Reinforcement Learning

PROF XIAOHUI XIE
SPRING 2019

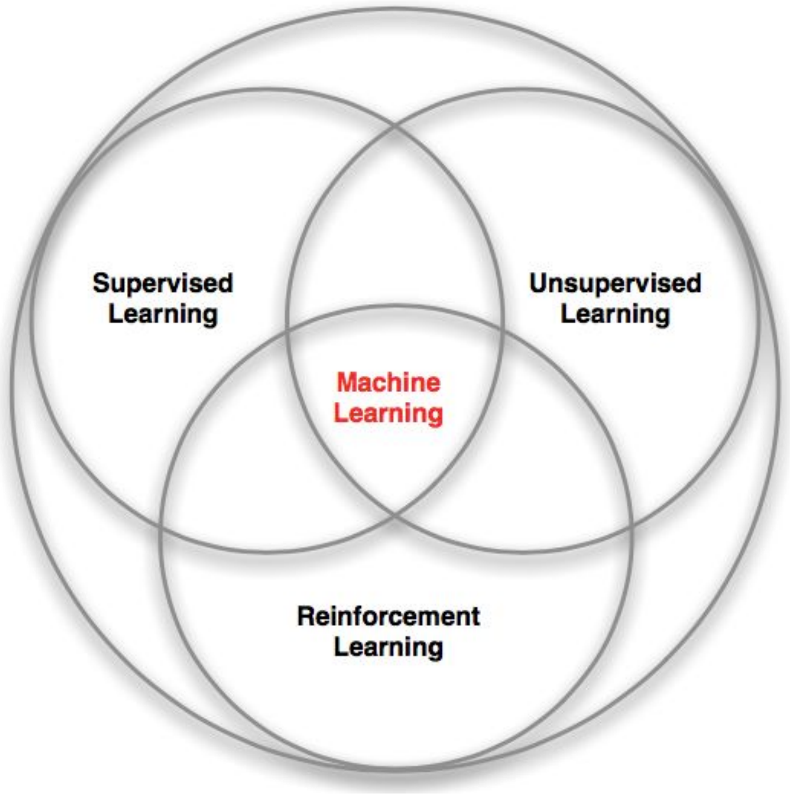
CS 273P Machine Learning and Data Mining

Slides courtesy of Sameer Singh, David Silver

Machine Learning

- Intro to Reinforcement Learning
- Markov Processes
- Markov Reward Processes
- Markov Decision Processes

Reinforcement Learning



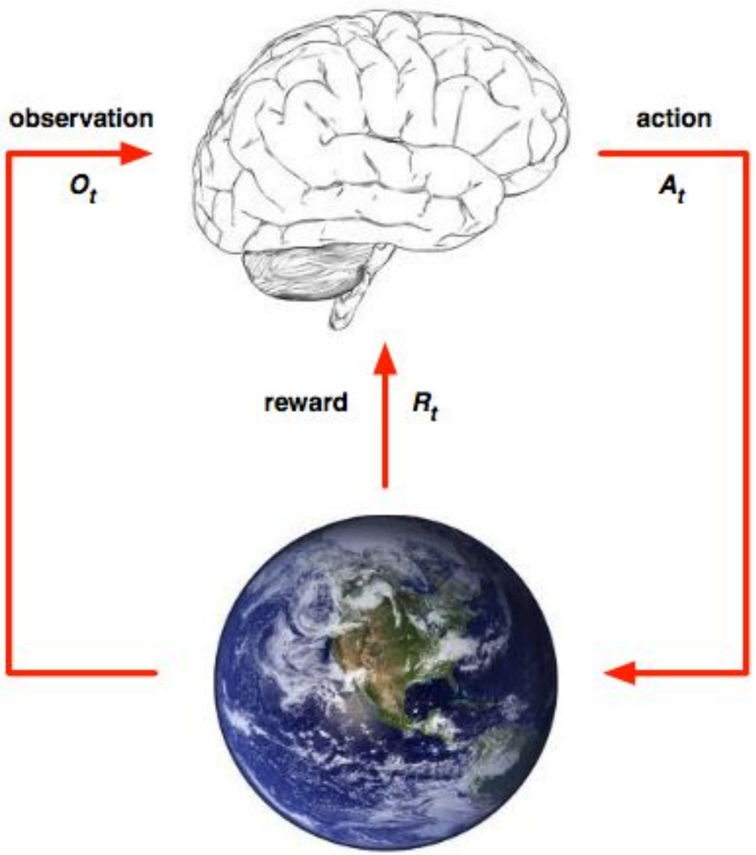
What makes it different?

No direct supervision, only rewards
Feedback is delayed, not instantaneous
Time really matters, i.e. data is sequential
Agent’s actions affect what data it will receive

Examples

- Fly stunt maneuvers in a helicopter
- Defeat the world champion at Backgammon or Go
- Manage an investment portfolio
- Control a power station
- Make a humanoid robot walk
- Play many different Atari games better than humans

Agent-Environment Interface



Agent

- decides on an action
- receives next observation
- receives next reward

Environment

- executes the action
- computes next observation
- computes next reward

Reward, R_t

How well the agent is doing

+, positive (Good)
-, negative (Bad)

Nothing about WHY it is doing well, could have little to do with A_{t-1}

Agent is trying to maximize its cumulative reward

Example of Rewards

- Fly stunt maneuvers in a helicopter
  - +ve reward for following desired trajectory
  - −ve reward for crashing
- Defeat the world champion at Backgammon
  - +/−ve reward for winning/losing a game
- Manage an investment portfolio
  - +ve reward for each $ in bank
- Control a power station
  - +ve reward for producing power
  - −ve reward for exceeding safety thresholds
- Make a humanoid robot walk
  - +ve reward for forward motion
  - −ve reward for falling over
- Play many different Atari games better than humans
  - +/−ve reward for increasing/decreasing score

Sequential Decision Making

Actions have long term consequences

Rewards may be delayed

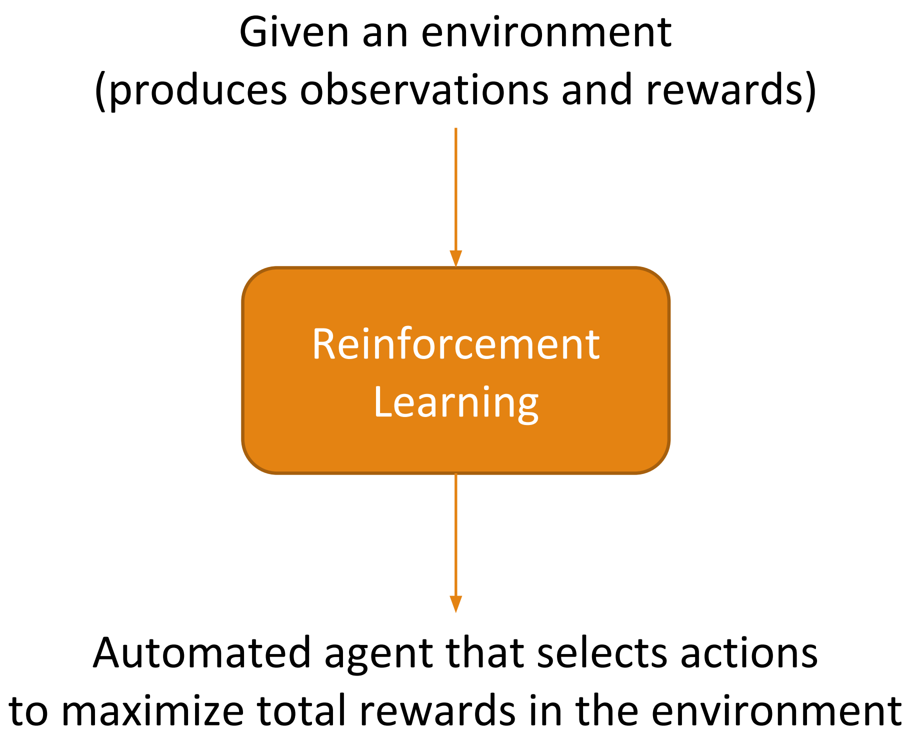
May be better to sacrifice short term reward for long term benefit

Examples

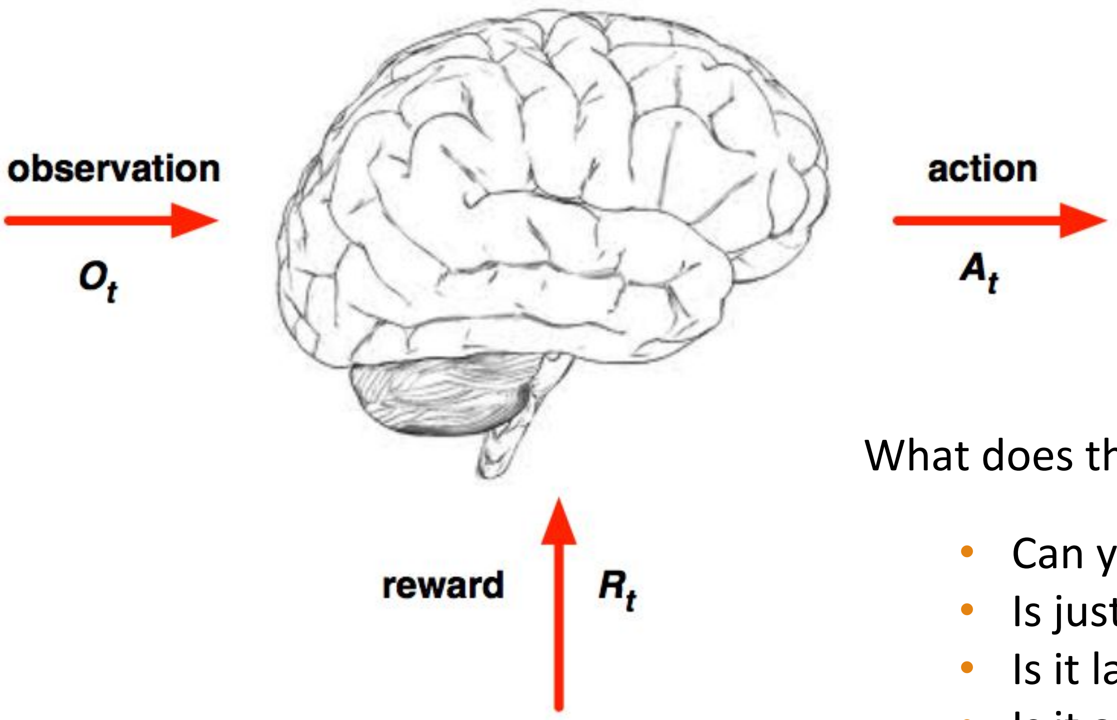
- A financial investment (may take months to mature)
- Refueling a helicopter (might prevent a crash later)
- Blocking opponent moves (might eventually help win)
- Spend a lot of money and go to college (earn more later)
- Don't commit crimes (rewarded by not going to jail)
- Get started on final project early (avoid stress later)

A key aspect of intelligence: How far ahead are you able to plan?

Reinforcement Learning



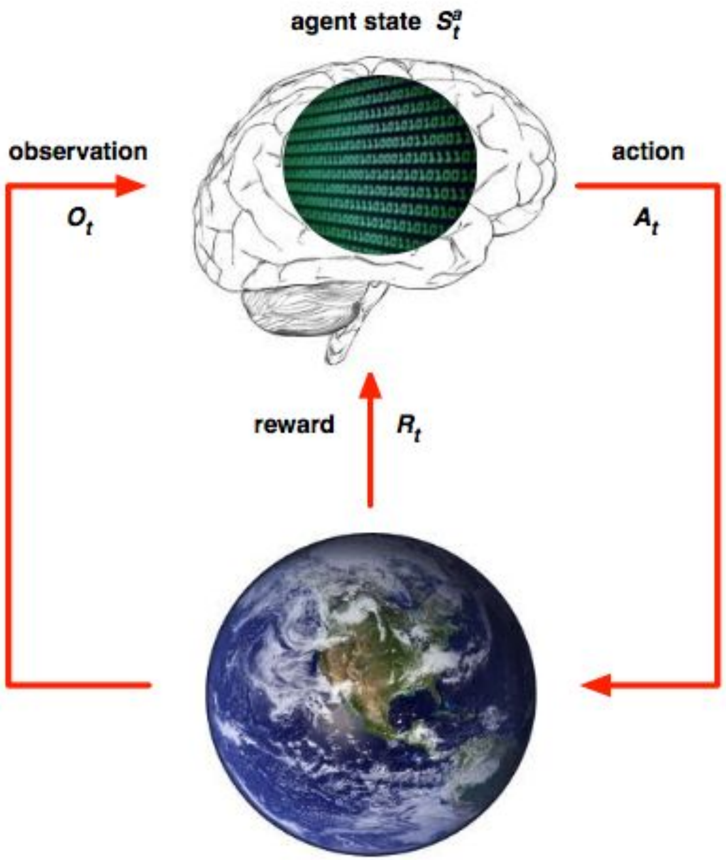
Let's look at the Agent



What does the choice of action depend on?

- Can you ignore O_t completely?
- Is just O_t enough? Or (O_t, A_t)?
- Is it last few observations?
- Is it all observations so far?

Agent State, S_t



History: everything that happened so far

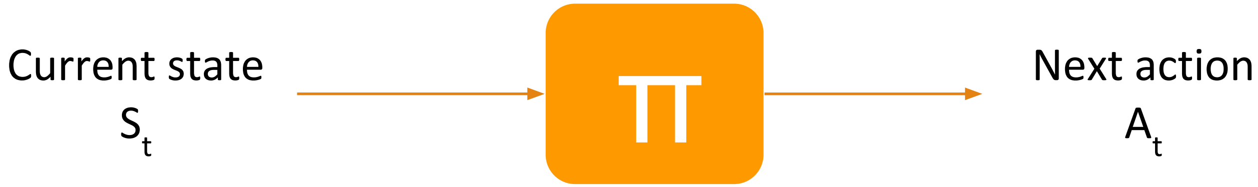
$H_t =$
$O_1R_1A_1O_2R_2A_2O_3R_3,\ldots,A_{t-1}O_tR_t$

State, S_t can be
O_t
O_tR_t
$A_{t-1}O_tR_t$
$O_{t-3}O_{t-2}O_{t-1}O_t$

In general, $S_t = f(H_t)$

You, as AI designer, specify this function

Agent Policy, π



Deterministic Policy: $A_t = \pi(S_t)$

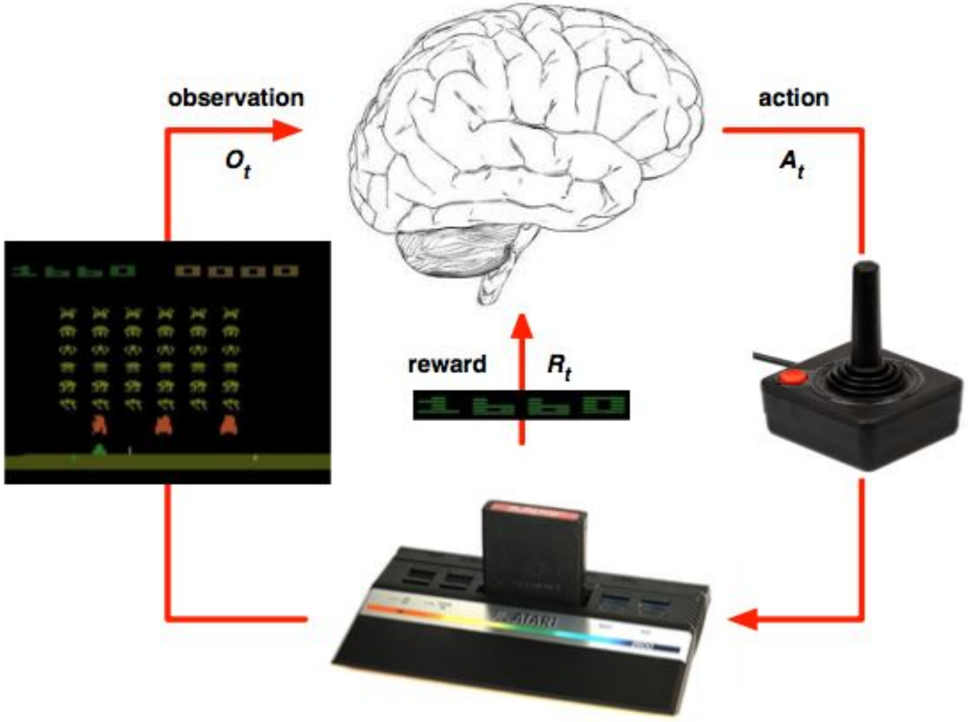
Stochastic Policy: $\pi(a|s) = P(A_t = a | S_t = s)$

Good policy: Leads to larger cumulative reward

Bad policy: Leads to worse cumulative reward

(we will explore this later)

Example: Atari



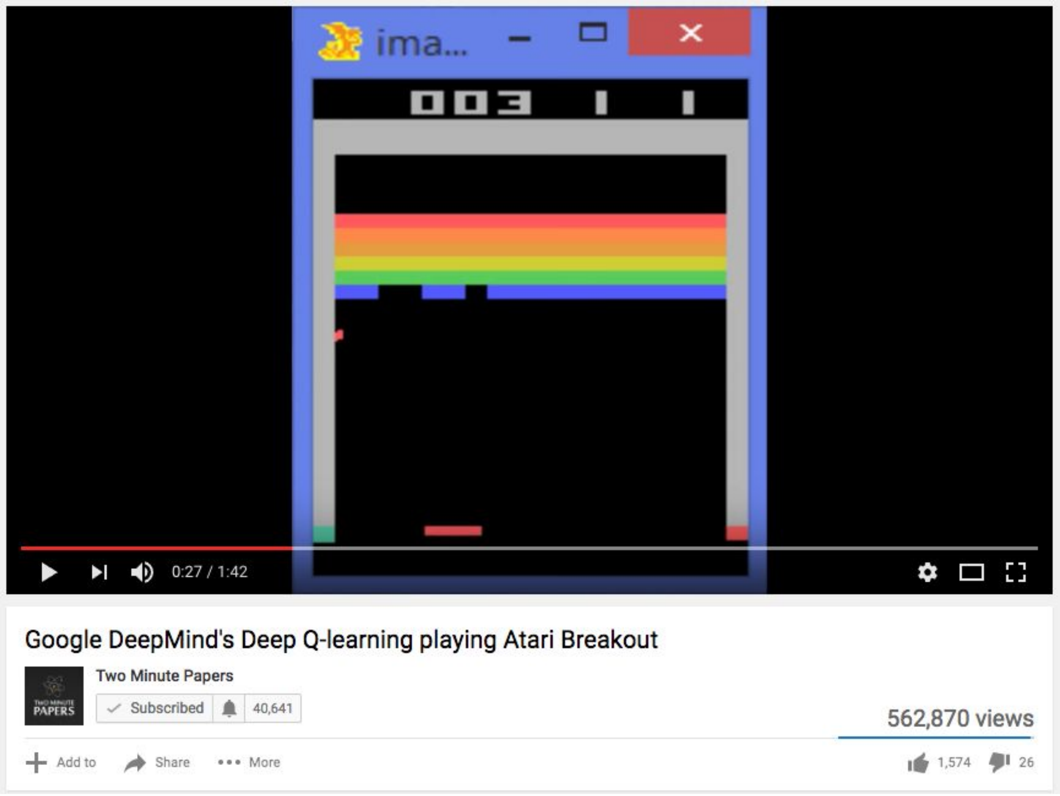
Rules are unknown

- What makes the score increase?

