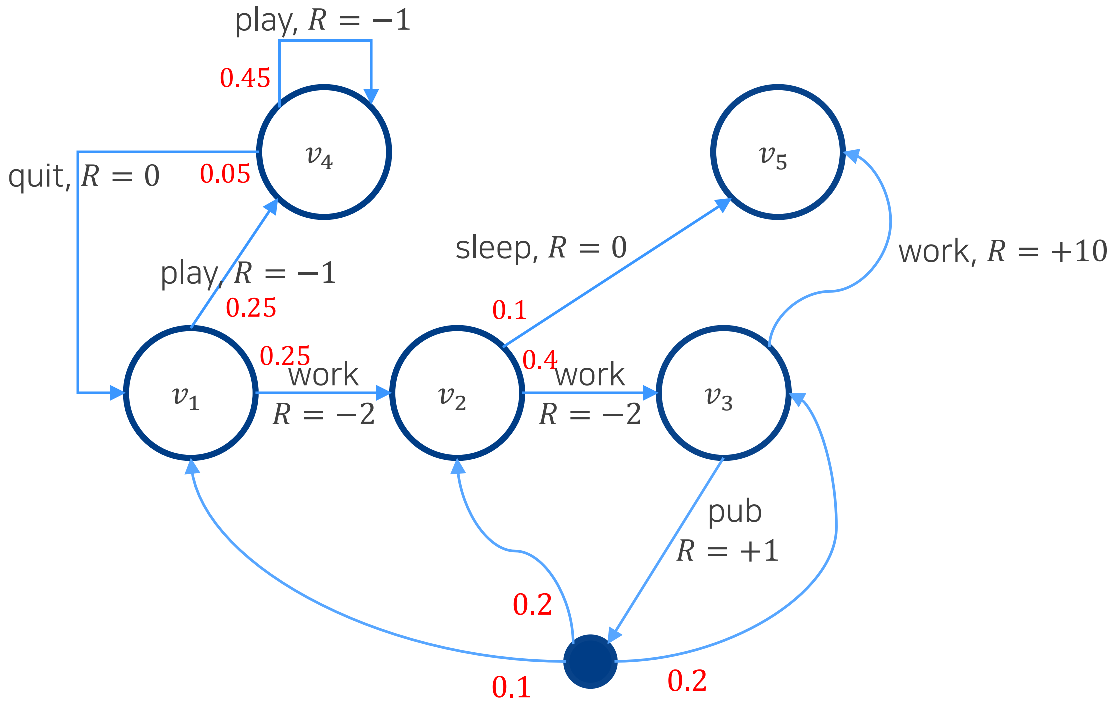
How do we find value function?

$$v_1 = ???$$
$$v_2 = ???$$
$$v_3 = ???$$
$$v_4 = ???$$
$$v_5 = ???$$

Markov Decision Processes

The Bellman Expectation Equation

eg. Let $\pi(a|s) = 0.5, \gamma = 1.$



Daily Life of an Employee

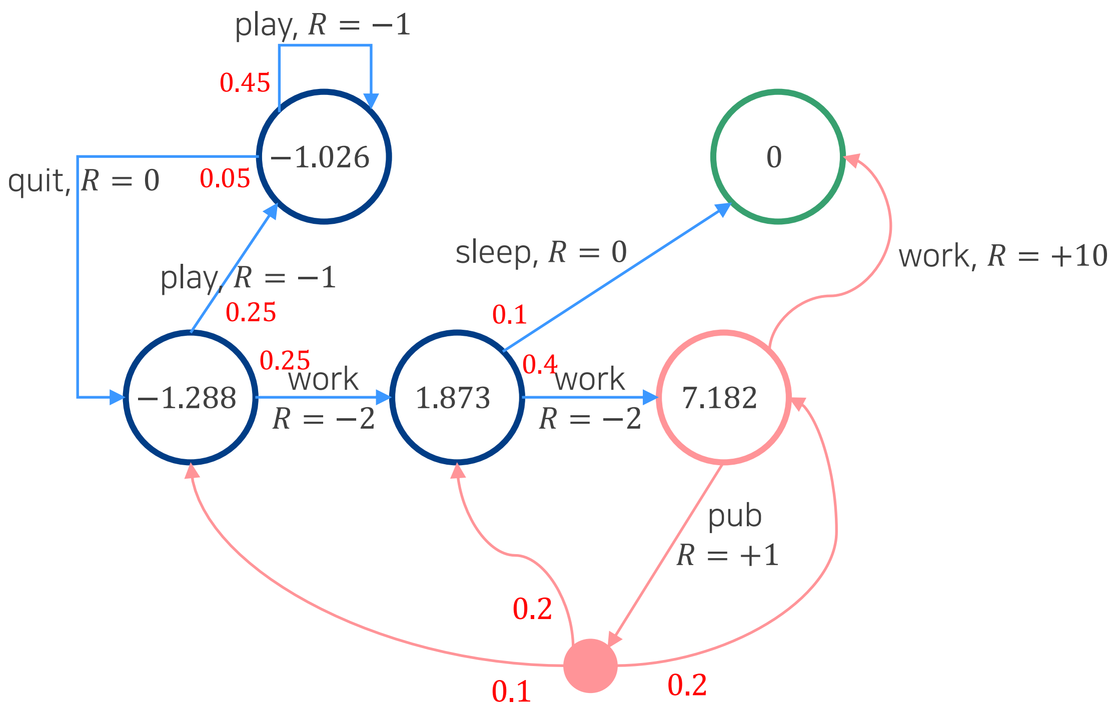
$$\begin{array}{c c} & \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} & \begin{pmatrix} 0.25 & 0 & 0 & 0.25 & 0.3 \\ 0 & 0 & 0.4 & 0 & 0.1 \\ 0.1 & 0.2 & 0.2 & 0 & 0 \\ 0.05 & 0 & 0 & 0.45 & 0 \\ 0 & 0 & 0 & 0 & 0.5 \end{pmatrix} \end{array}$$

Transition Matrix

Markov Decision Processes

The Bellman Expectation Equation

eg. Let $\pi(a|s) = 0.5, \gamma = 1.$



Daily Life of an Employee

How do we find value function?

$$
\begin{aligned}
v_3 &= 0.5 \times (1 + 0.2 \times (-1.288) + 0.4 \times (1.873) + \\
&\quad 0.4 \times (7.182) + 0.5 \times 10 \\
&= 7.1822
\end{aligned}
$$

Markov Decision Processes

The Bellman Expectation Equation in matrix form

The Bellman Equation can be expressed as the following using matrices,

$$v_\pi = R^\pi + \gamma P^\pi v_\pi$$

$$v_\pi = (I - \gamma P^\pi)^{-1} R^\pi$$

P^π = Average state transition dynamics
v_π = Averaged reward function

Since the Bellman Equation is a linear equation, we can solve for v now.

Source : UCL Course on RL

Markov Decision Processes

Optimal Value Function

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

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

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

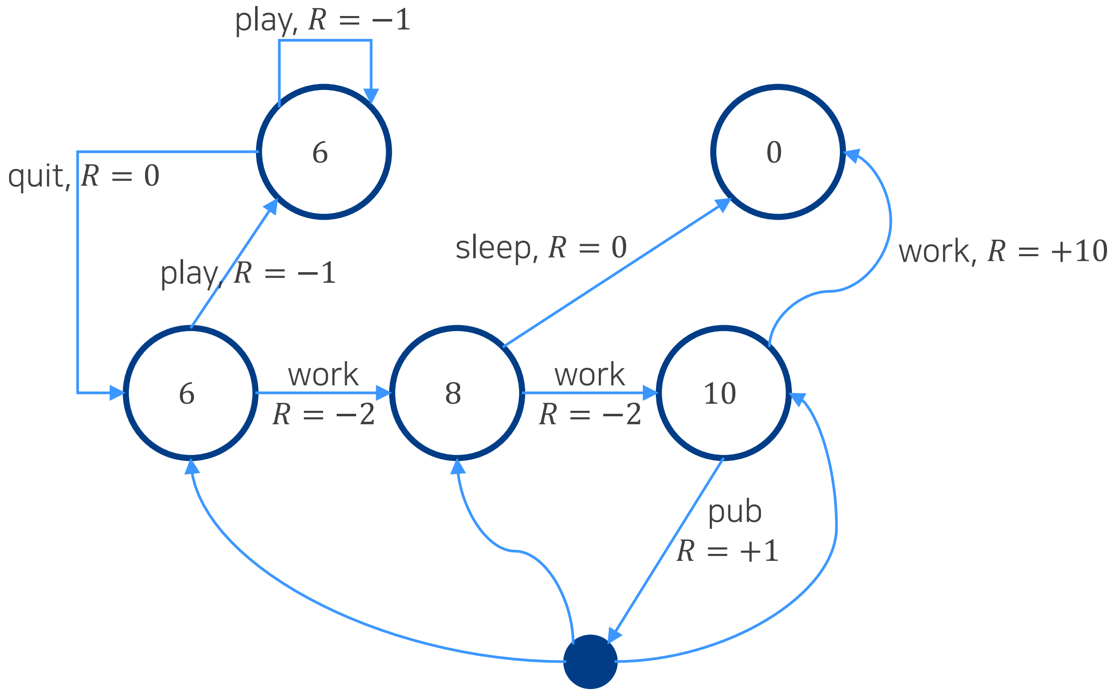
$$q_*(s,a) = \max_{\pi} q_{\pi}(s,a)$$

- The optimal value function specifies the best possible performance in the Markov Decision Processes
- Markov Decision Processes is “solved” when we know the optimal value function

Markov Decision Processes

Optimal State-Value Function

eg. Find $v_*(s)$ for $\gamma = 1$.



Daily Life of an Employee

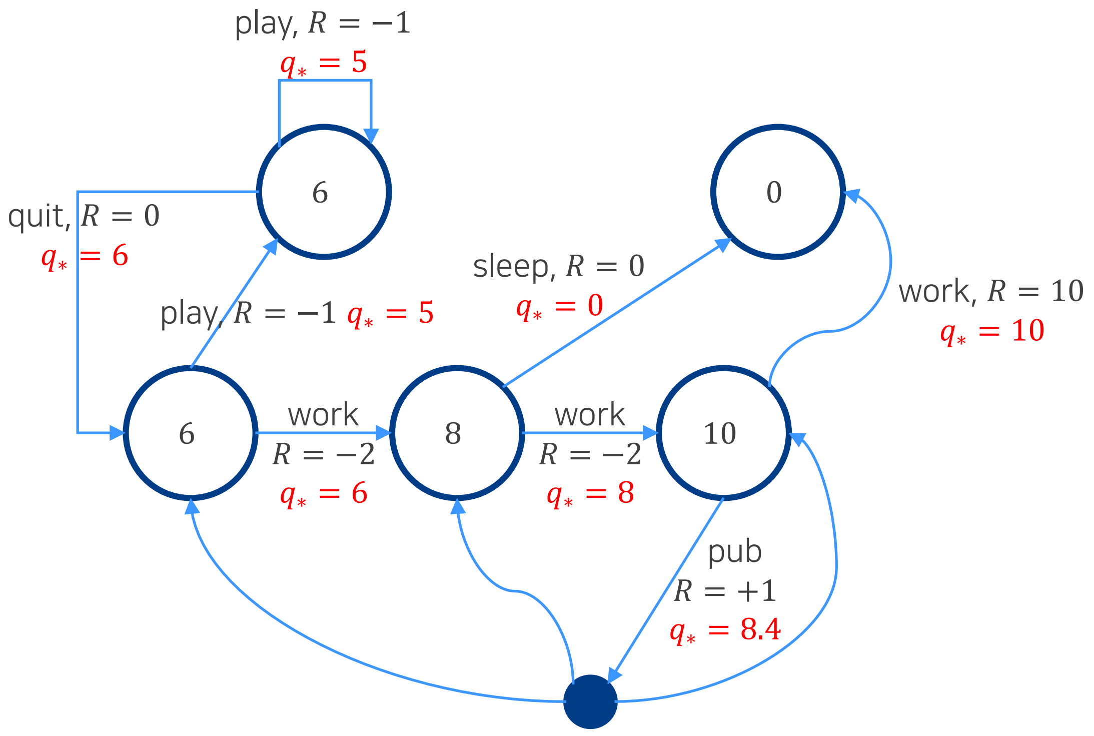
Optimal state-value function?

$v_1 = 8 - 2$ or $v_1 = 6 - 1$
$v_2 = 10 - 2$ or $v_2 = 0 + 0$
$v_3 = 1$ or $v_3 = 0 + 10$
$v_4 = 6 + 0$ or $v_4 = 6 - 1$
$v_5 = 0$

Markov Decision Processes

Optimal Action-Value Function

eg. Find $q_*(s,a)$ for $\gamma = 1$?



Daily Life of an Employee

Optimal action-value function?

$$q_* = 0 + 10 = 10$$
$$q_* = 10 - 2 = 8$$
$$q_* = 0 + 0 = 0$$
$$q_* = 8 - 2 = 6$$
$$q_* = 6 - 1 = 5$$
$$q_* = 6 + 0 = 6$$
$$q_* = 6 - 1 = 5$$

※ Arc for pub = DIY

Markov Decision Processes

Optimal Policy

Define a partial ordering over policies,

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

Theorem. For any Markov Decision Process,

- 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 value function, $v_{\pi_*}(s) = v_*(s)$
- All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s,a) = q_*(s,a)$