Dynamics are unknown

- How do actions change pixels?

Video Time!



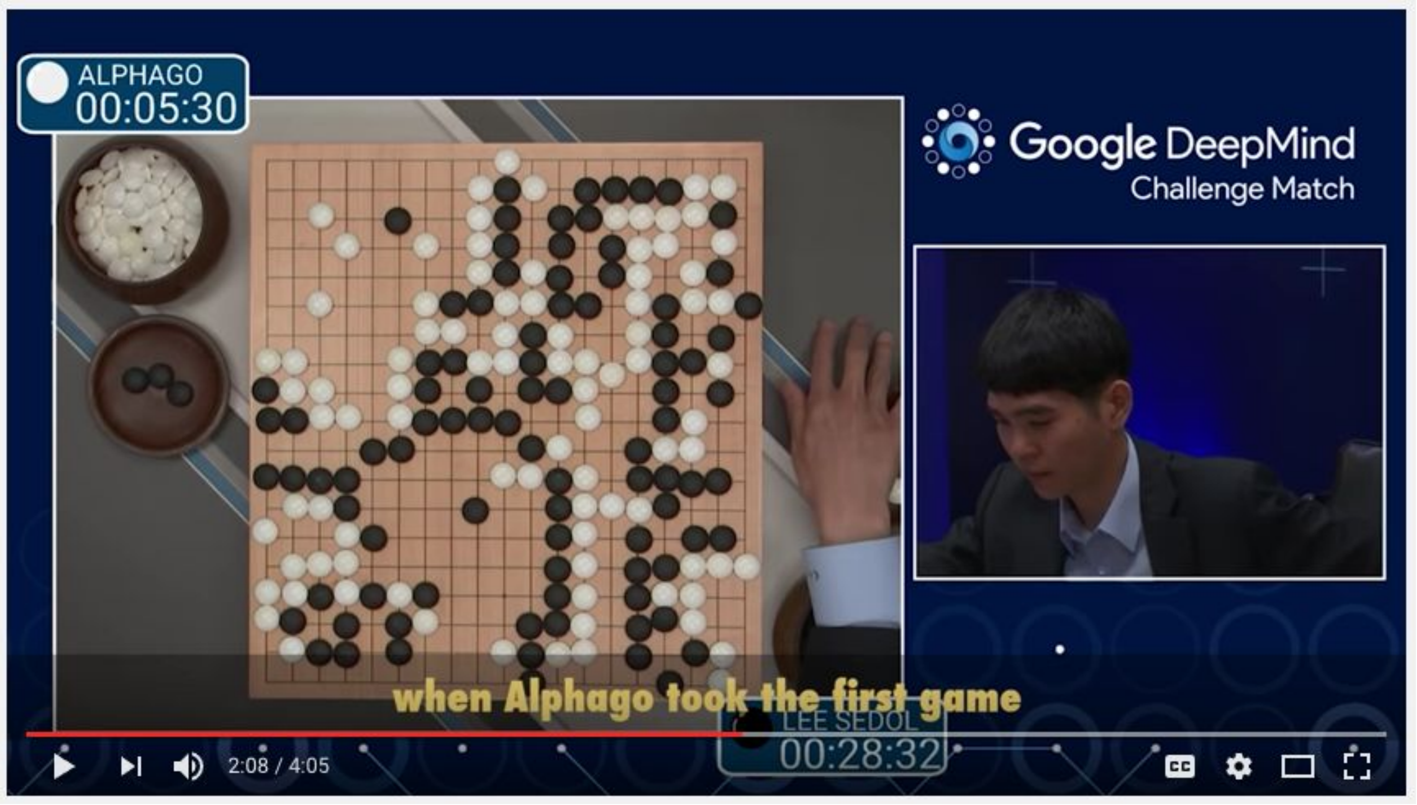
https://www.youtube.com/watch?v=V1eYniJ0Rnk

Example: Robotic Soccer

https://www.youtube.com/watch?v=CIF2SBVY-J0

AlphaGo

https://www.youtube.com/watch?v=I2WFvGl4y8c



Overview of Lecture

Machine Learning

Intro to Reinforcement Learning

Markov Processes

Markov Reward Processes

Markov Decision Processes

Markov Property

"The future is independent of the past given the present"

Markov Property

"The future is independent of the past given the present"

Definition

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 Property

"The future is independent of the past given the present"

Definition

A state S_t is *Markov* if and only if

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

- The state captures all relevant information from the history
- Once the state is known, the history may be thrown away
- i.e. The state is a sufficient statistic of the future

State Transition Matrix

For a Markov state s and successor state s', the *state transition probability* is defined by

$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

State Transition Matrix

For a Markov state s and successor state s', the *state transition probability* is defined by

$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

State transition matrix $\mathcal{P}$ defines transition probabilities from all states s to all successor states s',

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

where each row of the matrix sums to 1.

Markov Processes

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

Markov Processes

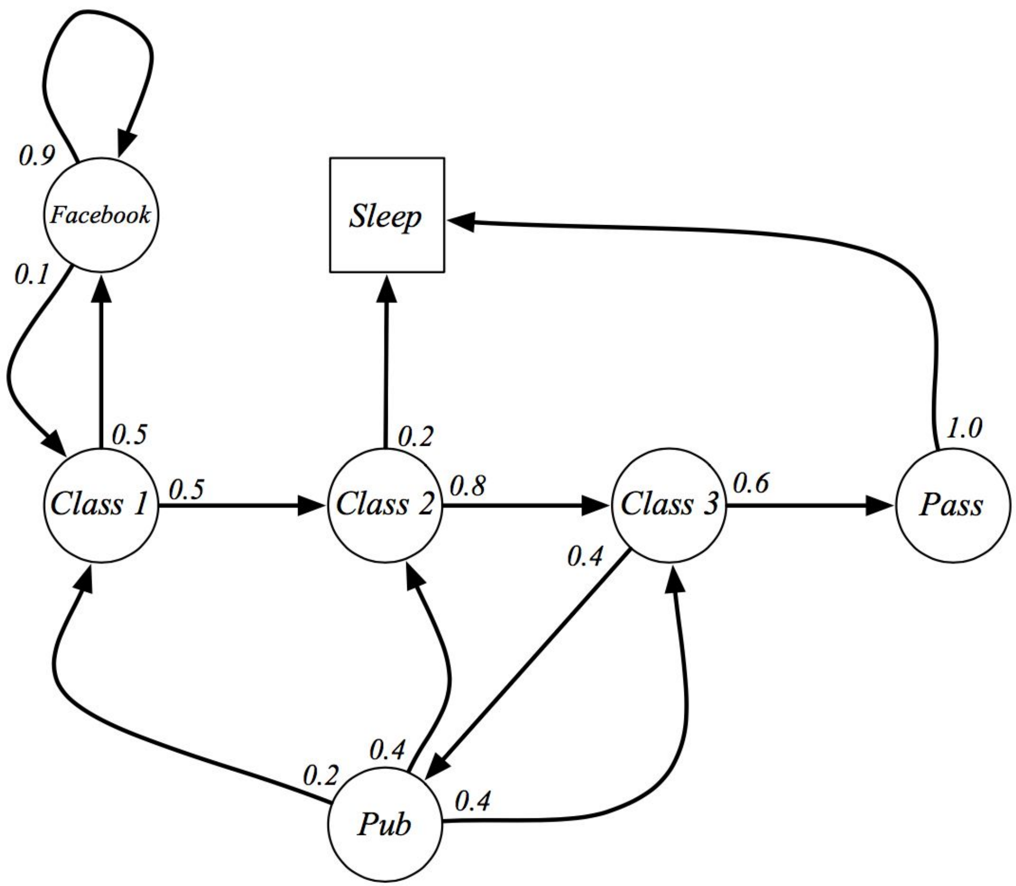
A Markov process is a memoryless random process, i.e. a sequence of random states $S_1, S_2, ...$ with the Markov property.

Definition

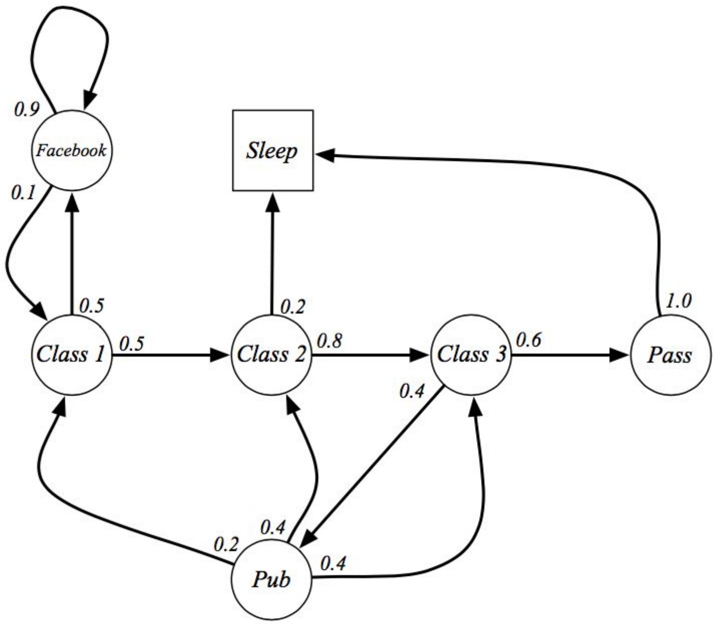
A *Markov Process* (or *Markov Chain*) is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix,
  $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s]$

Student Markov Chain



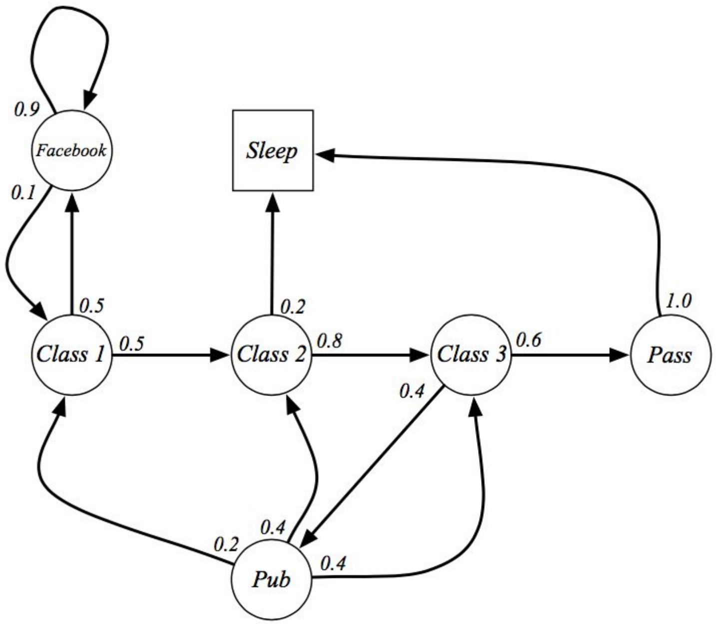
Student MC: Episodes



Sample episodes for Student Markov Chain starting from $S_1 = \text{C1}$

$$S_1, S_2, ..., S_T$$

Student MC: Episodes

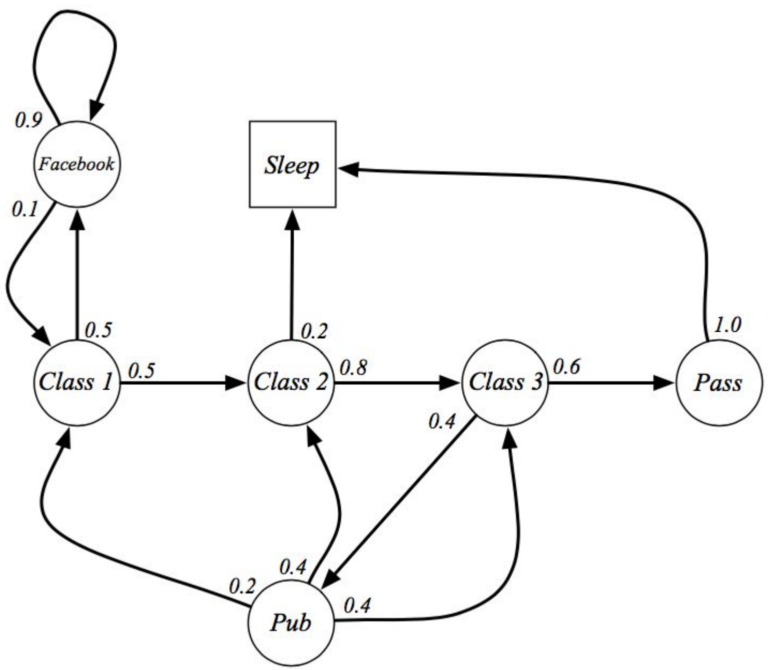


Sample episodes for Student Markov Chain starting from $S_1 = C1$

$$S_1, S_2, ..., S_T$$

- C1 C2 C3 Pass Sleep
- C1 FB FB C1 C2 Sleep
- C1 C2 C3 Pub C2 C3 Pass Sleep
- C1 FB FB C1 C2 C3 Pub C1 FB FB FB C1 C2 C3 Pub C2 Sleep

Student MC: Transition Matrix



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

Machine Learning

- Intro to Reinforcement Learning
- Markov Processes
- **Markov Reward Processes**
- Markov Decision Processes

Markov Reward Process

A Markov reward process is a Markov chain with values.

Markov Reward Process

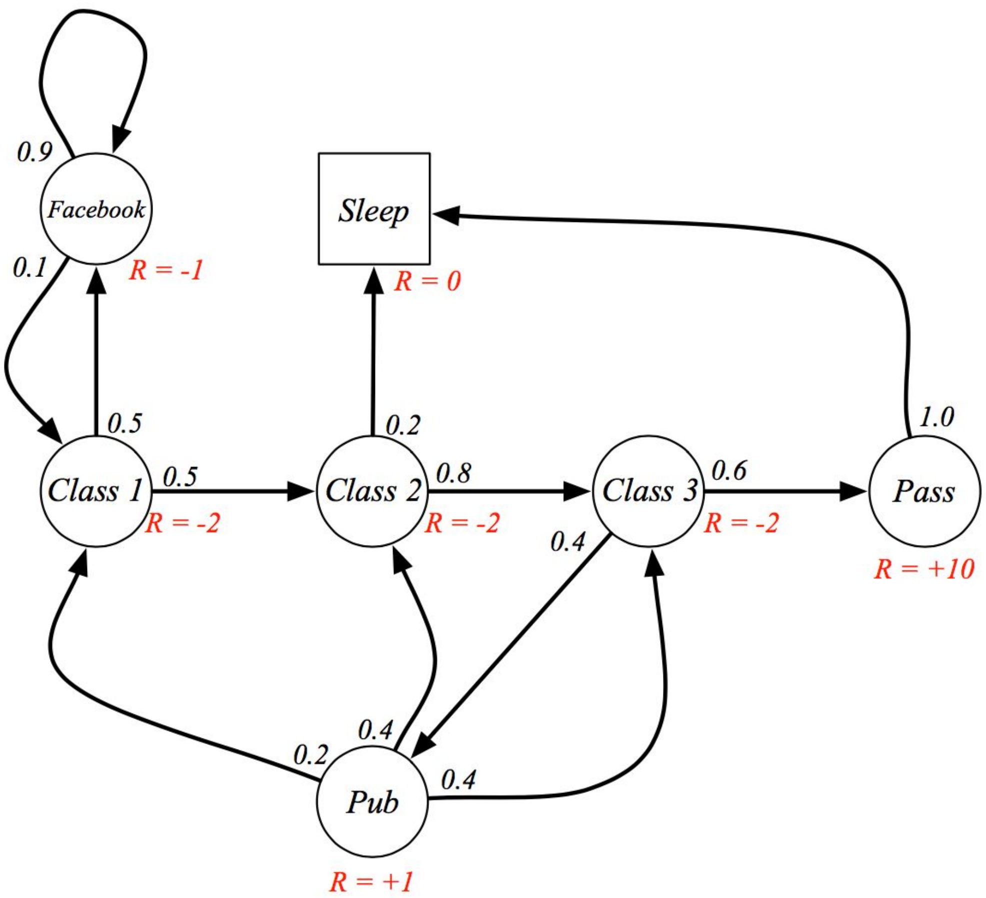
A Markov reward process is a Markov chain with values.

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, $\gamma \in [0, 1]$

The Student MRP



Return

Definition

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

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

Return

Definition

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

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

- The *discount* $\gamma \in [0, 1]$ is the present value of future rewards
- The value of receiving reward R after $k + 1$ time-steps is $\gamma^k R$.

Return

Definition

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

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

- The *discount* $\gamma \in [0, 1]$ is the present value of future rewards
- The value of receiving reward R after $k + 1$ time-steps is $\gamma^k R$.
- This values immediate reward above delayed reward.
  - γ close to 0 leads to "myopic" evaluation
  - γ close to 1 leads to "far-sighted" evaluation

Why discount?

Most Markov reward and decision processes are discounted. Why?

Why discount?

Most Markov reward and decision processes are discounted. Why?

- Mathematically convenient to discount rewards
- Avoids infinite returns in cyclic Markov processes

Why discount?

Most Markov reward and decision processes are discounted. Why?

- Mathematically convenient to discount rewards
- Avoids infinite returns in cyclic Markov processes
- Uncertainty about the future may not be fully represented
- If the reward is financial, immediate rewards may earn more interest than delayed rewards

Why discount?

Most Markov reward and decision processes are discounted. Why?

- Mathematically convenient to discount rewards
- Avoids infinite returns in cyclic Markov processes
- Uncertainty about the future may not be fully represented
- If the reward is financial, immediate rewards may earn more interest than delayed rewards
- Animal/human behaviour shows preference for immediate reward

Why discount?

Most Markov reward and decision processes are discounted. Why?

- Mathematically convenient to discount rewards
- Avoids infinite returns in cyclic Markov processes
- Uncertainty about the future may not be fully represented
- If the reward is financial, immediate rewards may earn more interest than delayed rewards
- Animal/human behaviour shows preference for immediate reward
- It is sometimes possible to use *undiscounted* Markov reward processes (i.e. $\gamma = 1$), e.g. if all sequences terminate.

Value Function

The value function $v(s)$ gives the long-term value of state s

Value Function

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

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

Student MRP: Returns

Sample returns for Student MRP:
Starting from $S_1 = \text{C1}$ with $\gamma = \frac{1}{2}$

$$G_1 = R_2 + \gamma R_3 + ... + \gamma^{T-2} R_T$$

C1 C2 C3 Pass Sleep

C1 FB FB C1 C2 Sleep

C1 C2 C3 Pub C2 C3 Pass Sleep

C1 FB FB C1 C2 C3 Pub C1 ...