Source : UCL Course on RL

Markov Decision Processes

Finding an Optimal Policy

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

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

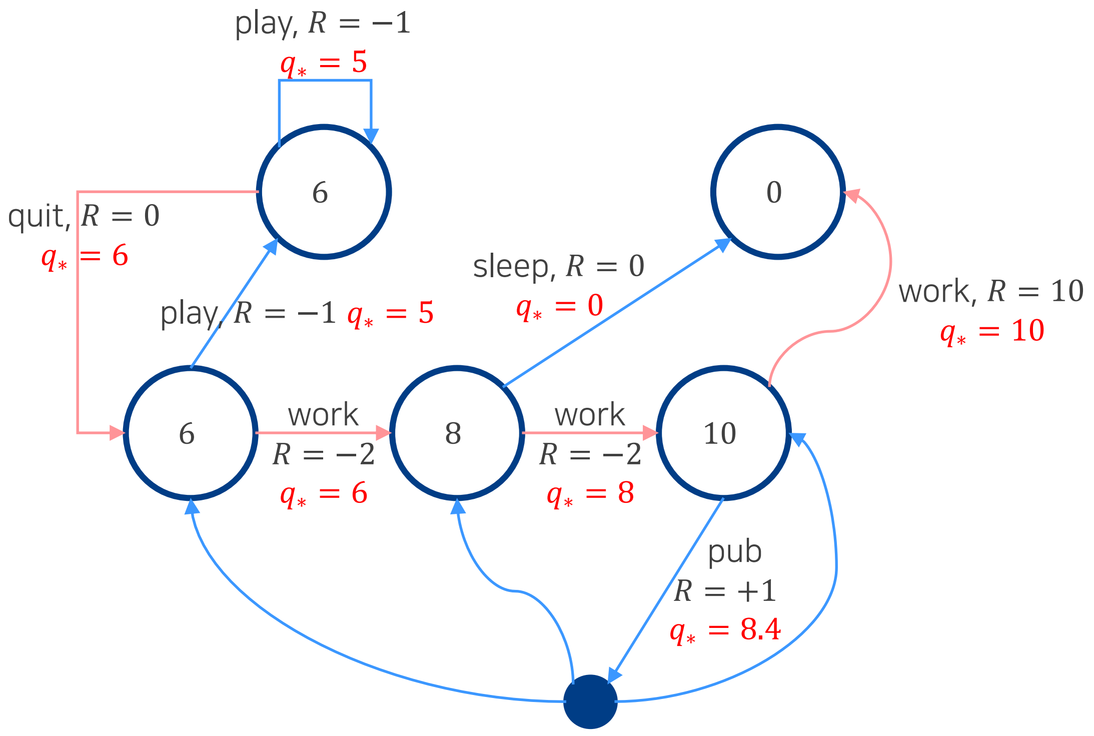
- There is always a deterministic optimal policy for any Markov Decision Processes
- If we know $q_*(s,a)$, we immediately have the optimal policy

Source : UCL Course on RL

Markov Decision Processes

Optimal Policy

eg. Find $\pi_*(s, a)$ for $\gamma = 1$?



Daily Life of an Employee

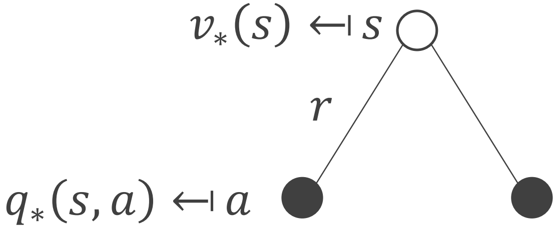
Optimal action-value function?

$$q_* = 0 + 10 = 10$$
$$q_* = 10 - 2 = 8$$
$$q_* = 0 + 0 = 0$$
$$q_* = 8 - 2 = 6$$
$$q_* = 6 - 1 = 5$$
$$q_* = 6 + 0 = 6$$
$$q_* = 6 - 1 = 5$$

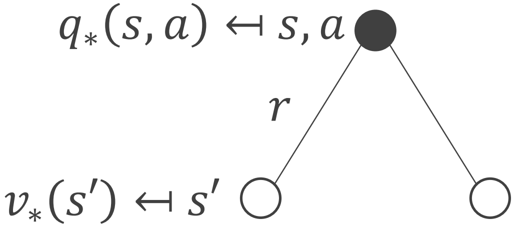
※ Arc for pub = DIY

Markov Decision Processes

The Bellman Optimality Equation for V^* and Q^*



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

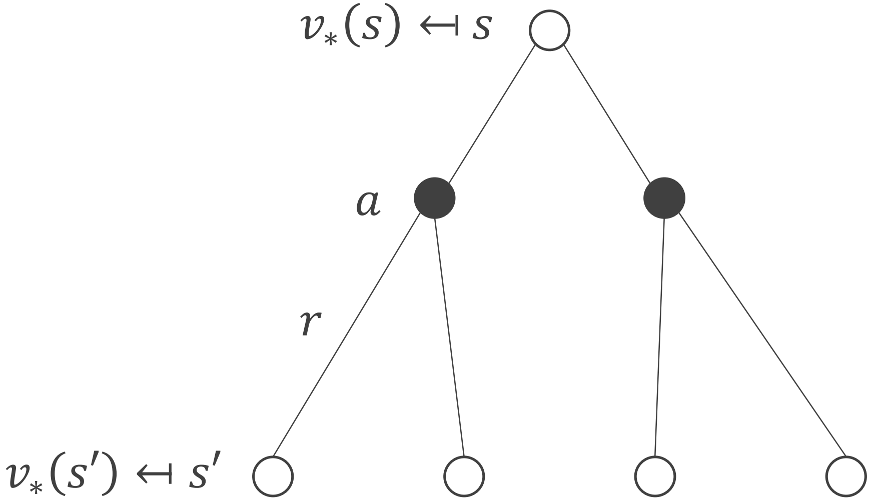


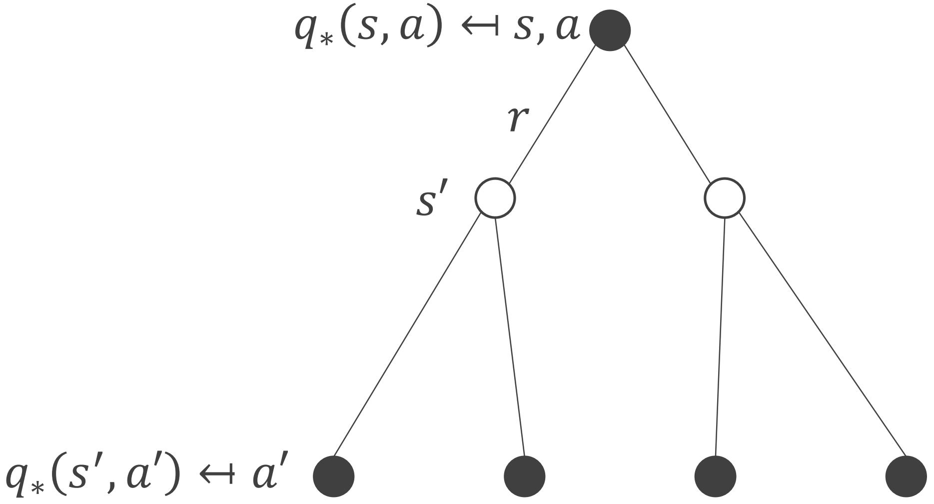
$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$$