FB FB FB C1 C2 C3 Pub C2 Sleep

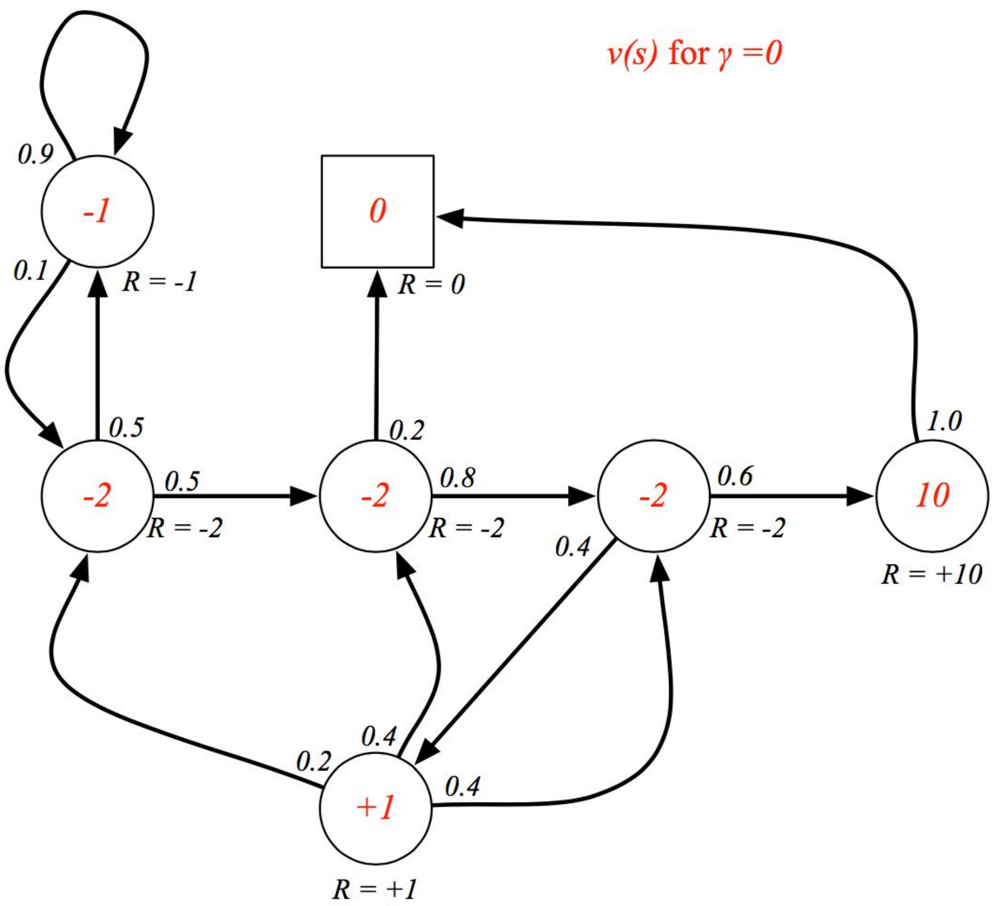
Student MRP: Returns

Sample returns for Student MRP:
Starting from $S_1 = C1$ with $\gamma = \frac{1}{2}$

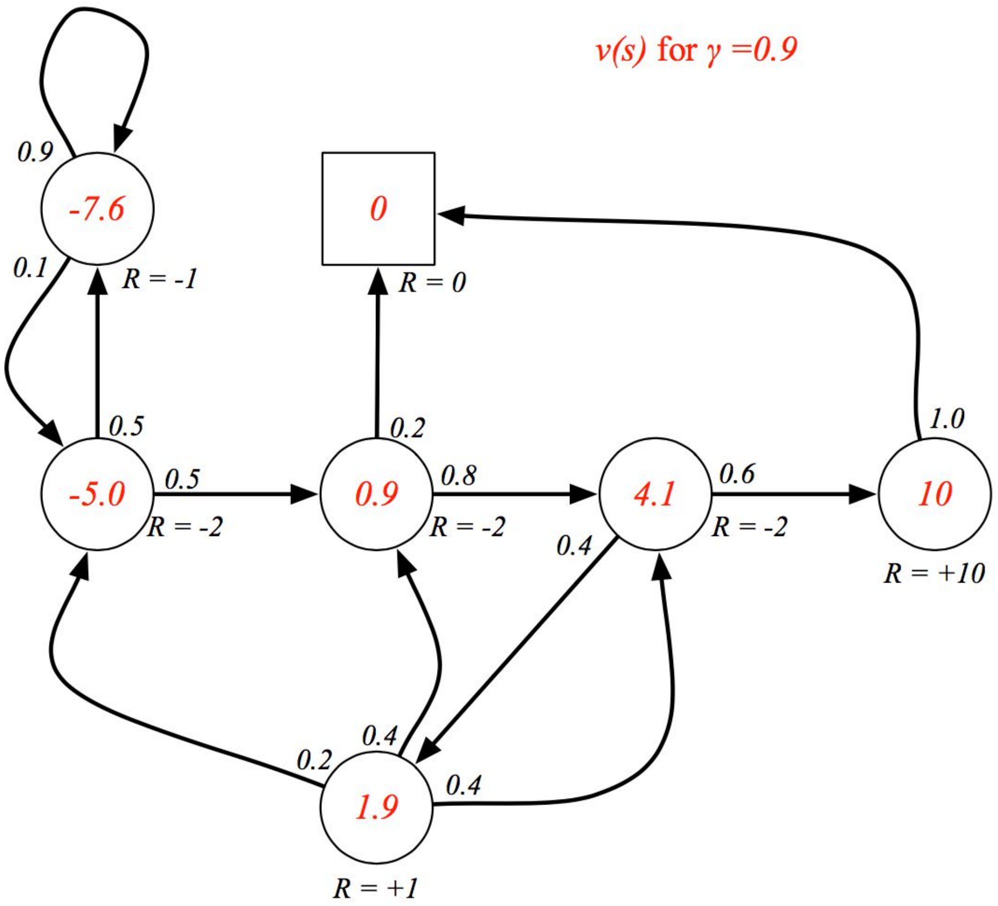
$$G_1 = R_2 + \gamma R_3 + ... + \gamma^{T-2} R_T$$

C1 C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 10 * \frac{1}{8}$	$=$	-2.25
C1 FB FB C1 C2 Sleep	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16}$	$=$	-3.125
C1 C2 C3 Pub C2 C3 Pass Sleep	$v_1 = -2 - 2 * \frac{1}{2} - 2 * \frac{1}{4} + 1 * \frac{1}{8} - 2 * \frac{1}{16} \dots$	$=$	-3.41
C1 FB FB C1 C2 C3 Pub C1 ...	$v_1 = -2 - 1 * \frac{1}{2} - 1 * \frac{1}{4} - 2 * \frac{1}{8} - 2 * \frac{1}{16} \dots$	$=$	-3.20
FB FB FB C1 C2 C3 Pub C2 Sleep			

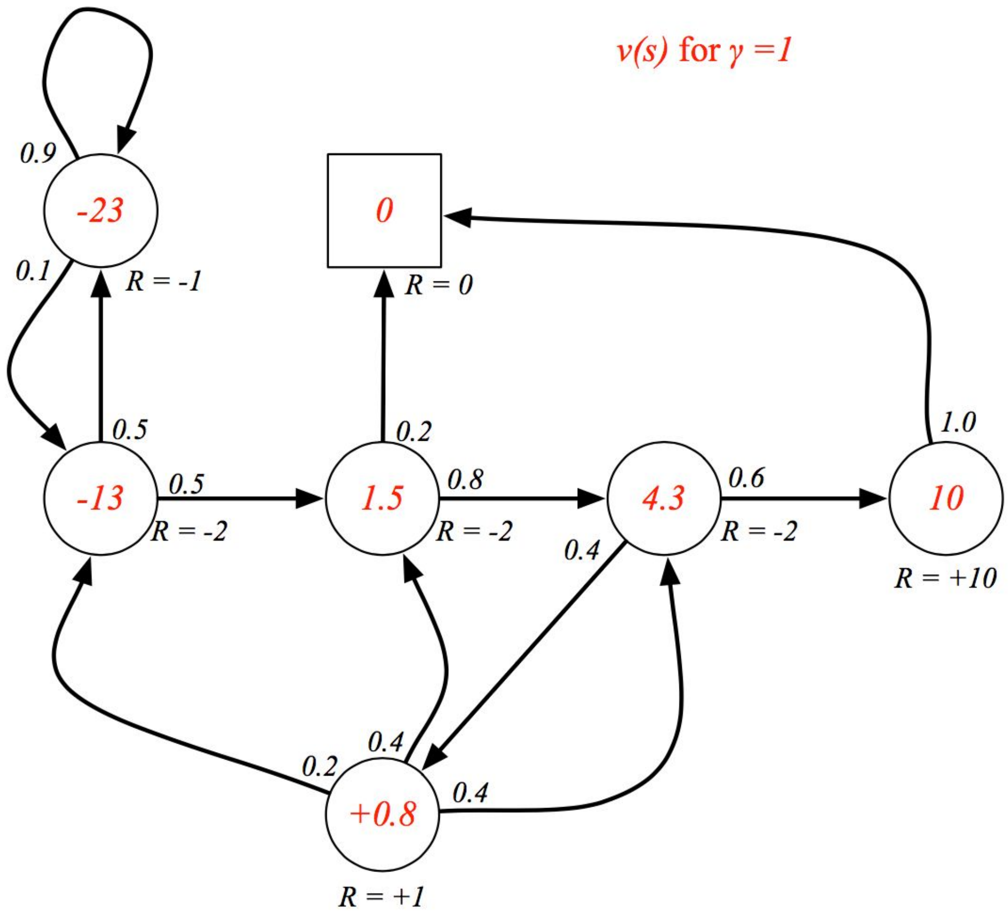
Student MRP: Value Function



Student MRP: Value Function



Student MRP: Value Function



Bellman Equations for MRP

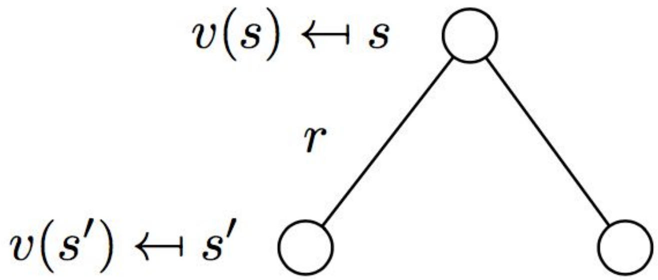
The value function can be decomposed into two parts:

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

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

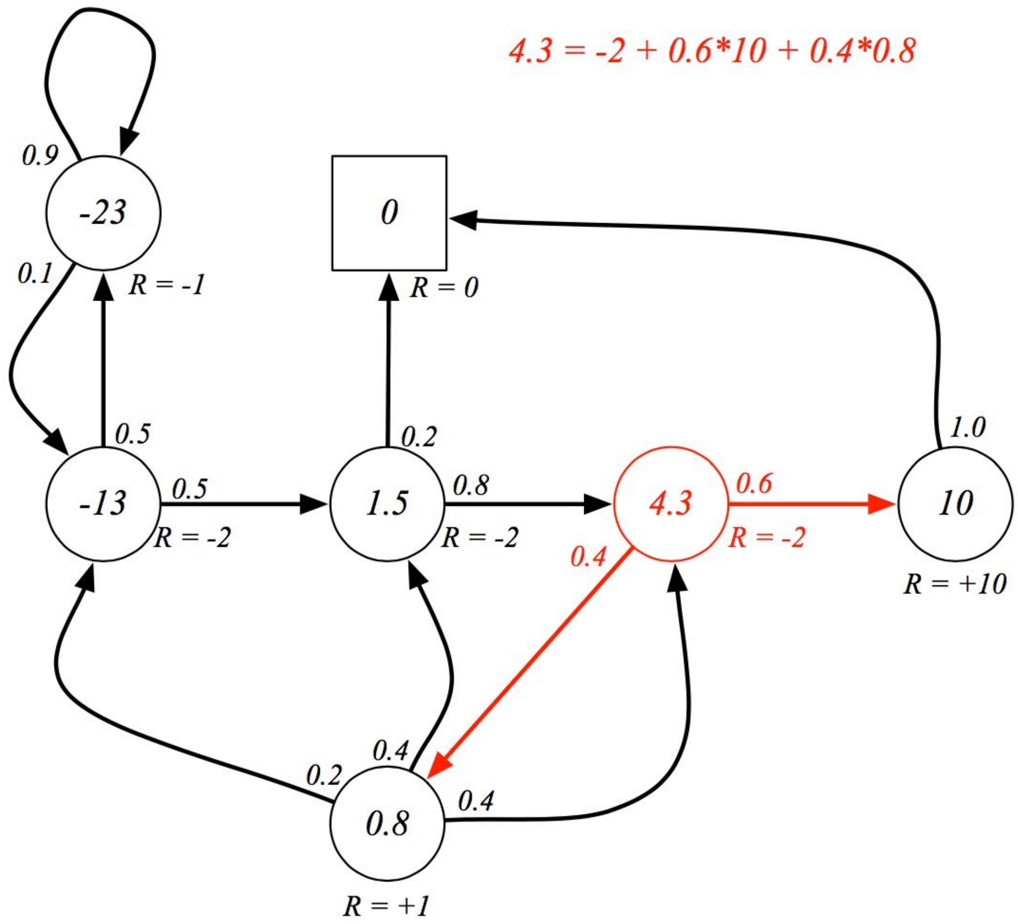
Backup Diagrams for MRP

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



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

Student MRP: Bellman Eq



Matrix Form of Bellman Eq

The Bellman equation can be expressed concisely using matrices,

$$v = \mathcal{R} + \gamma \mathcal{P} v$$

where v is a column vector with one entry per state

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

Solving the Bellman Equation

- The Bellman equation is a linear equation
- It can be solved directly:

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

Solving the Bellman Equation

- The Bellman equation is a linear equation
- It can be solved directly:

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

- Computational complexity is $O(n^3)$ for n states
- Direct solution only possible for small MRPs
- There are many iterative methods for large MRPs, e.g.
  - Dynamic programming
  - Monte-Carlo evaluation
  - Temporal-Difference learning

Dynamic Programming

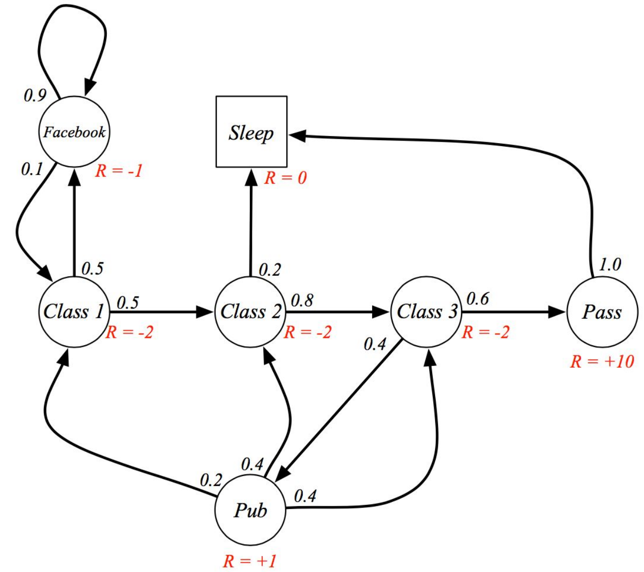
$v^2(C1) = -2 + \gamma\ (\ .5\ v^1(FB) + .5\ v^1(C2)\ )$

$v^2\ (FB) = -1 + \gamma\ (\ .9\ v^1(FB) + .1\ v^1(C1)\ )$

…

$v^3\ (FB) = -1 + \gamma\ (\ .9\ v^2(FB) + .1\ v^2(C1)\ )$

…



γ=0.5

	C1	C2	C3	Pa	Pub	FB	Slp
t=1	-2	-2	-2	10	1	-1	0
t=2	-2.75	-2.8	1.2	10	0	-1.55	0
t=3	-3.09	-1.52	1	10	0.41	-1.83	0
t=4	-2.84	-1.6	1.08	10	0.59	-1.98	0

Machine Learning

- Intro to Reinforcement Learning
- Markov Processes
- Markov Reward Processes
- **Markov Decision Processes**

Markov Decision Process

A Markov decision process (MDP) is a Markov reward process with decisions. It is an *environment* in which all states are Markov.

Markov Decision Process

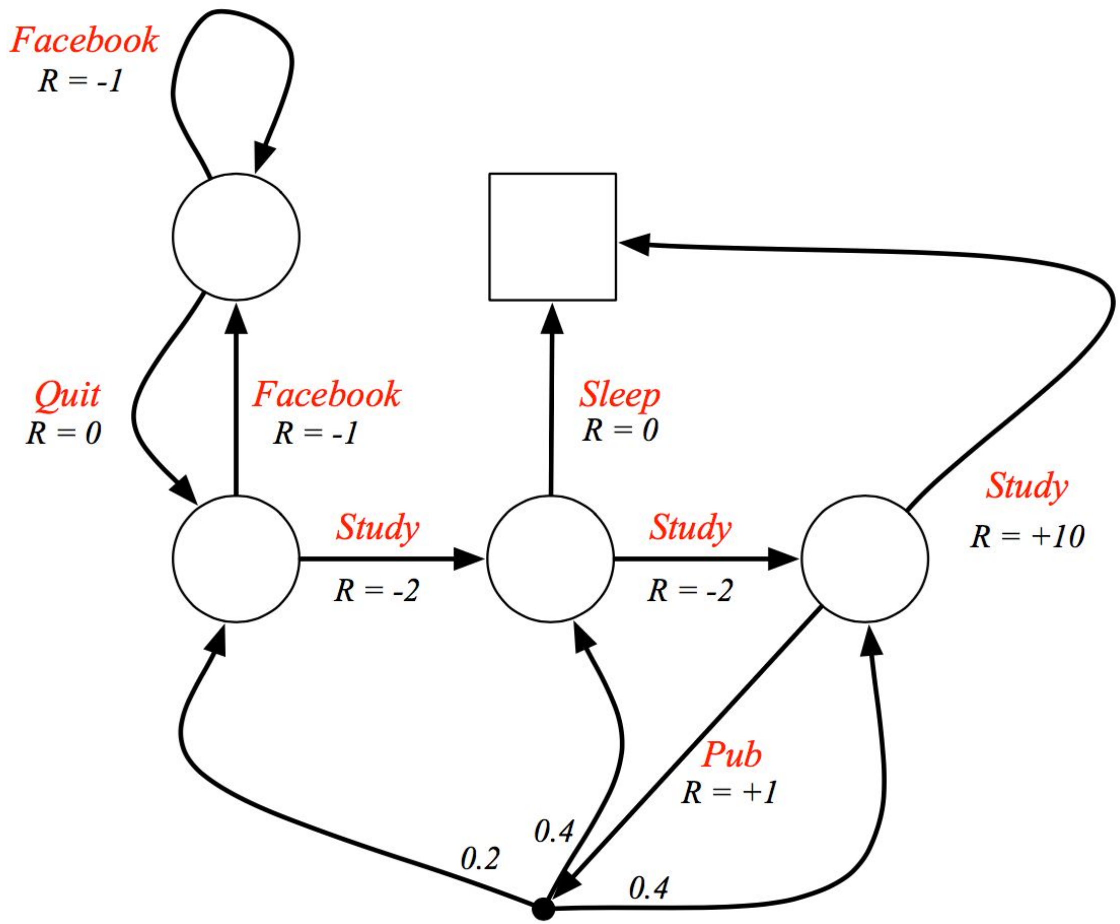
A Markov decision process (MDP) is a Markov reward process with decisions. It is an *environment* in which all states are Markov.