Source : UCL Course on RL

Markov Decision Processes

The Bellman Optimality Equation for V^* and Q^*





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

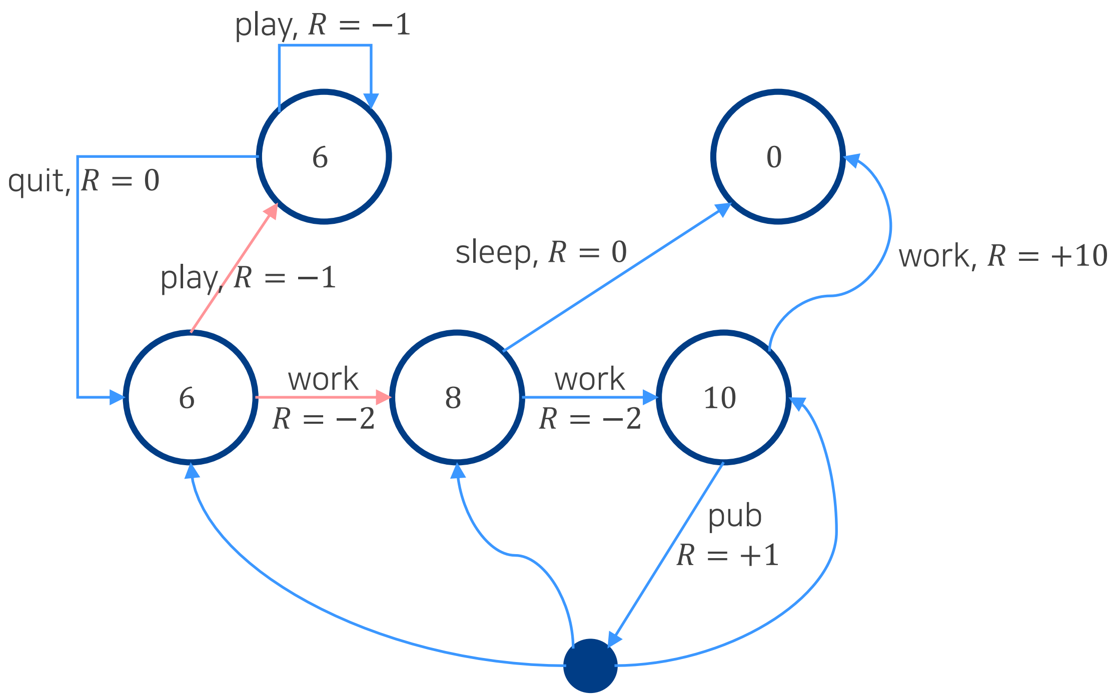
$$q_*(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q_*(s',a')$$

Source : UCL Course on RL

Markov Decision Processes

The Bellman Optimality Equation

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



Daily Life of an Employee

Optimal state-value function?

$$6 = \max_{\pi}\{8 - 2, 6 - 1\}$$

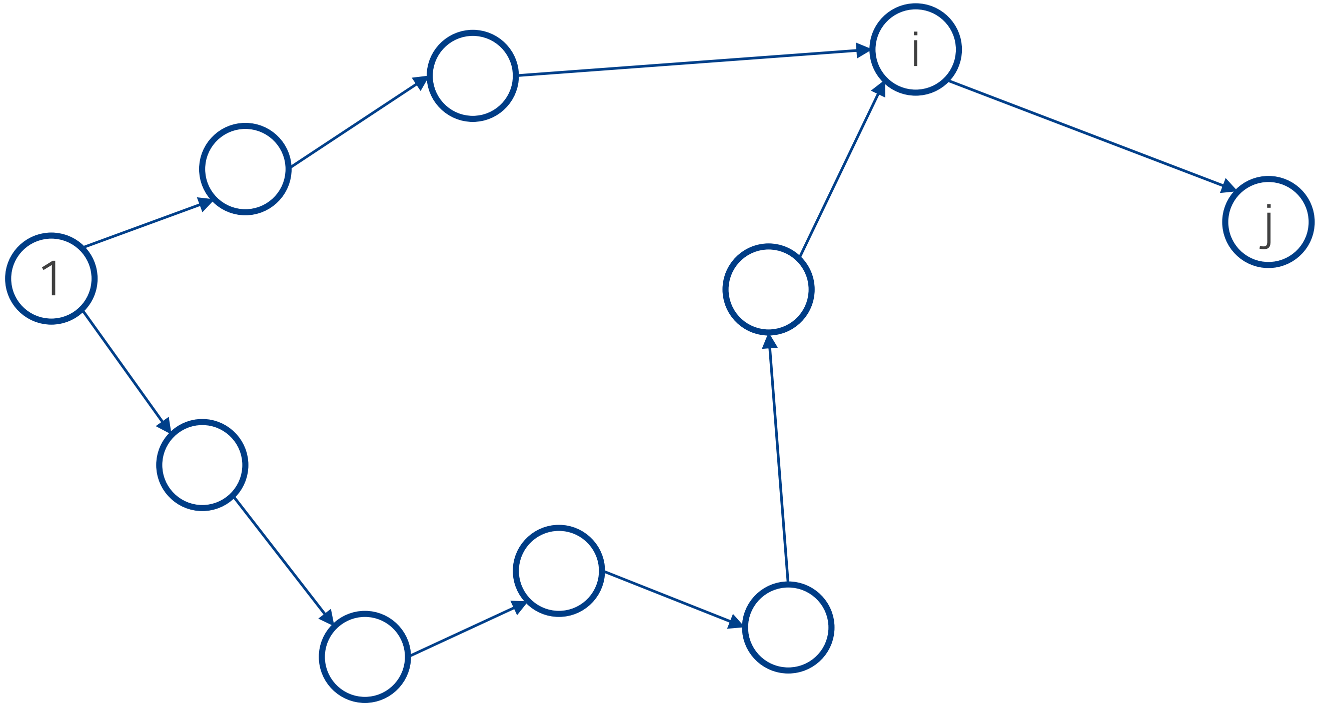
Dynamic Programming

Dynamic Programming

Dynamic Programming divides problem into subproblems, which are themselves usually divided into further subproblems.

A better name for Dynamic Programming might be *Recursive Optimization*.