Definition

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

The Student MDP



Policies

Definition

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

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

Policies

Definition

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

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

- A policy fully defines the behaviour of an agent
- MDP policies depend on the current state (not the history)

Policies

Definition

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

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

- A policy fully defines the behaviour of an agent
- MDP policies depend on the current state (not the history)
- i.e. Policies are *stationary* (time-independent), $A_t \sim \pi(\cdot|S_t), \forall t > 0$

MPs $\longrightarrow$ MRPs $\longrightarrow$ MDPs

- Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π

MPs $\longrightarrow$ MRPs $\longrightarrow$ MDPs

- Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π
- The state sequence $S_1, S_2, ...$ is a Markov process $\langle \mathcal{S}, \mathcal{P}^{\pi} \rangle$
- The state and reward sequence $S_1, R_2, S_2, ...$ is a Markov reward process $\langle \mathcal{S}, \mathcal{P}^{\pi}, \mathcal{R}^{\pi}, \gamma \rangle$
- where

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

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

Value Function

Definition

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

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

Value Function

Definition

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

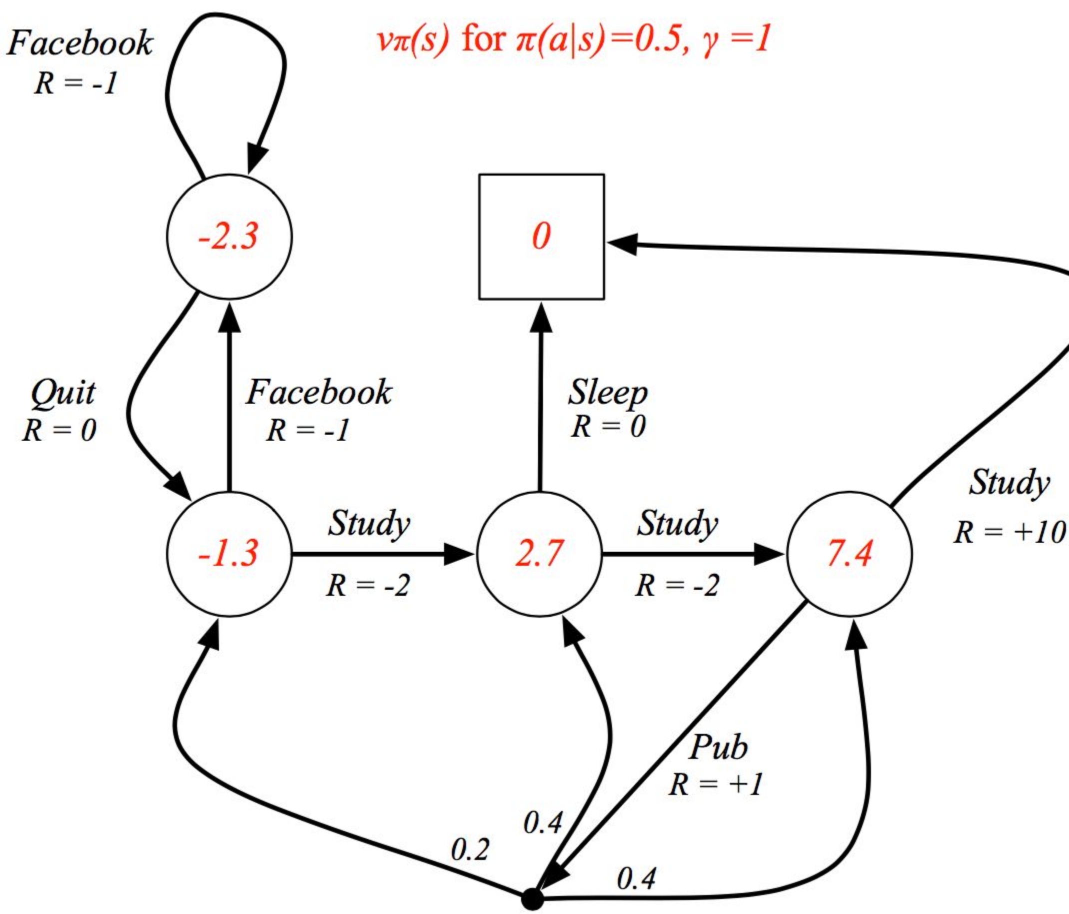
$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

Definition

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

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

Student MDP: Value Function



Bellman Expected Equation

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

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

Bellman Expected Equation

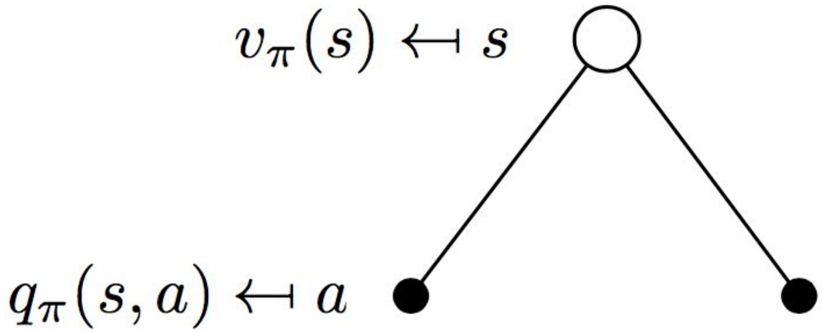
The state-value function can again be decomposed into immediate reward plus discounted value of successor state,

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

The action-value function can similarly be decomposed,

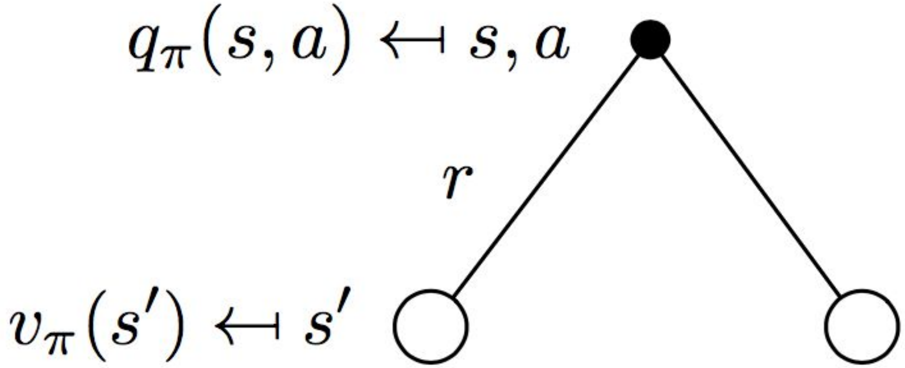
$$q_\pi(s, a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a \right]$$

Bellman Expected Equation, V



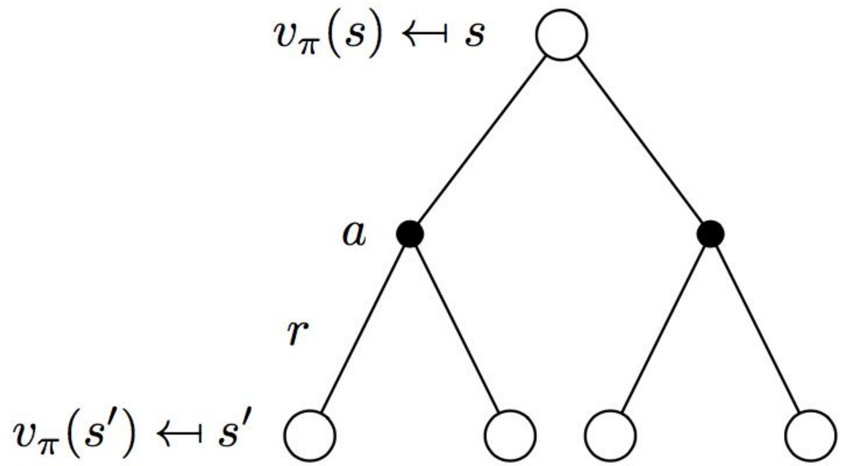
$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

Bellman Expected Equation, Q



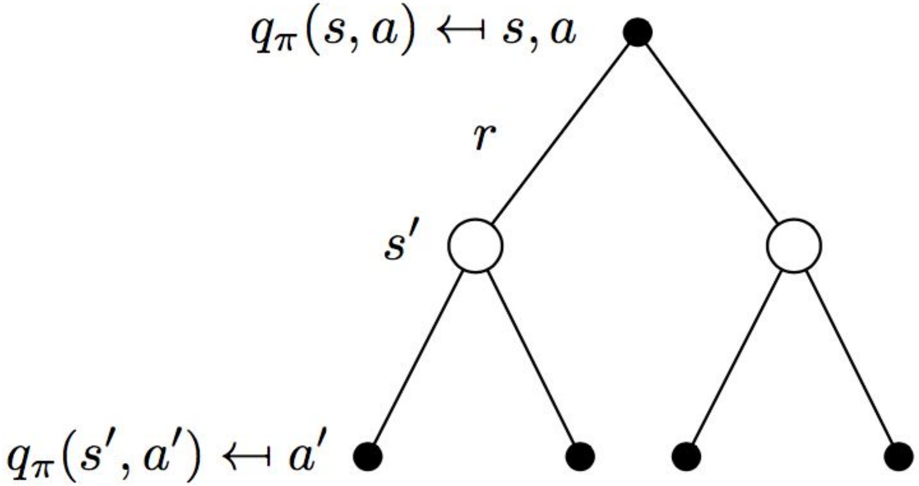
$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

Bellman Expected Equation, V



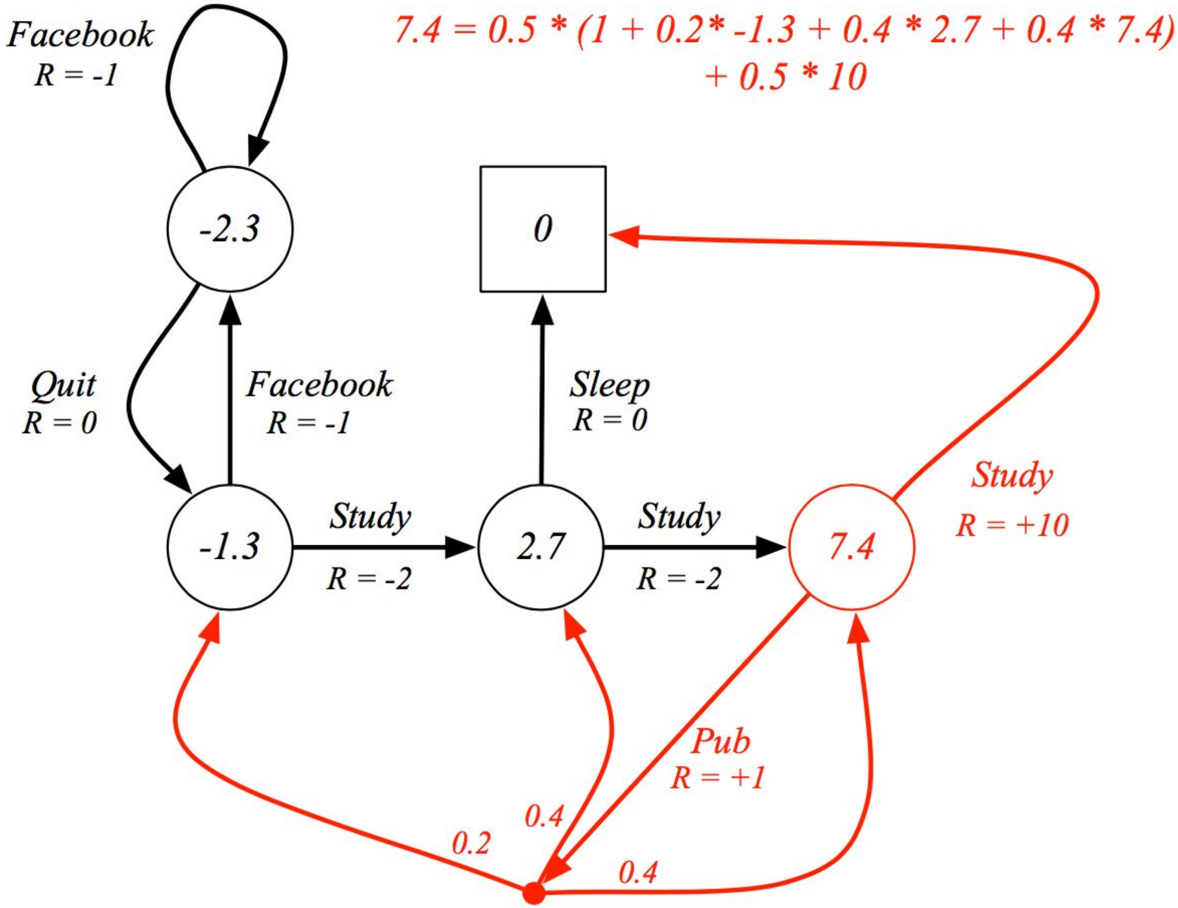
$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

Bellman Expected Equation, Q



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

Student MDP: Bellman Exp Eq.



Bellman Exp Eq: Matrix Form

The Bellman expectation equation can be expressed concisely using the induced MRP,

$$v_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi$$

with direct solution

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

Optimal Value Function

Definition

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

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

Optimal Value Function

Definition

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

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

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

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

Optimal Value Function

Definition

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

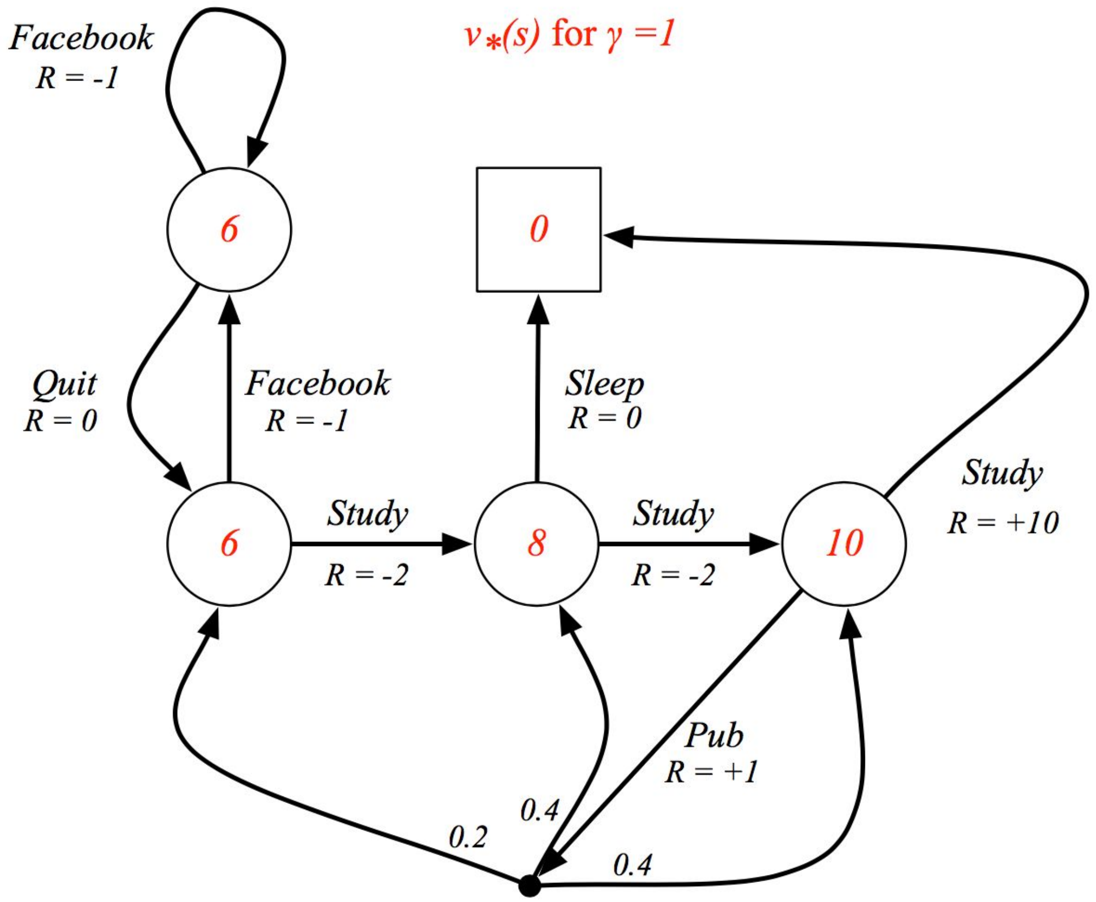
$$v_*(s) = \max_\pi v_\pi(s)$$

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

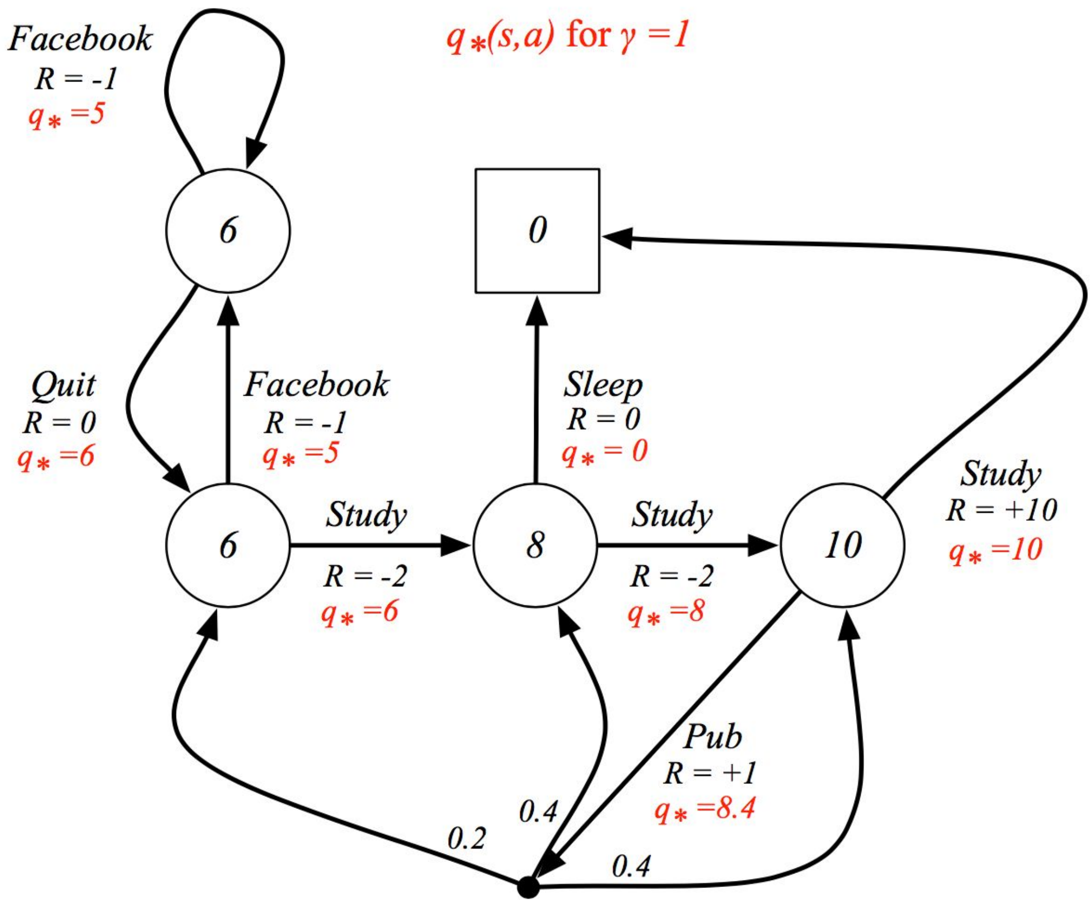
$$q_*(s, a) = \max_\pi q_\pi(s, a)$$

- The optimal value function specifies the best possible performance in the MDP.
- An MDP is "solved" when we know the optimal value fn.

Student MDP: Optimal V



Student MDP: Optimal Q



Optimal Policy

Define a partial ordering over policies

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

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

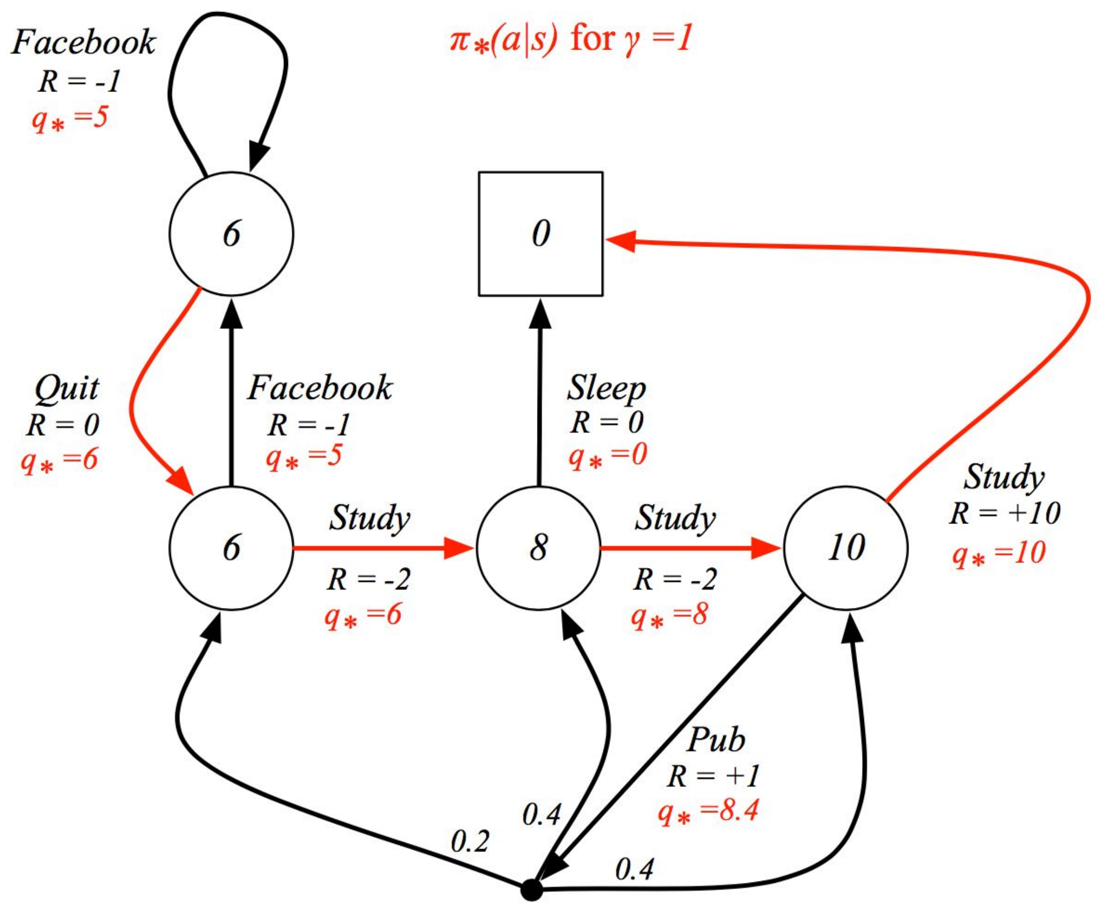
Finding Optimal Policy

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

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

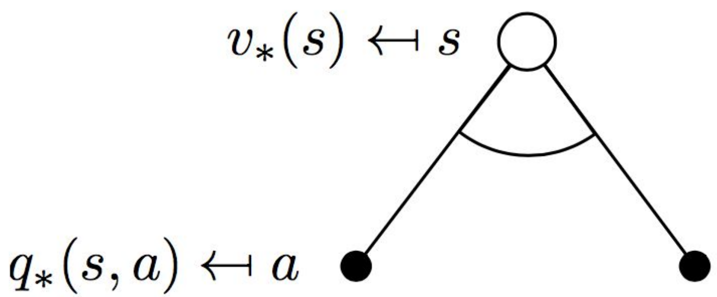
- There is always a deterministic optimal policy for any MDP
- If we know $q_*(s, a)$, we immediately have the optimal policy

Student MDP: Optimal Policy



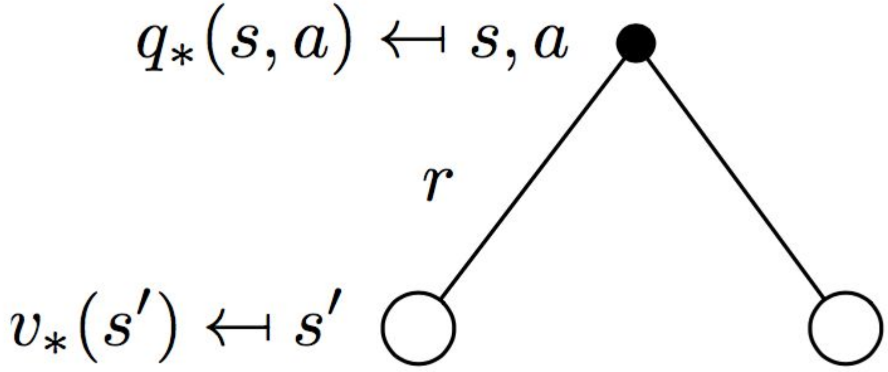
Bellman Optimality Eq, V

The optimal value functions are recursively related by the Bellman optimality equations:



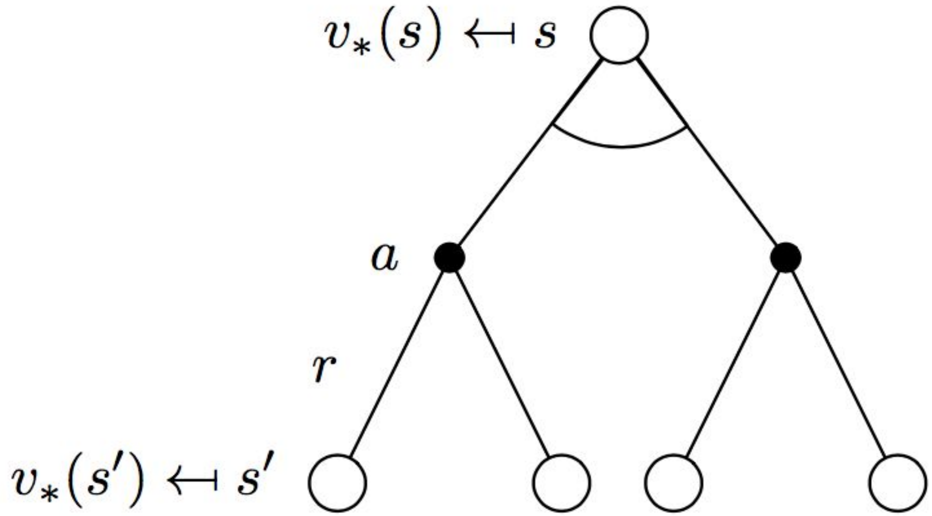
$$v_*(s) = \max_a q_*(s, a)$$

Bellman Optimality Eq, Q



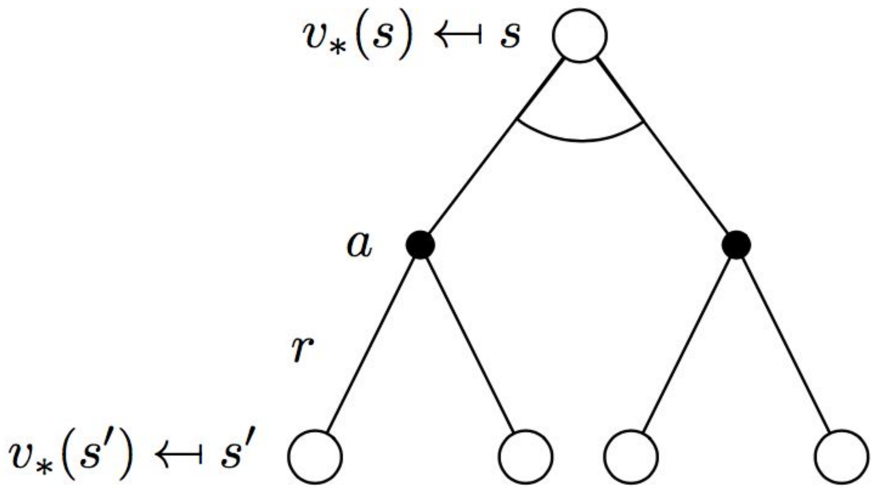
$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

Bellman Optimality Eq, V



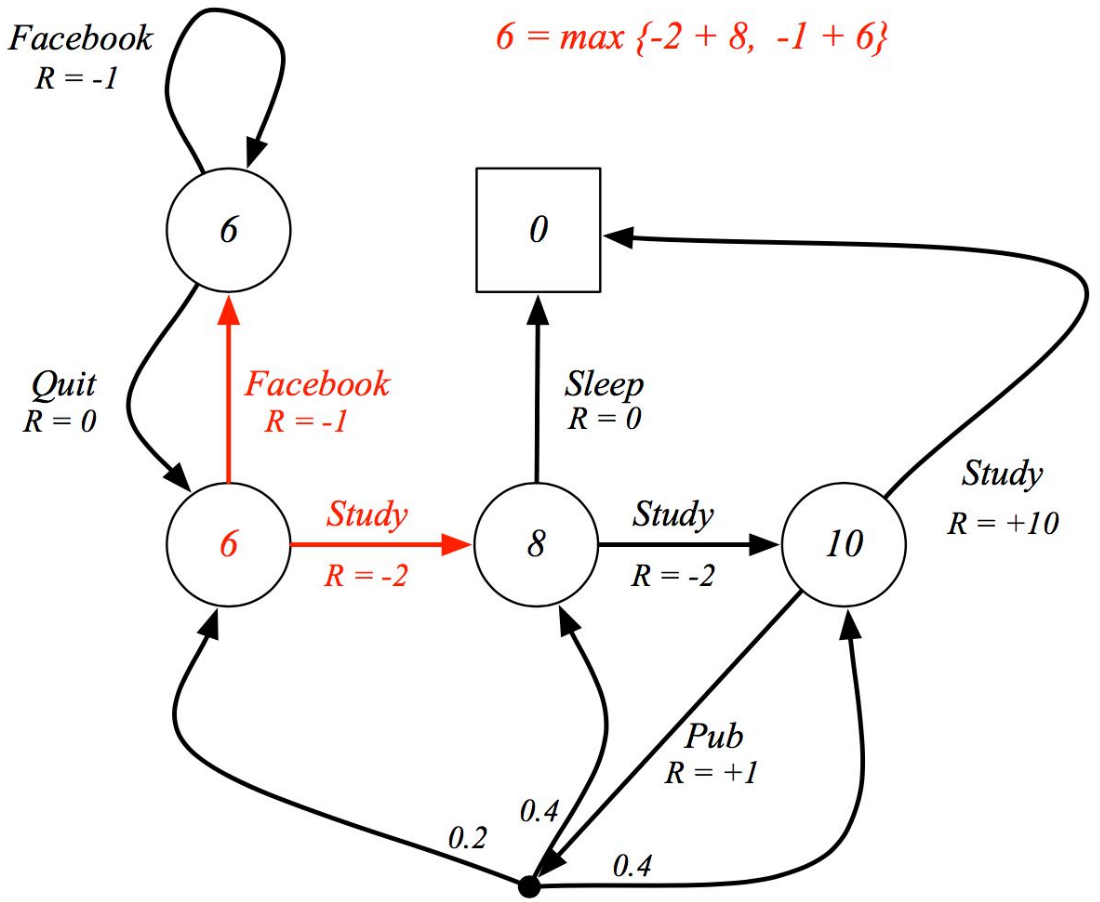
$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

Bellman Optimality Eq, Q



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

Student MDP: Bellman Optimality



Solving Bellman Equations in MDP

Not easy

- Not a linear equation
- No "closed-form" solutions

Overview

MDPs — States, Transitions, Actions, Rewards

Prediction — Given Policy π, Estimate State Value Functions, Action Value Functions

Control — Estimate Optimal Value Functions, Optimal Policy

Does the agent know the MDP?

Yes! It's "planning"
Agent knows everything

No! It's "Model-free RL"
Agent observes everything as it goes

Overview

	Evaluate Policy, π	Find Best Policy, π*
MDP Known	Policy Evaluation	Policy/Value Iteration
MDP Unknown	MC and TD Learning	Q-Learning

Overview

	Evaluate Policy, π	Find Best Policy, π*
Planning MDP Known	Policy Evaluation	Policy/Value Iteration
MDP Unknown	MC and TD Learning	Q-Learning

Dynamic Programming

Dynamic sequential or temporal component to the problem

Programming optimising a “program”, i.e. a policy

- c.f. linear programming

- A method for solving complex problems
- By breaking them down into subproblems
  - Solve the subproblems
  - Combine solutions to subproblems

Requirements for Dynamic Programming

Dynamic Programming is a very general solution method for problems which have two properties:

Requirements for Dynamic Programming

Dynamic Programming is a very general solution method for problems which have two properties:

- Optimal substructure
  - *Principle of optimality* applies
  - Optimal solution can be decomposed into subproblems

Planning by Dynamic Programming

- Dynamic programming assumes full knowledge of the MDP
- It is used for *planning* in an MDP
- For prediction:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π
  - or: MRP $\langle \mathcal{S}, \mathcal{P}^\pi, \mathcal{R}^\pi, \gamma \rangle$
  - Output: value function v_π
- Or for control:
  - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
  - Output: optimal value function v_*
  - and: optimal policy π_*

Applications of DPs

Dynamic programming is used to solve many other problems, e.g.

- Scheduling algorithms
- String algorithms (e.g. sequence alignment)
- Graph algorithms (e.g. shortest path algorithms)
- Graphical models (e.g. Viterbi algorithm)
- Bioinformatics (e.g. lattice models)

Overview

	Evaluate Policy, π	Find Best Policy, π*
MDP Known	Policy Evaluation	Policy/Value Iteration
MDP Unknown	MC and TD Learning	Sarsa + Q-Learning

Iterative Policy Evaluation

- Problem: evaluate a given policy π
- Solution: iterative application of Bellman expectation backup

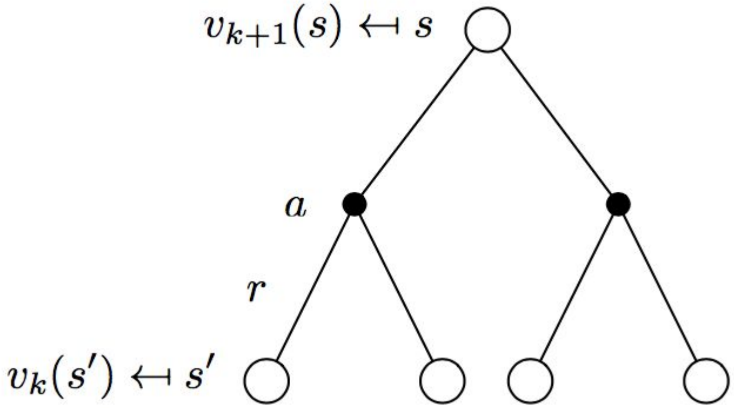
Iterative Policy Evaluation

- Problem: evaluate a given policy π
- Solution: iterative application of Bellman expectation backup
- $v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_\pi$

Iterative Policy Evaluation

- Problem: evaluate a given policy π
- Solution: iterative application of Bellman expectation backup
- $v_1 \to v_2 \to ... \to v_\pi$
- Using *synchronous* backups,
  - At each iteration $k + 1$
  - For all states $s \in \mathcal{S}$
  - Update $v_{k+1}(s)$ from $v_k(s')$
  - where s' is a successor state of s

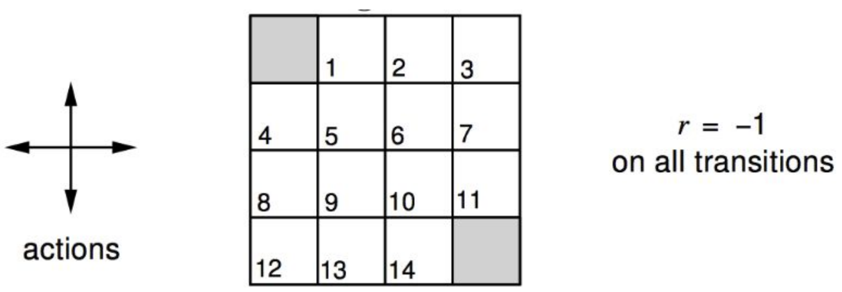
Iterative Policy Evaluation



$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}^{k+1} = \boldsymbol{\mathcal{R}}^{\boldsymbol{\pi}} + \gamma \boldsymbol{\mathcal{P}}^{\boldsymbol{\pi}} \mathbf{v}^k$$

Random Policy: Grid World



- Undiscounted episodic MDP ($\gamma = 1$)
- Nonterminal states $1, \ldots, 14$
- One terminal state (shown twice as 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

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

Policy Evaluation: Grid World

v_k for the Random Policy

$k = 0$

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

Time 0 : do nothing, stop; no cost.

$k = 1$

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

Time 1 : move (reward -1); then k=0

Unless in goal: reward 0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

Time 2 : move (reward -1); then k=1

Most: move (-1) + [v1 = -1] = -2

Some: move (-1) + ¾ [v1 = -1] + ¼ [v1=0] = 1.75

Policy Evaluation: Grid World

v_k for the Random Policy

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

Policy Evaluation: Grid World

v_k for the Random Policy

Greedy Policy w.r.t. v_k

$k = 0$

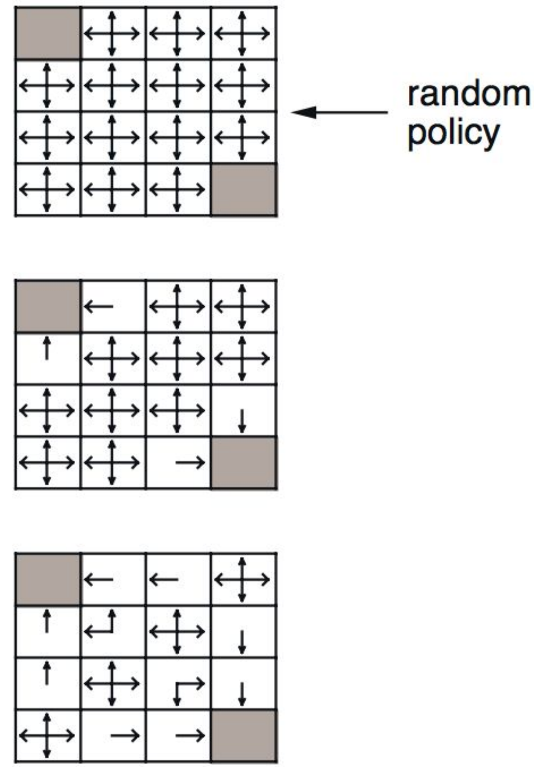
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

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	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

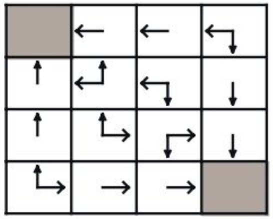


Policy Evaluation: Grid World

v_k for the Random Policy	Greedy Policy w.r.t. v_k

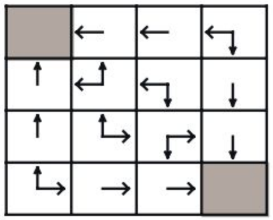
$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0



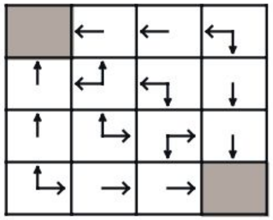
$k = 10$

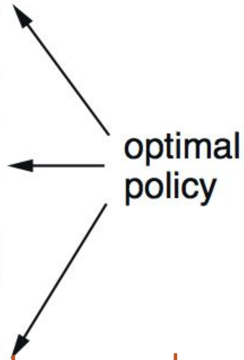
0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0



$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0





In general:
best policy & value for
"one step, then
follow random policy"
(always better policy than random!)