eg. Shortest dipath problems

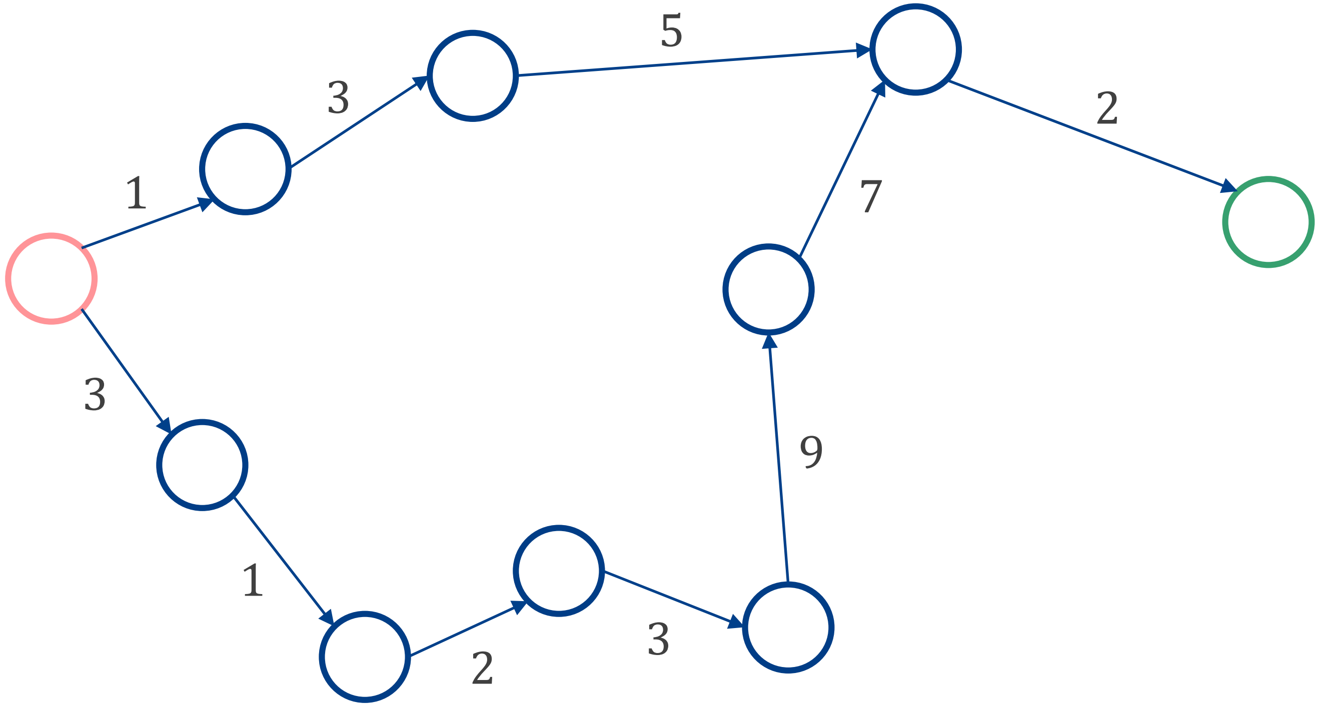


Dynamic Programming

Dynamic Programming divides problem into subproblems, which are themselves usually divided into further subproblems.

A better name for Dynamic Programming might be *Recursive Optimization*.

eg. Shortest dipath problems

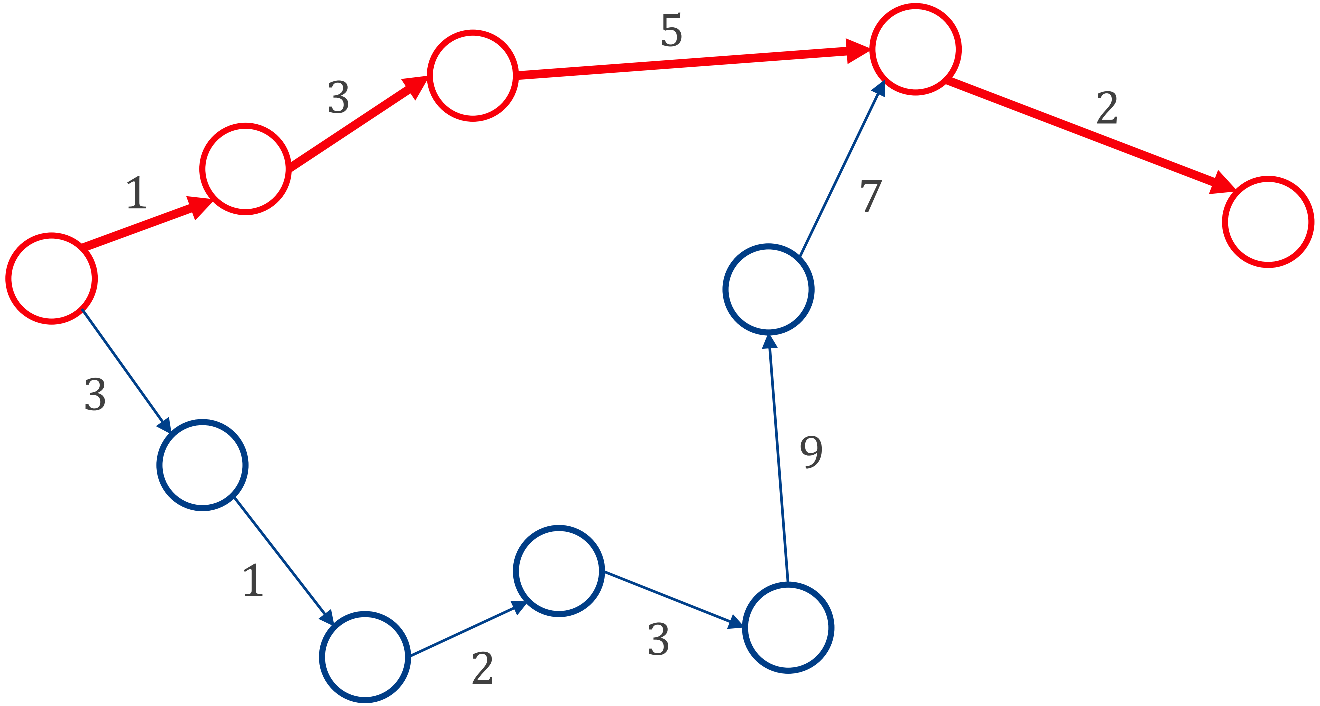


Dynamic Programming

Dynamic Programming divides problem into subproblems, which are themselves usually divided into further subproblems.

A better name for Dynamic Programming might be *Recursive Optimization*.

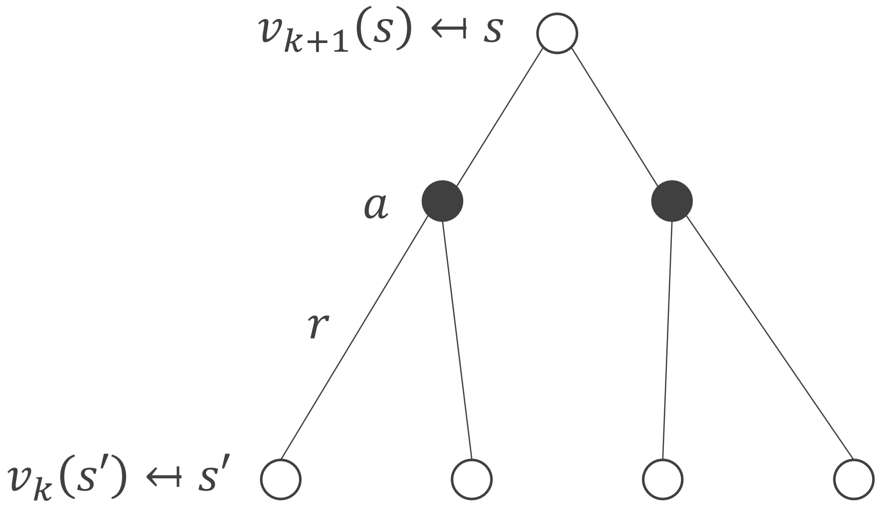
eg. Shortest dipath problems



Dynamic Programming

Policy Evaluation

- Iterative Policy Evaluation



$$v_{k+1}(s) = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \, v_k(s') \right)$$