Overview

	Evaluate Policy, π	Find Best Policy, π*
MDP Known	Policy Evaluation	Policy/Value Iteration
MDP Unknown	MC and TD Learning	Sarsa + Q-Learning

Improving a Policy!

- Given a policy π
  - Evaluate the policy π

$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + ...|S_t = s\right]$$

Improving a Policy!

- Given a policy π
  - Evaluate the policy π

$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + ... | S_t = s\right]$$

  - Improve the policy by acting greedily with respect to v_π

$$\pi' = \text{greedy}(v_\pi)$$

Improving a Policy!

- Given a policy π
  - Evaluate the policy π

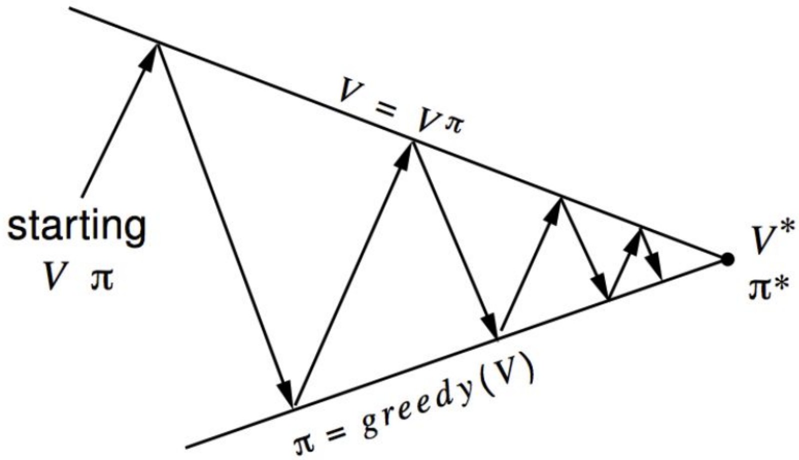
$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + ...|S_t = s\right]$$

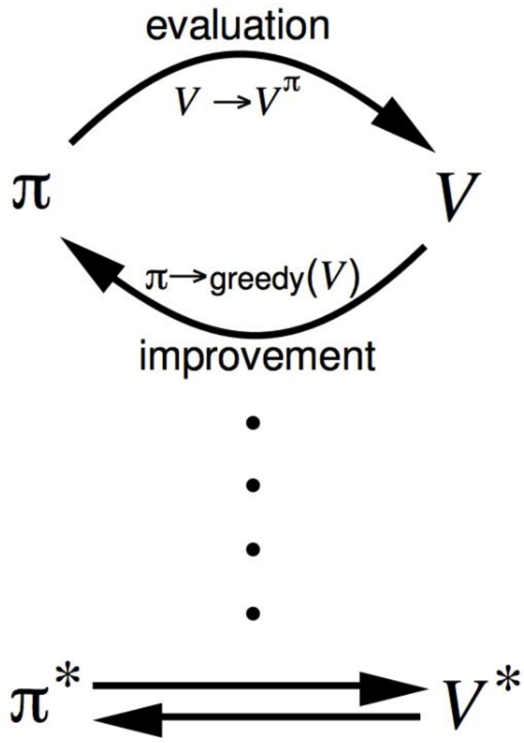
  - Improve the policy by acting greedily with respect to v_π

$$\pi' = \text{greedy}(v_\pi)$$

- 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 $\pi*$

Policy Iteration





Policy evaluation Estimate v_π

Iterative policy evaluation

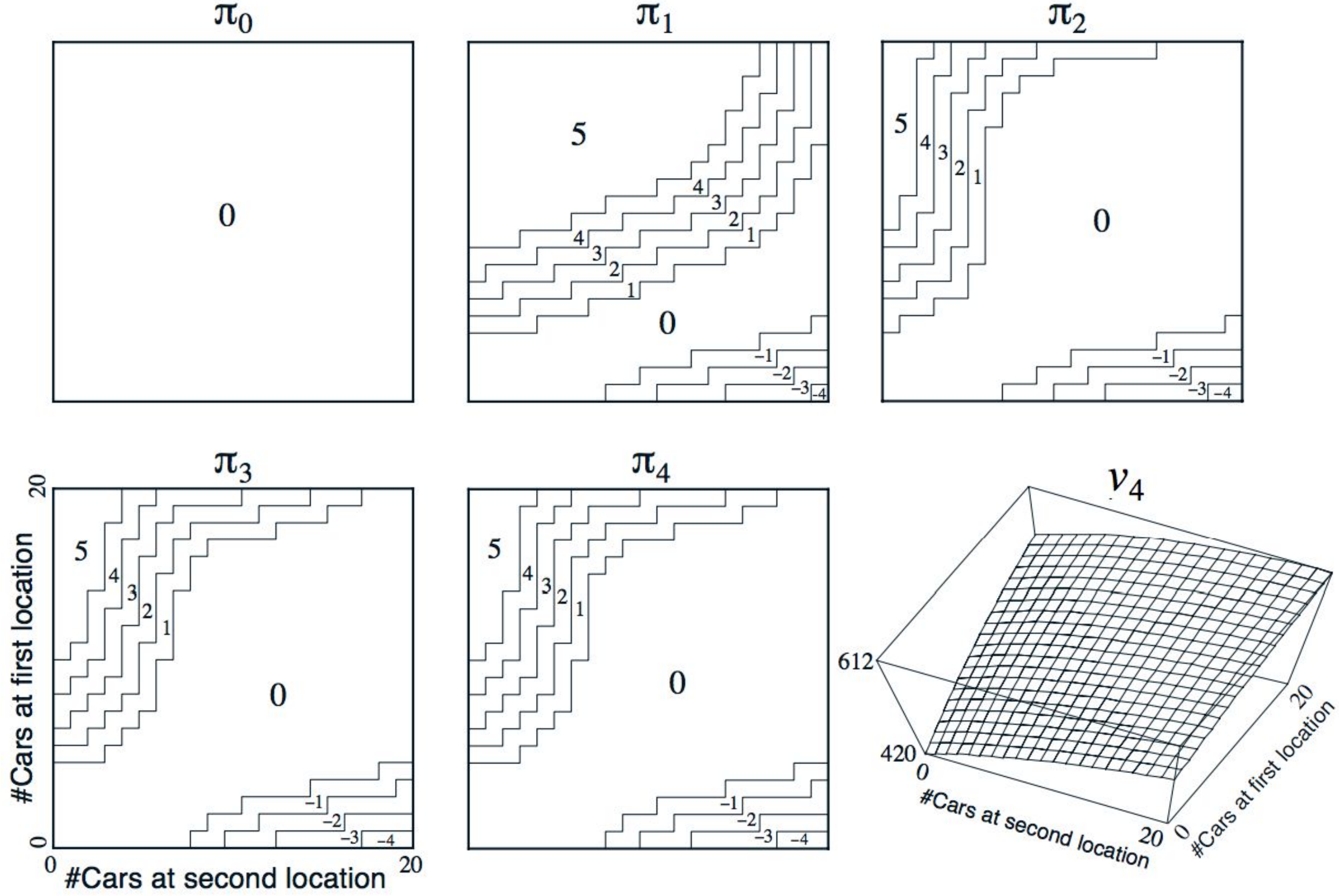
Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement

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 prob $\frac{\lambda^n}{n!}e^{-\lambda}$
  - 1st location: average requests = 3, average returns = 3
  - 2nd location: average requests = 4, average returns = 2

Policy Iteration in Car Rental



Policy Improvement

- If improvements stop,

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

Policy Improvement

- If improvements stop,

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

- Then the Bellman optimality equation has been satisfied

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

Policy Improvement

- If improvements stop,

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

- Then the Bellman optimality equation has been satisfied

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

- Therefore $v_\pi(s) = v_*(s)$ for all $s \in \mathcal{S}$
- so π is an optimal policy

Modified Policy Iteration

- Does policy evaluation need to converge to v_π?

Modified Policy Iteration

- Does policy evaluation need to converge to v_π?
- Or should we introduce a stopping condition
  - e.g. ϵ-convergence of value function

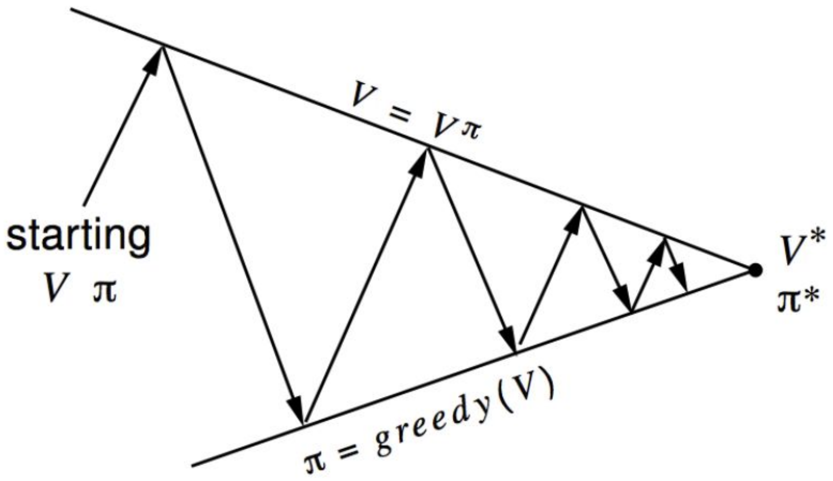
Modified Policy Iteration

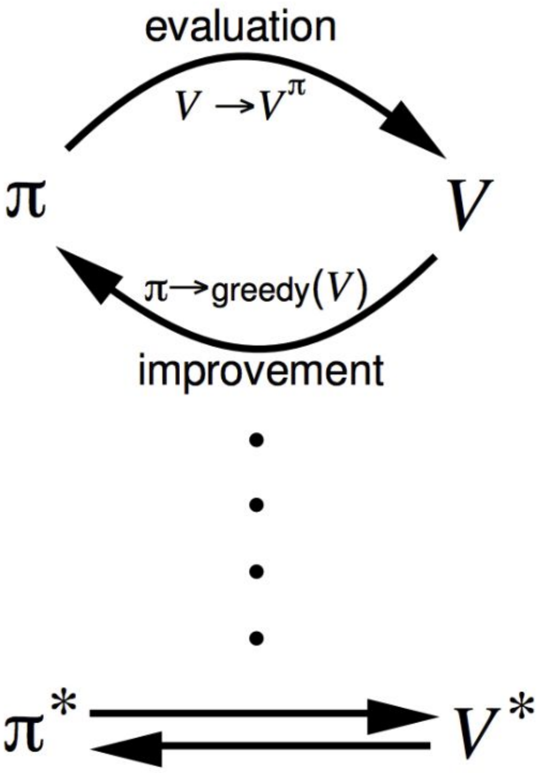
- Does policy evaluation need to converge to v_π?
- Or should we introduce a stopping condition
  - e.g. ϵ-convergence of value function
- Or simply stop after k iterations of iterative policy evaluation?
- For example, in the small gridworld $k = 3$ was sufficient to achieve optimal policy

Modified Policy Iteration

- Does policy evaluation need to converge to v_π?
- Or should we introduce a stopping condition
  - e.g. ϵ-convergence of value function
- Or simply stop after k iterations of iterative policy evaluation?
- For example, in the small gridworld $k = 3$ was sufficient to achieve optimal policy
- Why not update policy every iteration? i.e. stop after $k = 1$
  - This is equivalent to *value iteration* (next section)

Generalized Policy Iteration





Policy evaluation Estimate v_π

Any policy evaluation algorithm

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

Any policy improvement algorithm

Overview

	Evaluate Policy, π	Find Best Policy, π*
MDP Known	Policy Evaluation	Policy/Value Iteration
MDP Unknown	MC and TD Learning	Sarsa + Q-Learning

Monte Carlo RL

- MC methods learn directly from episodes of experience
- MC is *model-free*: no knowledge of MDP transitions / rewards

Monte Carlo RL

- MC methods learn directly from episodes of experience
- MC is *model-free*: no knowledge of MDP transitions / rewards
- MC learns from *complete* episodes: no bootstrapping
- MC uses the simplest possible idea: value = mean return

Monte Carlo RL

- MC methods learn directly from episodes of experience
- MC is *model-free*: no knowledge of MDP transitions / rewards
- MC learns from *complete* episodes: no bootstrapping
- MC uses the simplest possible idea: value = mean return
- Caveat: can only apply MC to *episodic* MDPs
  - All episodes must terminate

Monte Carlo Policy Evaluation

- Goal: learn v_π from episodes of experience under policy π

$$S_1, A_1, R_2, ..., S_k \sim \pi$$

- Recall that the *return* is the total discounted reward:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... + \gamma^{T-1} R_T$$

- Recall that the value function is the expected return:

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

- Monte-Carlo policy evaluation uses *empirical mean* return instead of *expected* return

Every-Visit MC Policy Evaluation

- To evaluate state s
- Every time-step t that state s is visited in an episode,
- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- Again, $V(s) \to v_\pi(s)$ as $N(s) \to \infty$

Equivalent, "incremental tracking" form:

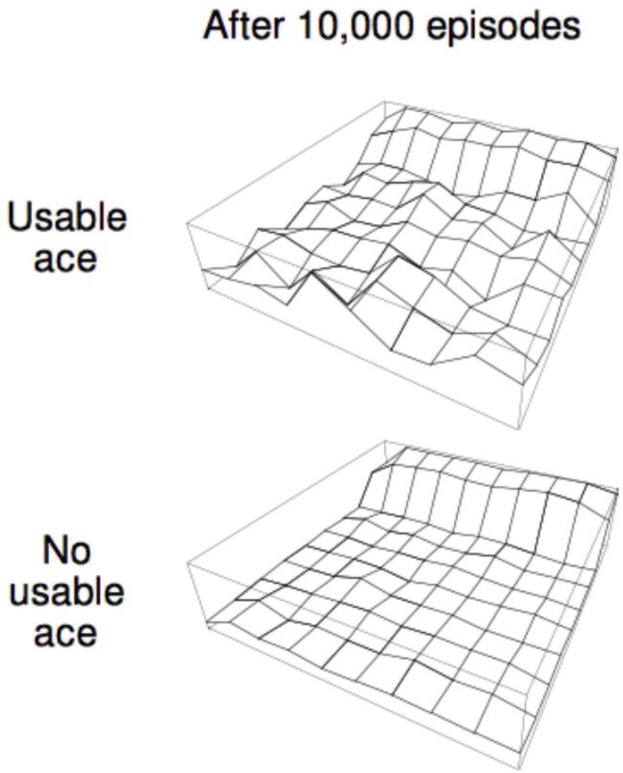
$$V(s) \leftarrow V(s) + \frac{1}{N(s)}\left(G_t - V(s)\right)$$

Looks like SGD to minimize MSE from the mean value…

Blackjack Example

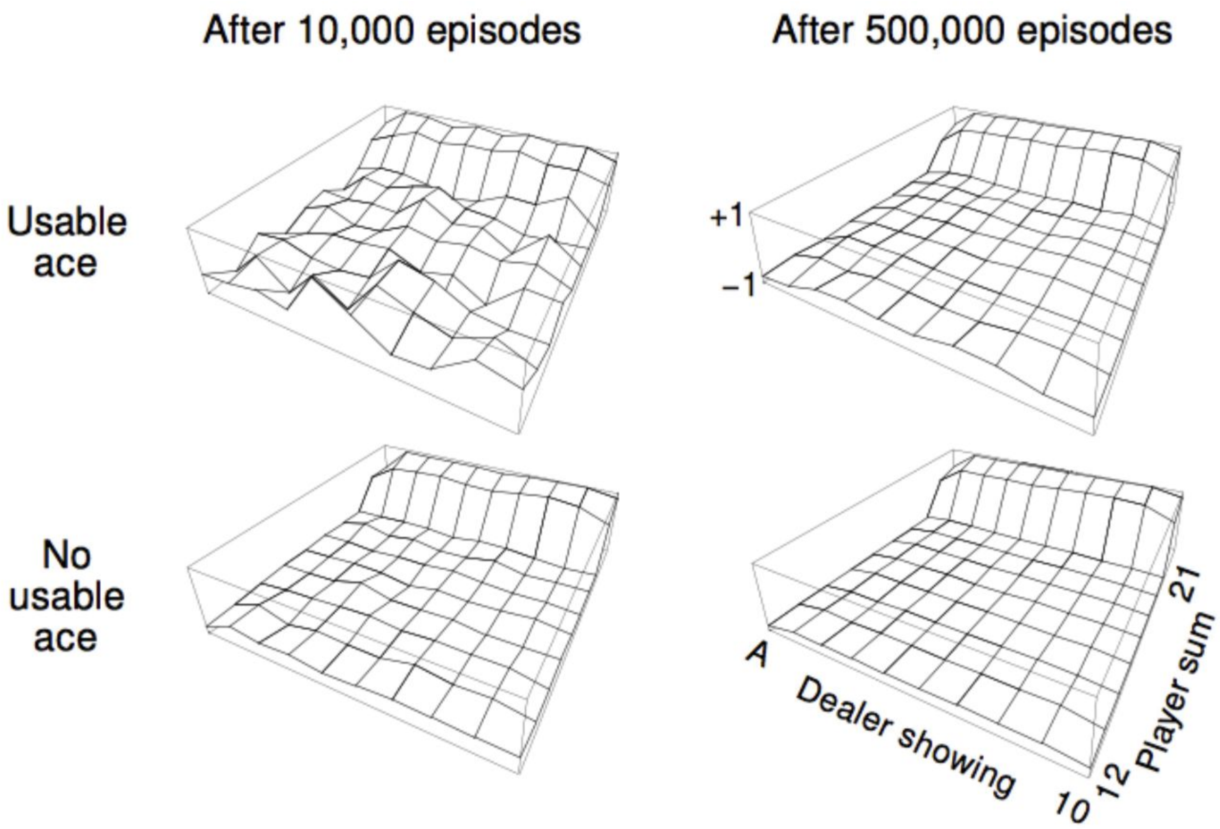
- States (200 of them):
  - Current sum (12-21)
  - Dealer's showing card (ace-10)
  - Do I have a "useable" ace? (yes-no)
- Action stand Stop receiving cards (and terminate)
- Action hit : Take another card (no replacement)
- Reward for stand
  - +1 if sum of cards > sum of dealer cards
  - 0 if sum of cards = sum of dealer cards
  - -1 if sum of cards < sum of dealer cards
- Reward for hit :
  - -1 if sum of cards > 21 (and terminate)
  - 0 otherwise
- Transitions: automatically hit if sum of cards < 12

Blackjack Value Function



Policy: stand if sum of cards ≥ 20, otherwise hit

Blackjack Value Function



Policy: stand if sum of cards ≥ 20, otherwise hit

Temporal Difference Learning

- TD methods learn directly from episodes of experience
- TD is *model-free*: no knowledge of MDP transitions / rewards

Temporal Difference Learning

- TD methods learn directly from episodes of experience
- TD is *model-free*: no knowledge of MDP transitions / rewards
- TD learns from *incomplete* episodes, by *bootstrapping*
- TD updates a guess towards a guess

MC and TD

- Goal: learn v_π online from experience under policy π
- Incremental every-visit Monte-Carlo
  - Update value $V(S_t)$ toward *actual* return G_t

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t) \right)$$

MC and TD

- Goal: learn v_π online from experience under policy π
- Incremental every-visit Monte-Carlo
  - Update value $V(S_t)$ toward *actual* return G_t

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t) \right)$$

- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $V(S_t)$ toward *estimated* return $R_{t+1} + \gamma V(S_{t+1})$

MC and TD

- Goal: learn v_π online from experience under policy π
- Incremental every-visit Monte-Carlo
  - Update value $V(S_t)$ toward *actual* return G_t

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$

- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $V(S_t)$ toward *estimated* return $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$

MC and TD

- Goal: learn v_π online from experience under policy π
- Incremental every-visit Monte-Carlo
  - Update value $V(S_t)$ toward *actual* return G_t

$$V(S_t) \leftarrow V(S_t) + \alpha \left(G_t - V(S_t) \right)$$

- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $V(S_t)$ toward *estimated* return $R_{t+1} + \gamma V(S_{t+1})$

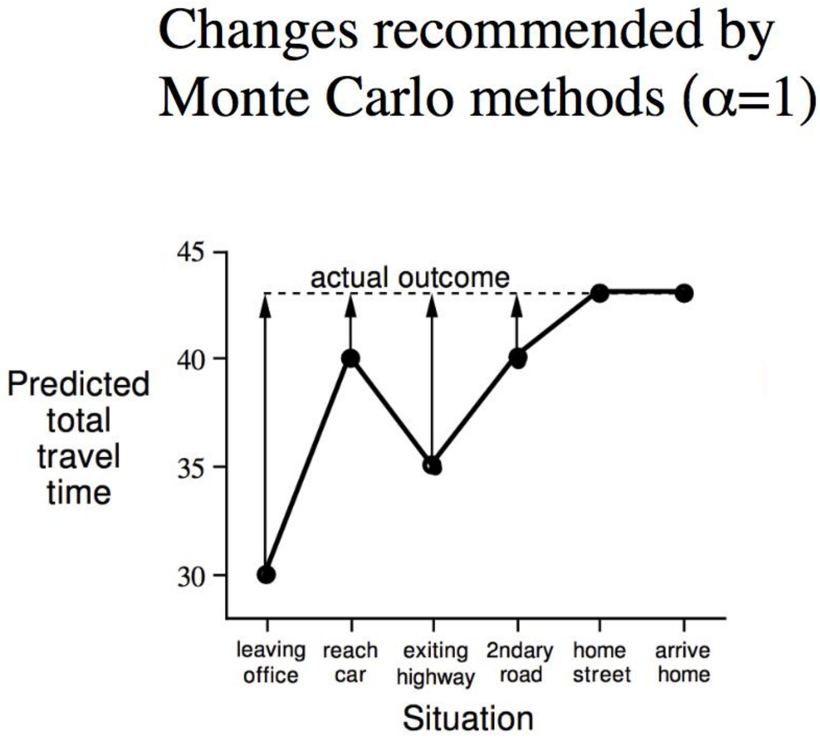
$$V(S_t) \leftarrow V(S_t) + \alpha \left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right)$$

  - $R_{t+1} + \gamma V(S_{t+1})$ is called the *TD target*
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the *TD error*

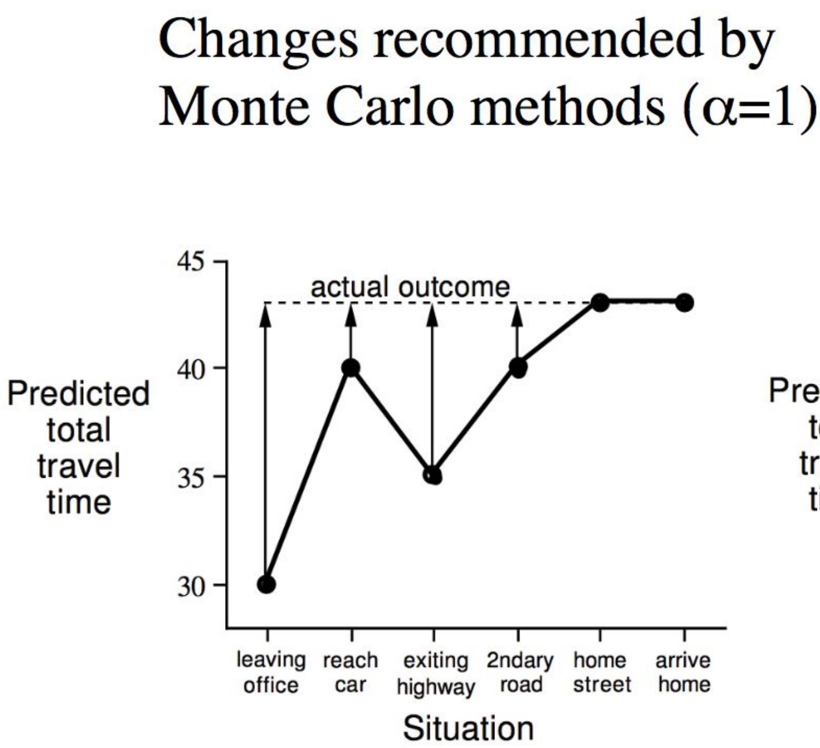
Driving Home Example

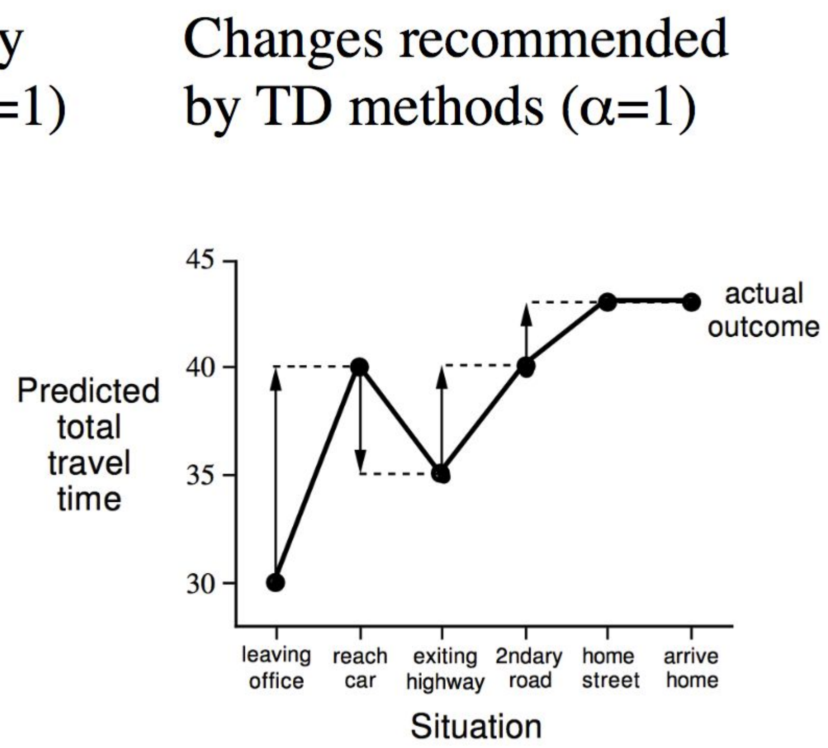
State	Elapsed Time (minutes)	Predicted Time to Go	Predicted Total Time
leaving office	0	30	30
reach car, raining	5	35	40
exit highway	20	15	35
behind truck	30	10	40
home street	40	3	43
arrive home	43	0	43

Driving Home: MC vs TD



Driving Home: MC vs TD





Finite Episodes: AB Example

Two states A, B; no discounting; 8 episodes of experience

A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0

What is $V(A), V(B)$?

MC & TD can give different answers on fixed data:

V(B) = 6 / 8

V(A) = 0 ?   (Direct MC estimate)

V(A) = 6 / 8?   (TD estimate)

MC vs TD

Monte Carlo	Temporal Difference

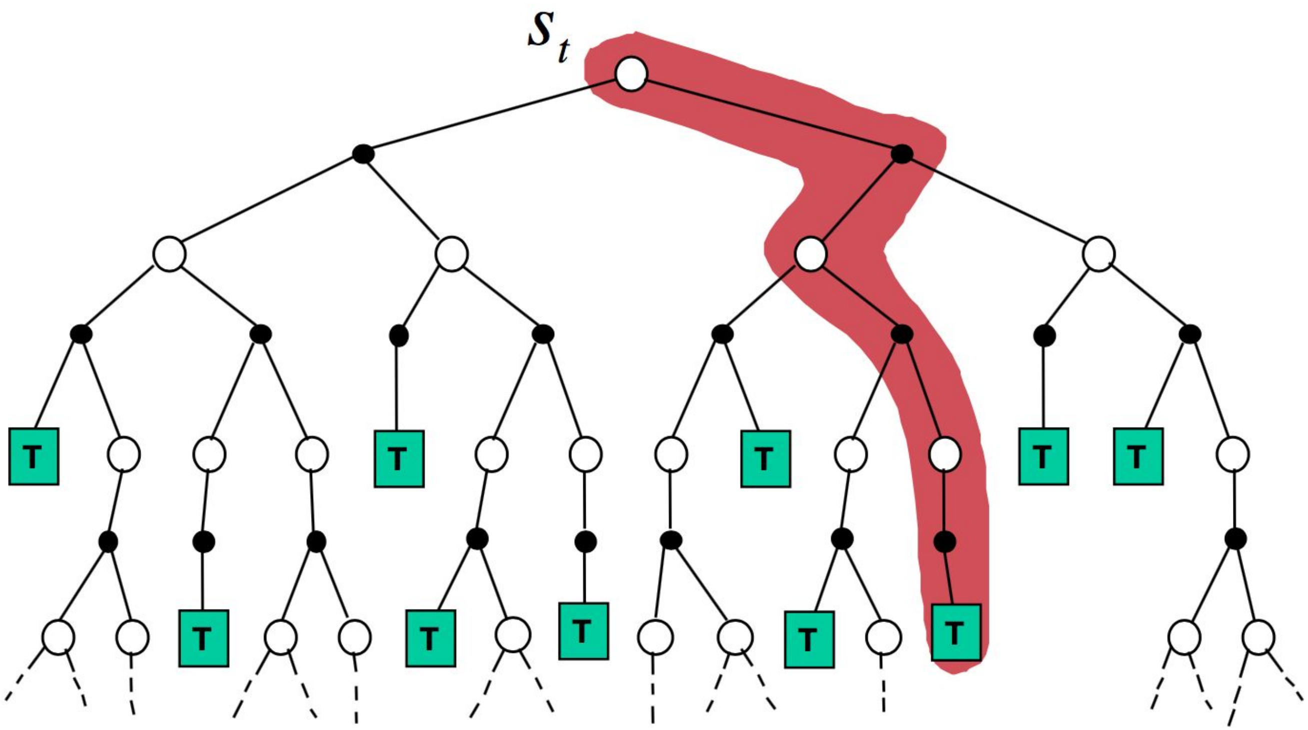
• Wait till end of episode to learn
  • Only for *terminating* worlds
	• Learn online after every step
	  • Non-*terminating* worlds ok

• High-variance, low bias
  • Not sensitive to initial value
  • Good convergence properties
	• Low variance, high bias
	  • Sensitive to initial value
	  • Much more efficient

• Doesn’t exploit Markov property	• Exploits Markov Property
• Minimizes squared error	• Maximizes log-likelihood

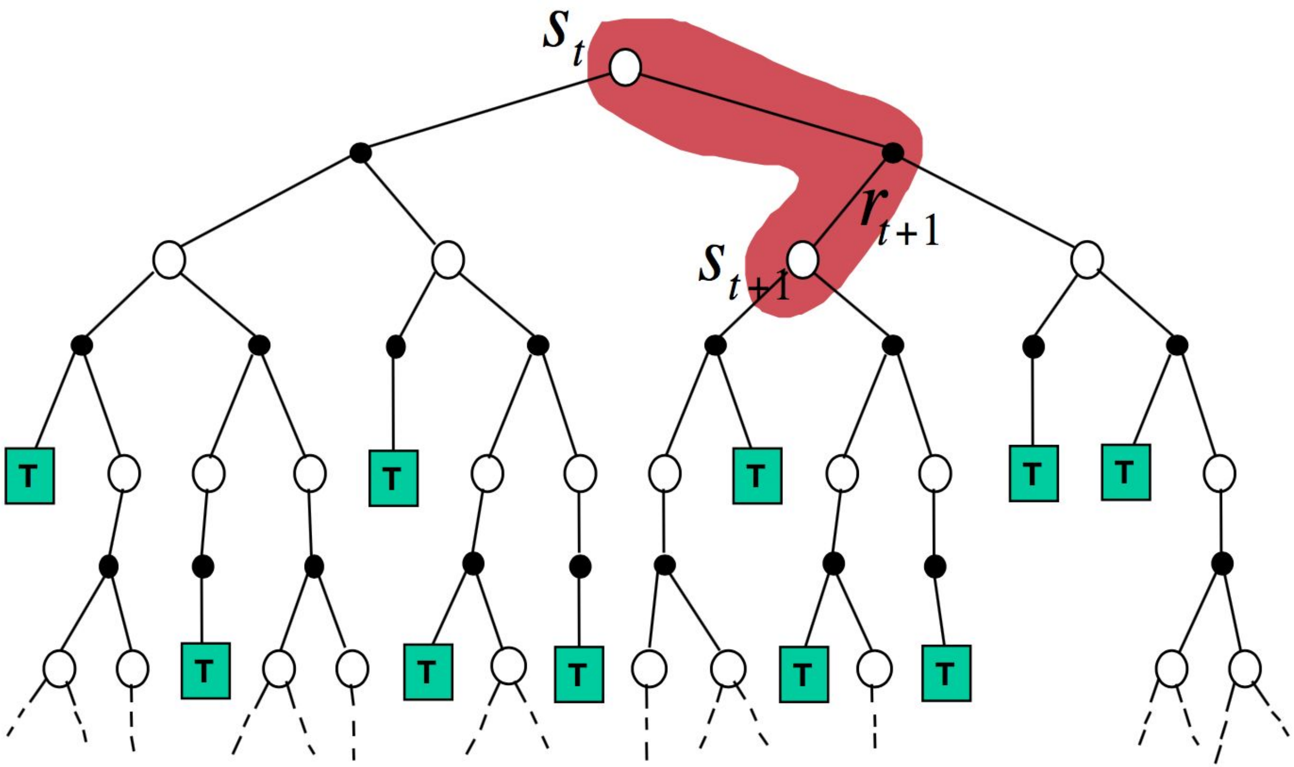
Unified View: Monte Carlo

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$



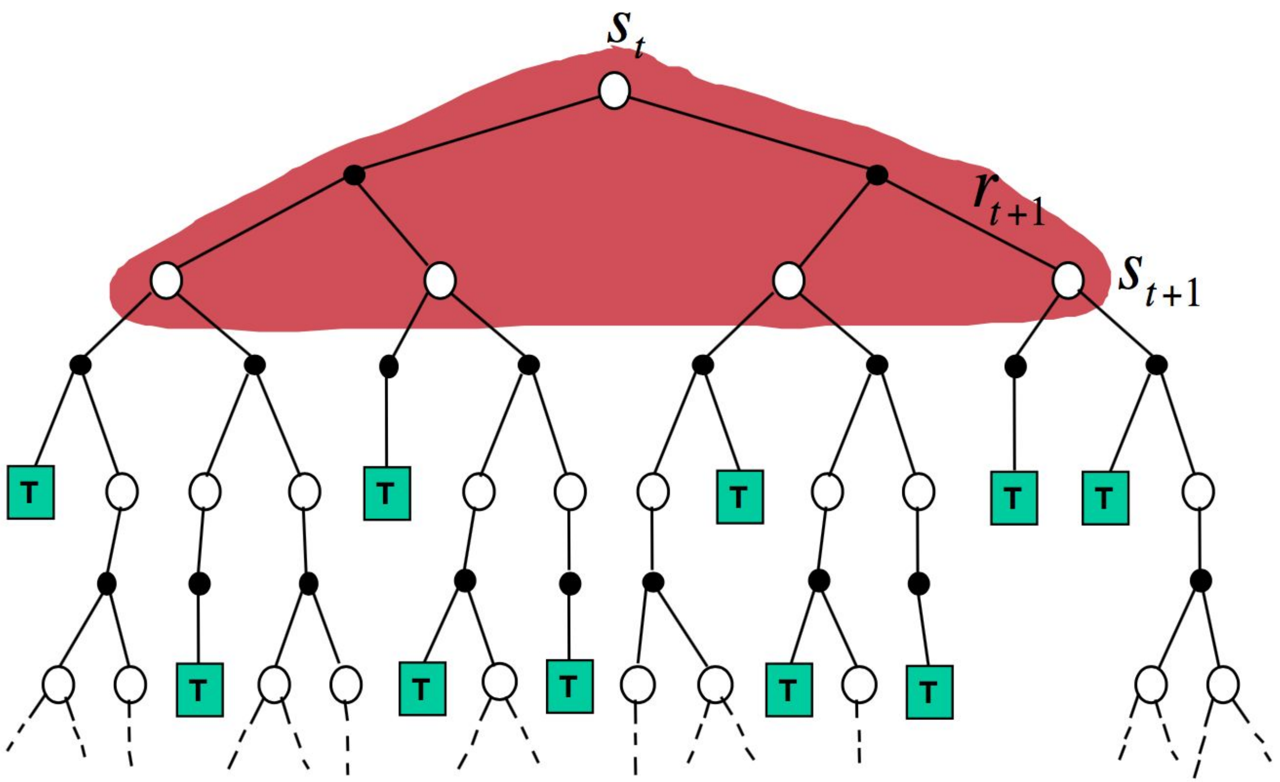
Unified View: TD Learning

$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$

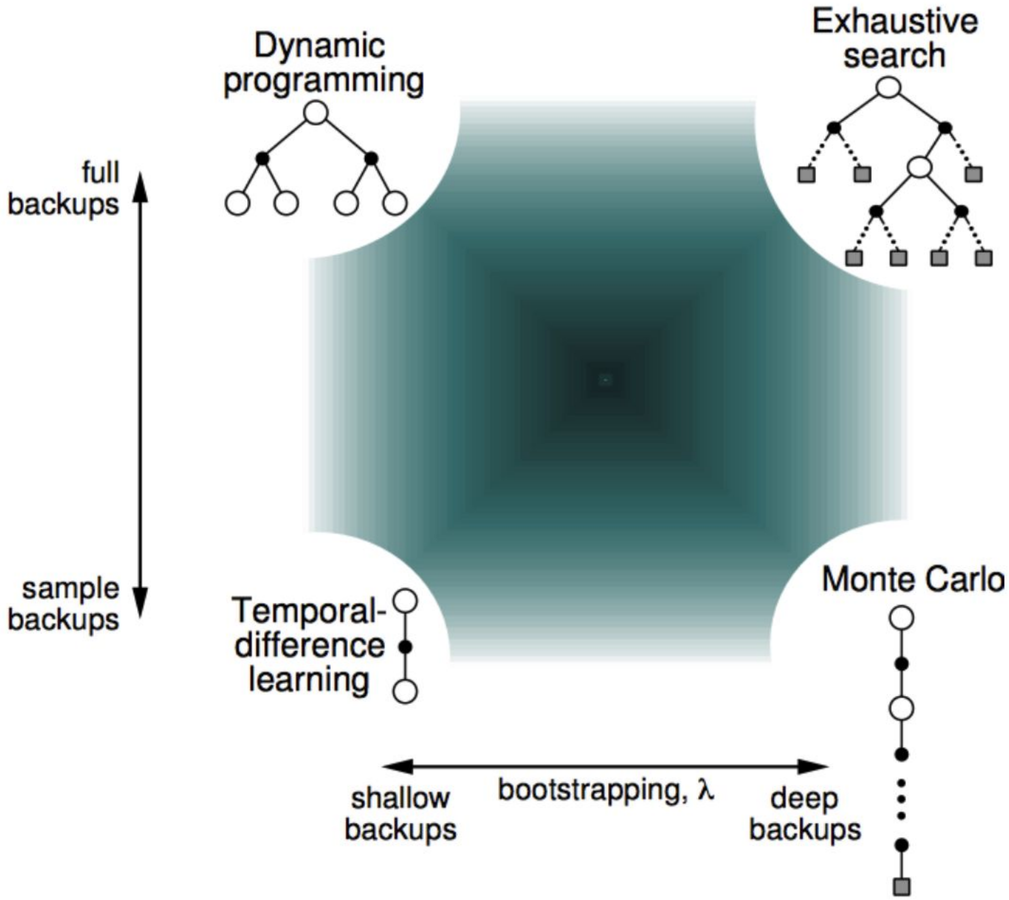


Unified View: Dynamic Prog.