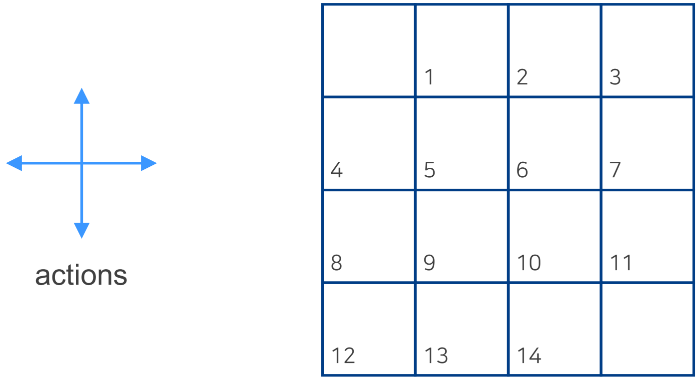
$$\vec{v}^{k+1} = \vec{R}^{\vec{\pi}} + \gamma \vec{P}^{\vec{\pi}} \vec{v}^{k}$$

Source : UCL Course on RL

Dynamic Programming

Policy Evaluation

eg. Evaluating a random policy in the Small Gridworld



$\gamma = -1$
on all transitions

- Undiscounted episodic Markov Decision Processes ($\gamma = 1$)
- Nonterminal states 1, …,14
- One terminal state (two shaded squares)
- Actions leading out of the grid leave state unchanged
- Reward is -1 until the terminal state is reached
- Agent follows uniform random policy

$$\pi(n|\cdot) = \pi(e|\cdot) = \pi(s|\cdot) = \pi(w|\cdot) = 0.25$$

Source : UCL Course on RL

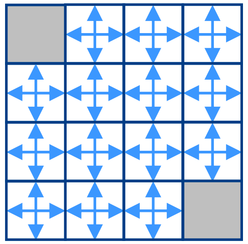
Dynamic Programming

Policy Evaluation

eg. Evaluating a random policy in the Small Gridworld

$k = 0$

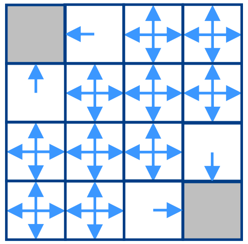
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



Random policy

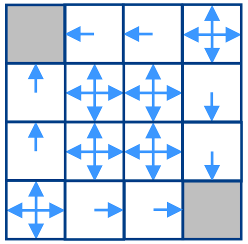
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



$k = 2$

0.0	-1.75	-2.0	-2.0
-1.75	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.75
-2.0	-2.0	-1.75	0.0



and so on…

Source : UCL Course on RL

Dynamic Programming

Policy Evaluation

eg. Evaluating a random policy in the Small Gridworld

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

Starting from each step…

$$-1 = \frac{(-1+0)+(-1+0)+(-1+0)+(-1+0)}{4}$$

Source : UCL Course on RL

Dynamic Programming

Policy Evaluation

eg. Evaluating a random policy in the Small Gridworld

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

Starting from each step…

$$-1 = \frac{(-1+0)+(-1+0)+(-1+0)+(-1+0)}{4}$$

Now update…

Source : UCL Course on RL

Dynamic Programming

Policy Evaluation

eg. Evaluating a random policy in the Small Gridworld

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

Starting from each step…

$$-2 = \frac{(-1-1)+(-1-1)+(-1-1)+(-1-1)}{4}$$

$$-1.75 = \frac{(-1-1)+(-1-1)+(-1-1)+(-1)}{4}$$

Now update…

Source : UCL Course on RL

Dynamic Programming

Policy Evaluation

eg. Evaluating a random policy in the Small Gridworld

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.75	-2.0	-2.0
-1.75	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.75
-2.0	-2.0	-1.75	0.0

Starting from each step…

$$-2 = \frac{(-1-1)+(-1-1)+(-1-1)+(-1-1)}{4}$$

$$-1.75 = \frac{(-1-1)+(-1-1)+(-1-1)+(-1)}{4}$$

Source : UCL Course on RL

Dynamic Programming

Policy Iteration

How do we improve a policy?

- Given a policy π
  - Evaluate the policy π,

$$v_\pi(s) = E(R_{t+1} + \gamma R_{t+2} + \cdots | S_t = s)$$

  - Improve the policy by acting greedily with respect to v_π,

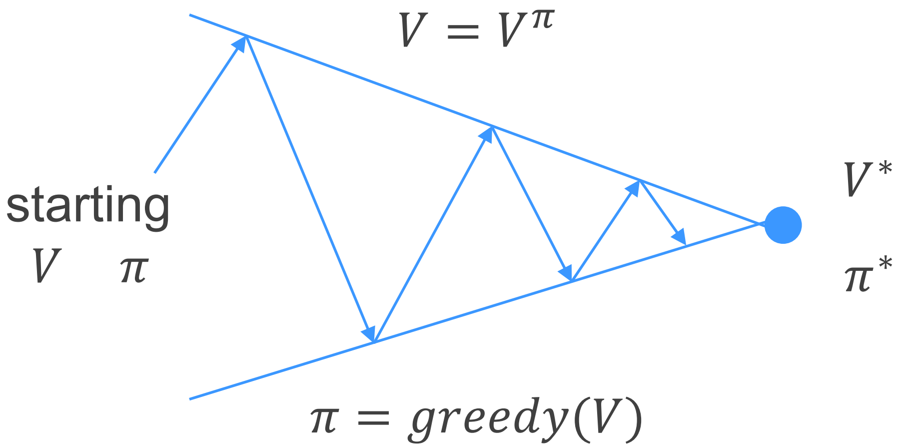
$$\pi' = greedy(v_\pi)$$

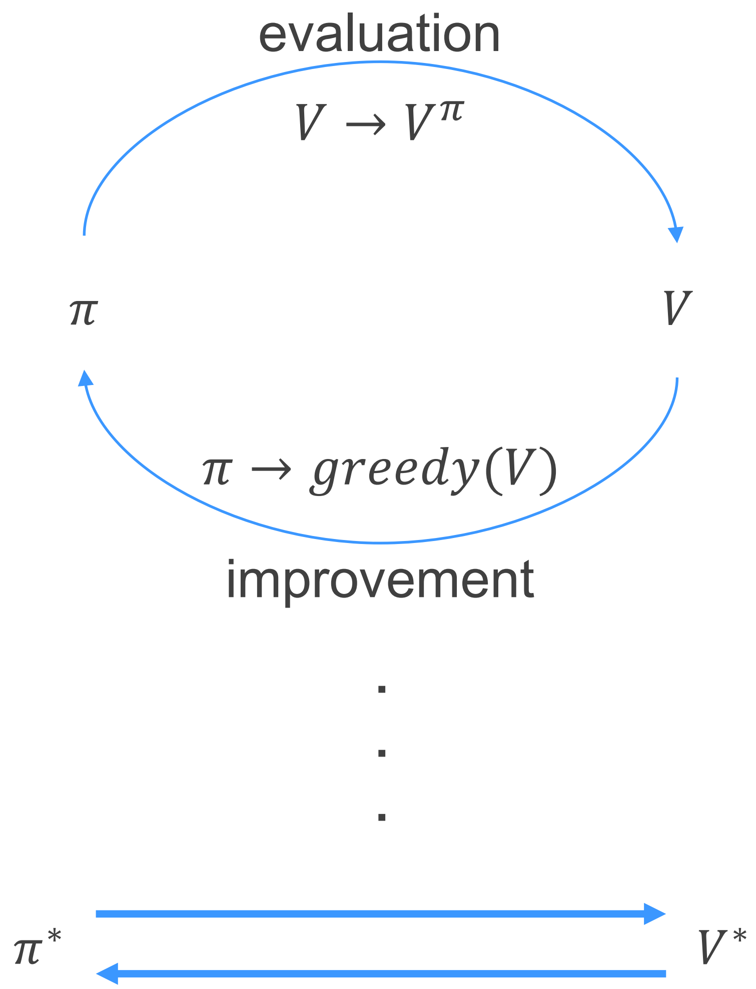
- In Small Gridworld improved policy was optimal, $\pi' = \pi^*$
- In general, need more iterations of improvement / evaluation
- But this process of policy iteration always converges to π^*

Source : UCL Course on RL

Dynamic Programming

Policy Iteration





Policy evaluation Estimate v_{π}
  Iterative policy evaluation
Policy improvement Generate $\pi' \geq \pi$
  Greedy policy improvement

Source : UCL Course on RL

Dynamic Programming

Policy Iteration

eg. Jack's Car Rental

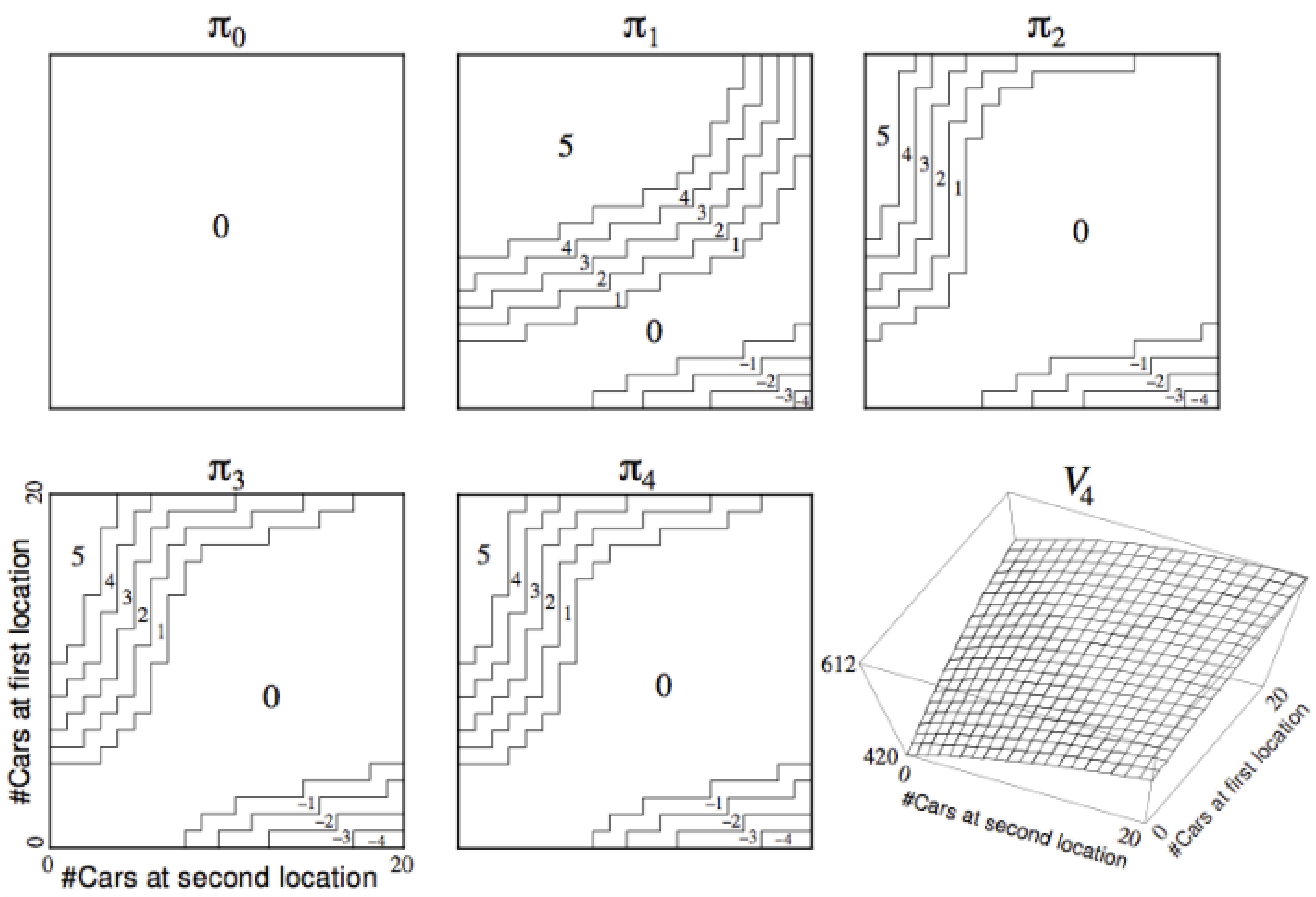
- States : Two locations, maximum of 20 cars at each
- Actions : Move up to 5 cars between locations overnight
- Reward : $10 for each car rented (must be available)
- Transitions : Cars returned and requested randomly
    - Poisson distribution, *n* returns / requests with probability $\frac{\rho^n}{n!}e^{-\rho}$
    - 1st location : average requests = 3, average returns = 3
    - 2nd location : average requests = 4, average returns = 2

Source : UCL Course on RL

Dynamic Programming

Policy Iteration

eg. Jack's Car Rental



Source : UCL Course on RL

Dynamic Programming

Policy Improvement

- Consider a deterministic policy, $a = \pi(s)$
- We can *improve* the policy by acting greedily,

$$\pi'(s) = \underset{a \in A}{\operatorname{argmax}}\, q_\pi(s, a)$$

- This improves the value from any state s over one step,

$$q_\pi(s, \pi'(s)) = \max_{a \in A} q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

- If therefore improves the value function, $v_{\pi'}(s) \geq v_\pi(s)$

$$\begin{aligned} v_\pi(s) \leq q_\pi(s, \pi'(s)) &= E_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\ &\leq E_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) | S_t = s] \\ &\leq E_{\pi'}[R_{t+1} + \gamma R_{t+2} + \cdots | S_t = s] = v_{\pi'}(s) \end{aligned}$$

Source : UCL Course on RL

Dynamic Programming

Policy Improvement

- If improvements stop,

$$q_\pi(s, \pi'(s)) = \max_{a \in A} q_\pi(s, a) = q_\pi(s, \pi(s)) = v_\pi(s)$$

- Then the Bellman optimality equation has been satisfied,

$$v_\pi(s) = \max_{a \epsilon A} q_\pi(s, a)$$

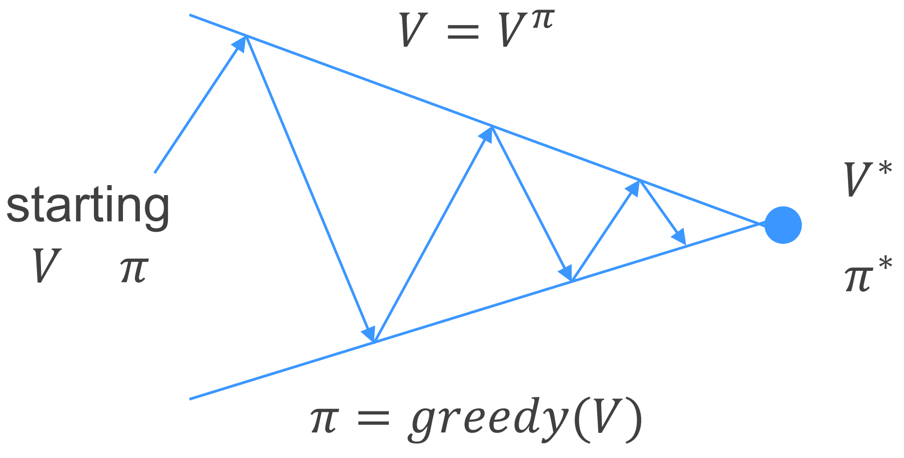
- Therefore $v_\pi(s) = v_*(s)$ for all $s \in S$