$$V(S_t) \leftarrow \mathbb{E}_\pi \left[R_{t+1} + \gamma V(S_{t+1})\right]$$



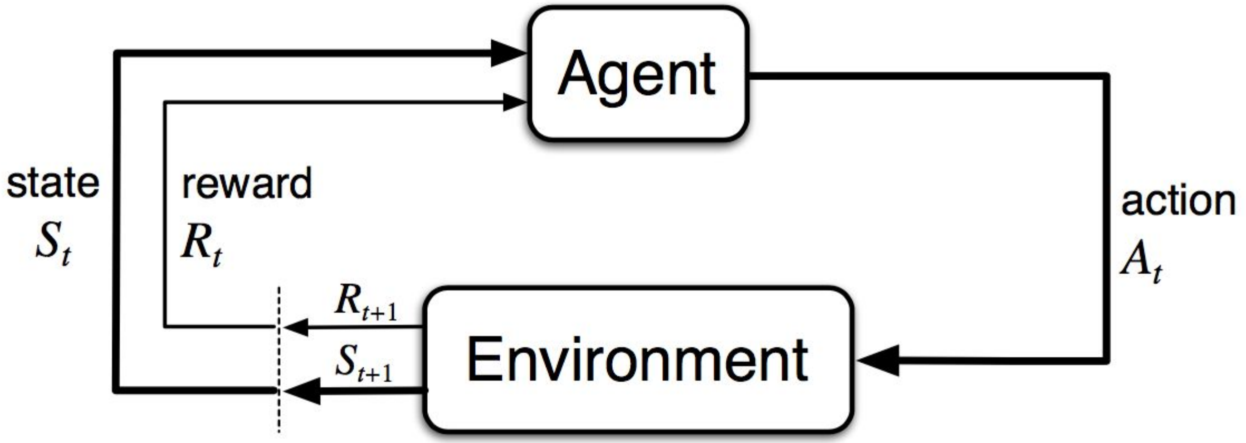
Unified View of RL (Prediction)



Overview

	Evaluate Policy, π	Find Best Policy, π*
MDP Known	Policy Evaluation	Policy/Value Iteration
MDP Unknown	MC and TD Learning	Sarsa + Q-Learning

Model-free Control

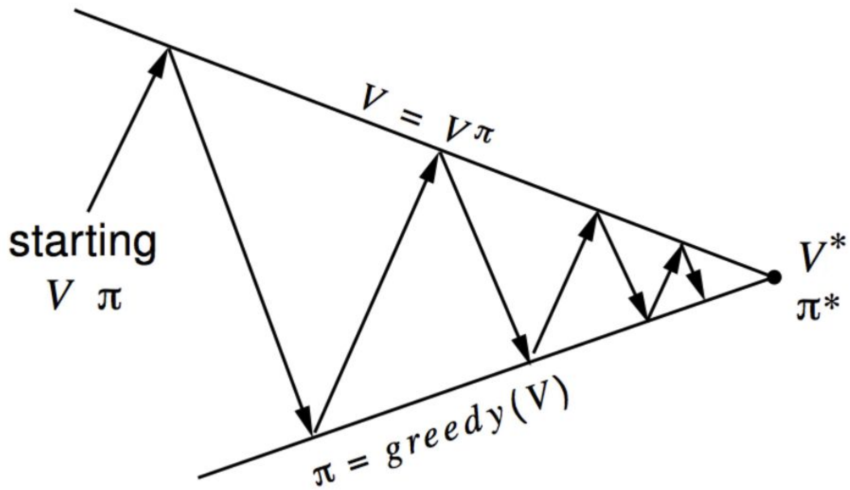


Learn a policy π to maximize rewards in the environment

On and Off Policy Learning

- On-policy learning
  - "Learn on the job"
  - Learn about policy π from experience sampled from π
- Off-policy learning
  - "Look over someone's shoulder"
  - Learn about policy π from experience sampled from μ

Generalized Policy Iteration

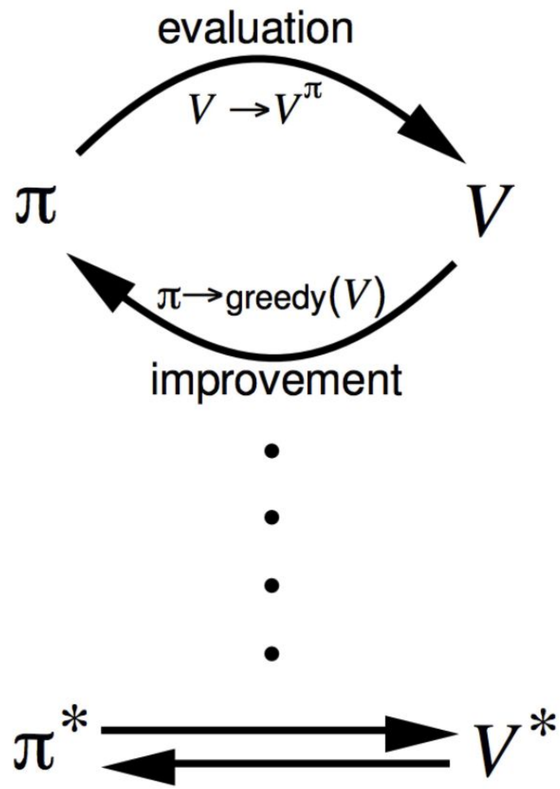


Policy evaluation Estimate v_π

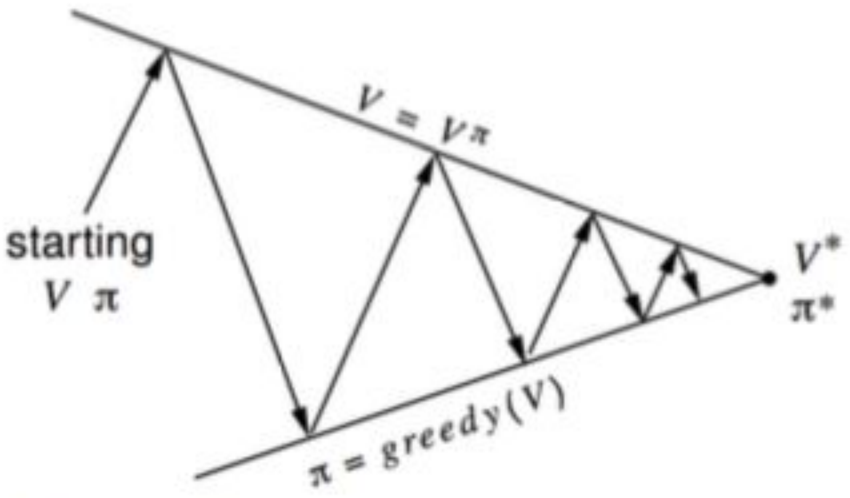
e.g. Iterative policy evaluation

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

e.g. Greedy policy improvement



Gen Policy Improvement?



Policy evaluation Monte-Carlo policy evaluation, $V = v_\pi$?

Policy improvement Greedy policy improvement?

Not quite!

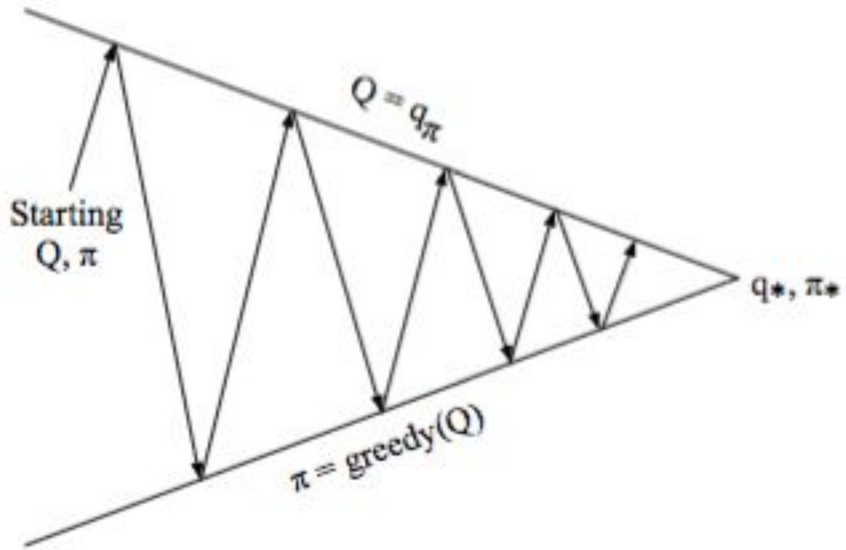
- Greedy policy improvement over $V(s)$ requires model of MDP

$$\pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}}\, \mathcal{R}_s^a + \mathcal{P}_{ss'}^a V(s')$$

- Greedy policy improvement over $Q(s, a)$ is model-free

$$\pi'(s) = \underset{a \in \mathcal{A}}{\text{argmax}}\, Q(s, a)$$

Learn Q function directly…



Policy evaluation Monte-Carlo policy evaluation, $Q = q_\pi$

Policy improvement Greedy policy improvement?

Greedy Action Selection?

- There are two doors in front of you.
- You open the left door and get reward 0
  $V(left) = 0$
- You open the right door and get reward $+1$
  $V(right) = +1$

Greedy Action Selection?

- There are two doors in front of you.
- You open the left door and get reward 0
  $V(left) = 0$
- You open the right door and get reward +1
  $V(right) = +1$
- You open the right door and get reward +3
  $V(right) = +2$
- You open the right door and get reward +2
  $V(right) = +2$

Greedy Action Selection?

- There are two doors in front of you.
- You open the left door and get reward 0
  $V(left) = 0$
- You open the right door and get reward +1
  $V(right) = +1$
- You open the right door and get reward +3
  $V(right) = +2$
- You open the right door and get reward +2
  $V(right) = +2$

$$\vdots$$

- Are you sure you've chosen the best door?

∈-Greedy Exploration

- Simplest idea for ensuring continual exploration
- All *m* actions are tried with non-zero probability

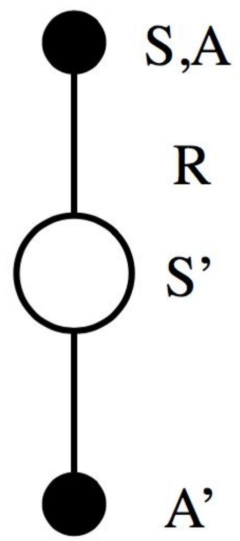
∈-Greedy Exploration

- Simplest idea for ensuring continual exploration
- All m actions are tried with non-zero probability
- With probability $1 - \epsilon$ choose the greedy action
- With probability ϵ choose an action at random

Which Policy Evaluation?

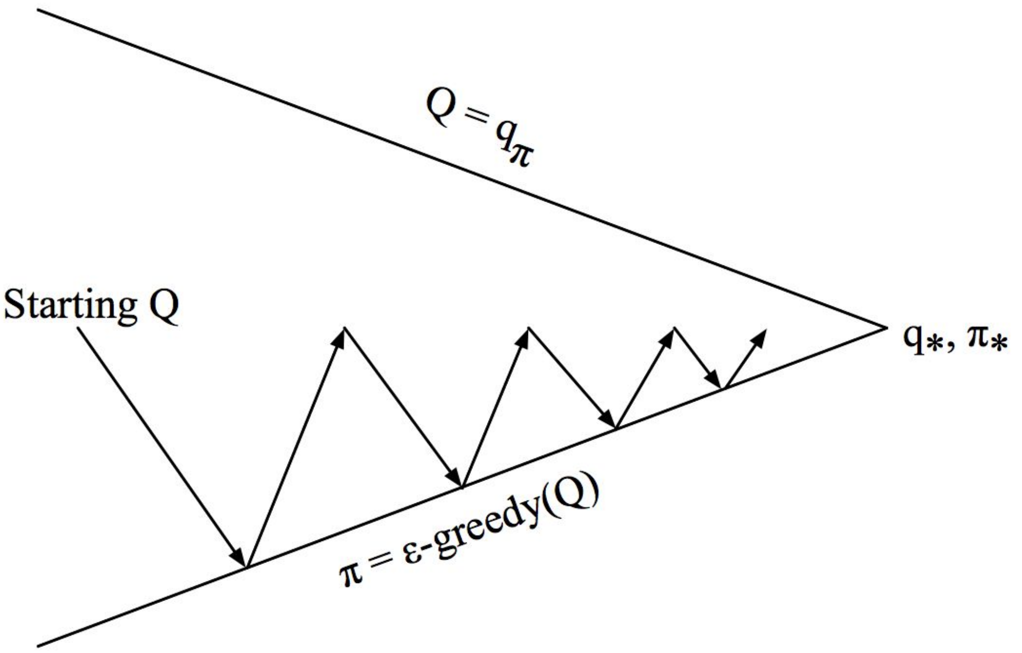
- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Online
  - Incomplete sequences
- Natural idea: use TD
  - Apply TD to $Q(S, A)$
  - Use ϵ-greedy policy improvement
  - Update every time-step

Sarsa: TD for Policy Evaluation



$$Q(S, A) \leftarrow Q(S, A) + \alpha \left(R + \gamma Q(S', A') - Q(S, A) \right)$$

On-Policy Control w/ Sarsa



Every time-step:

Policy evaluation Sarsa, $Q \approx q_\pi$

Policy improvement ϵ-greedy policy improvement

Off-Policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s,a)$
- While following behaviour policy $\mu(a|s)$

$$\{S_1, A_1, R_2, ..., S_T\} \sim \mu$$

Off-Policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s, a)$
- While following behaviour policy $\mu(a|s)$

$$\{S_1, A_1, R_2, ..., S_T\} \sim \mu$$

- Why is this important?
- Learn from observing humans or other agents
- Re-use experience generated from old policies $\pi_1, \pi_2, ..., \pi_{t-1}$
- Learn about *optimal* policy while following *exploratory* policy
- Learn about *multiple* policies while following *one* policy

Q-Learning

- We now consider off-policy learning of action-values $Q(s, a)$

Q-Learning

- We now consider off-policy learning of action-values $Q(s, a)$
- Next action is chosen using behaviour policy $A_{t+1} \sim \mu(\cdot|S_t)$
- But we consider alternative successor action $A' \sim \pi(\cdot|S_t)$

Q-Learning

- We now consider off-policy learning of action-values $Q(s, a)$

- Next action is chosen using behaviour policy $A_{t+1} \sim \mu(\cdot|S_t)$
- But we consider alternative successor action $A' \sim \pi(\cdot|S_t)$
- And update $Q(S_t, A_t)$ towards value of alternative action

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t) \right)$$

Off-Policy w/ Q-Learning

- We now allow both behaviour and target policies to improve

Off-Policy w/ Q-Learning

- We now allow both behaviour and target policies to improve
- The target policy π is greedy w.r.t. $Q(s, a)$

$$\pi(S_{t+1}) = \underset{a'}{\operatorname{argmax}}\; Q(S_{t+1}, a')$$

- The behaviour policy μ is e.g. ϵ-greedy w.r.t. $Q(s, a)$

Off-Policy w/ Q-Learning

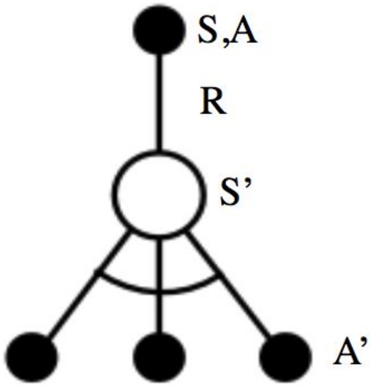
- We now allow both behaviour and target policies to improve
- The target policy π is greedy w.r.t. $Q(s, a)$

$$\pi(S_{t+1}) = \underset{a'}{\operatorname{argmax}}\ Q(S_{t+1}, a')$$

- The behaviour policy μ is e.g. ϵ-greedy w.r.t. $Q(s, a)$
- The Q-learning target then simplifies:

$$\begin{aligned} & R_{t+1} + \gamma Q(S_{t+1}, A') \\ =& R_{t+1} + \gamma Q(S_{t+1}, \underset{a'}{\operatorname{argmax}}\ Q(S_{t+1}, a')) \\ =& R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a') \end{aligned}$$

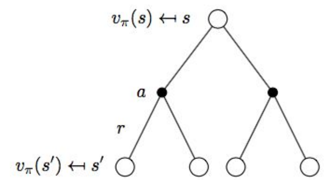
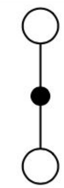
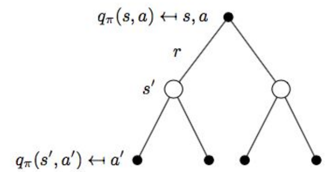
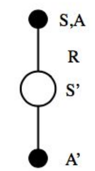
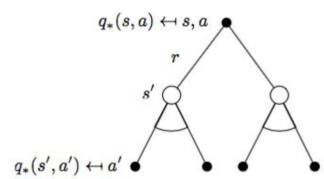
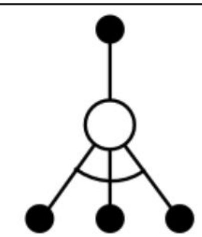
Q-Learning Control Algorithm



(SARSAMAX)

$$Q(S,A) \leftarrow Q(S,A) + \alpha \left(R + \gamma \max_{a'} Q(S', a') - Q(S,A) \right)$$

Relation between DP and TD

	Full Backup (DP)	*Sample Backup (TD)*
Bellman Expectation Equation for $v_\pi(s)$	 Iterative Policy Evaluation	 TD Learning
Bellman Expectation Equation for $q_\pi(s, a)$	 Q-Policy Iteration	 Sarsa
Bellman Optimality Equation for $q_*(s, a)$	 Q-Value Iteration	 Q-Learning

Update Eqns for DP and TD

Full Backup (DP)	*Sample Backup (TD)*
Iterative Policy Evaluation	TD Learning
$V(s) \leftarrow \mathbb{E}[R + \gamma V(S') \mid s]$	$V(S) \xleftarrow{\alpha} R + \gamma V(S')$
Q-Policy Iteration	Sarsa
$Q(s,a) \leftarrow \mathbb{E}[R + \gamma Q(S',A') \mid s,a]$	$Q(S,A) \xleftarrow{\alpha} R + \gamma Q(S',A')$
Q-Value Iteration	Q-Learning
$Q(s,a) \leftarrow \mathbb{E}\left[R + \gamma \max_{a' \in \mathcal{A}} Q(S',a') \mid s,a\right]$	$Q(S,A) \xleftarrow{\alpha} R + \gamma \max_{a' \in \mathcal{A}} Q(S',a')$

where $x \xleftarrow{\alpha} y \equiv x \leftarrow x + \alpha(y - x)$