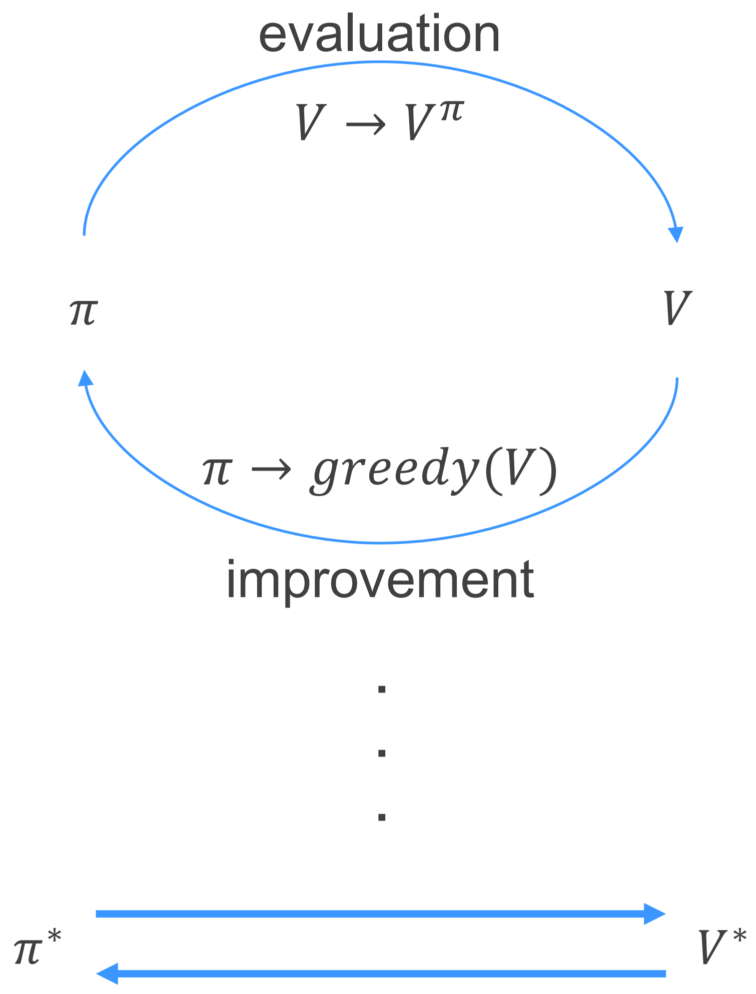
- So π is an optimal policy

Source : UCL Course on RL

Dynamic Programming

Generalised Policy Iteration





Policy evaluation Estimate v_π

Any policy evaluation algorithm

Policy improvement Generate $\pi' \geq \pi$

Any policy improvement algorithm

Source : UCL Course on RL

Dynamic Programming

Deterministic Value Iteration

- If we know the solution to subproblems $v_*(s')$
- Then the solution $v_*(s')$ can be found by one-step lookahead
- The idea of value iteration is to apply these updates iteratively
- Intuition : Start with final rewards and work backwards
- Still works with loopy, stochastic Markov Decision Processes

Source : UCL Course on RL

Dynamic Programming

Value Iteration

eg. Shortest Path Problem

g			

Problem

g	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

g	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

g	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

g	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

g	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

g	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

g	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

Source : UCL Course on RL

Dynamic Programming

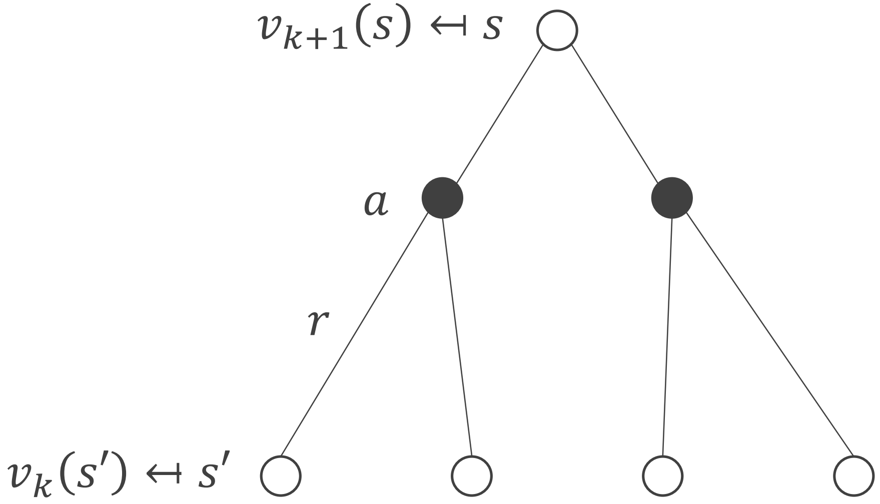
Value Iteration

- Problem : Find optimal policy π
- Solution : Iterative application of the Bellman optimality backup
- $v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_*$
- Using synchronous backups
  - At each iteration $k + 1$
  - For all states $s \in S$
  - Update $v_{k+1}(s)$ from $v_k(s')$
- Unlike policy iteration, there is no explicit policy
- Intermediate value functions may not correspond to any policy

Source : UCL Course on RL

Dynamic Programming

Value Iteration



$$v_{k+1}(s) = \max_{a \in A} \left(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s') \right)$$

$$\vec{v}_{k+1} = \max_{a \in A} \left(\vec{R}^{\vec{a}} + \gamma \vec{P}^{\vec{a}} \vec{v}_k \right)$$

Source : UCL Course on RL

Dynamic Programming

Extensions to Dynamic Programming

- Synchronous Dynamic Programming Algorithms

Problem	The Bellman Equation	Algorithm
Prediction	The Bellman Expectation Equation	Iterative Policy Evaluation
Control	The Bellman Expectation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

- Algorithms are based on state-value function $v_\pi(s)$ or $v_*(s)$

- Could also apply to action-value function $q_\pi(s,a)$ or $q_*(s,a)$

- Others :
  Asynchronous Dynamic Programming, In-Place Dynamic Programming, Prioritised Sweeping, Real-Time Dynamic Programming, Full-Width Backups, Sample Backups, Approximate Dynamic Programming

Source : UCL Course on RL

Reference

Reference

UCL Course on RL
http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html

Simple Beginner's guide to Reinforcement Learning & its implementation
https://www.analyticsvidhya.com/blog/2017/01/introduction-to-reinforcement-learning-implementation/

Thank you