CSCI567 Machine Learning (Spring 2018) Introduction to Reinforcement Learning

Michael Shindler

Lecture 25, April 23 2018

Outline

1. Administration
2. Reinforcement Learning

Acknowledgements

- Today's lecture is based on a lecture by Chao-Kai Chiang
- His lecture was given at USC last November
- Dr. Chiang's acknowledgements are the next slide:

Slides References

- The slides are **modified** or **inspired** from the following slides
  - Remi Munos.
    http://mlss11.bordeaux.inria.fr/docs/mlss11Bordeaux_MunosPart1.pdf
  - David Silver.
    http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
  - Satinder Singh.
    http://videolectures.net/site/normal_dl/tag=69270/mlss2010_singh_rlt.pdf
  - Richard Sutton.
    http://media.nips.cc/Conferences/2015/tutorialslides/SuttonIntroRL-nips-2015-tutorial.pdf

Outline

1. Administration
2. Reinforcement Learning

Administration

- Quiz 3 is on Friday.
  - Have a pencil for the Scantron portion
  - You may use writing tool of your choice for non-Scantron part.
  - Know your USC Student ID #
    Be sure to bubble in your ID # on Scantron
  - Know your **enrolled** discussion section
- Quiz 2 scores are posted on blackboard.
  - Average: 18.82
  - Median: 19.0
  - St Dev: 4.42

Outline

What is RL & What for

- What is **Reinforcement Learning** (RL)?
  - A **general purpose framework** mimicking **human's learning** **process**

- What for?
  - To **advance AI**

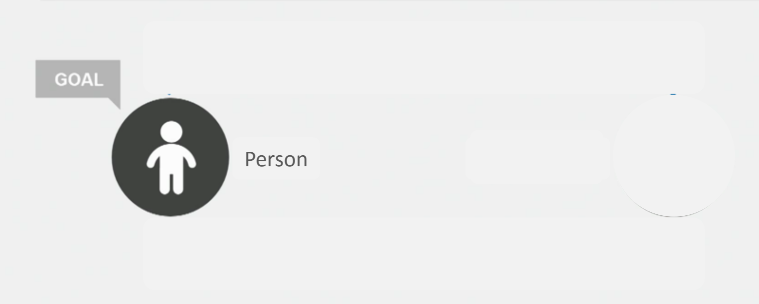
Human Learning

- **Human learning** as a **Sequential Decision Making** problem

  - E.g., financial investment, playing chess, etc.

Human Learning

- **Human learning** as a **Sequential Decision Making** problem



  – A **person** with a **goal**

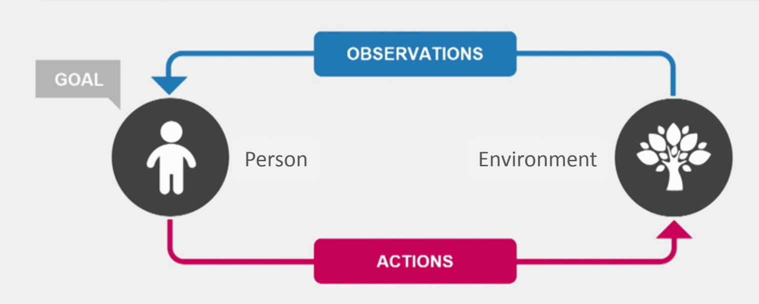
Human Learning

- **Human learning** as a **Sequential Decision Making** problem

  – A person with a goal **observing** an **environment**

Human Learning

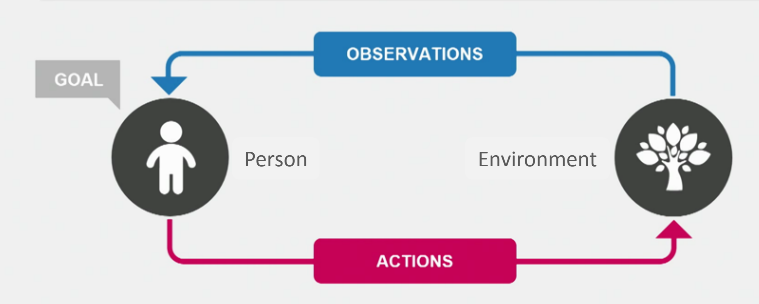
- **Human learning** as a **Sequential Decision Making** problem



  – Needs to decide an **action** to take ("**Decision**")
    - Action ⇒ affect environment ⇒ toward the goal

Human Learning

- **Human learning** as a **Sequential Decision Making** problem



  – Cycle repeated for multiple **iterations** until the goal is achieved ("**Sequential**")

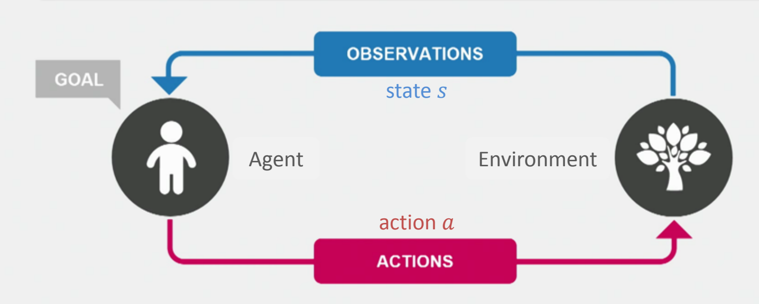
Reinforcement Learning

- **Reinforcement learning** as a **Sequential Decision Making** problem

  - An **agent** observing the **state s** of the **environment**

Reinforcement Learning

- **Reinforcement learning** as a **Sequential Decision Making** problem



  – The agent needs to decide an **action a** to take

Reinforcement Learning

- **Reinforcement learning** as a **Sequential Decision Making** problem

  - Feedback: **reward signal r** and **next state s'** of the environment

Reinforcement Learning

- **Reinforcement learning** as a **Sequential Decision Making** problem

  – **Goal**: Make **sequential decisions** to **maximize** the **total reward** $\sum_t r_t$ it gathers

Why Can RL Advance AI?

- Freedom to **learn** **interactively** with the environment

- Freedom to **switch/modify/improve** the way of **learning and acting** during its learning process

Characteristics of RL

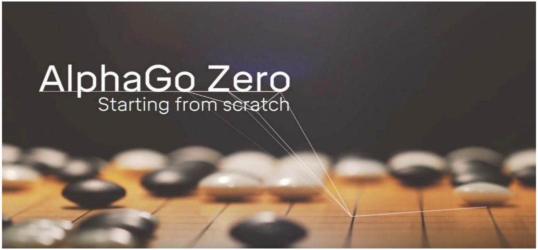
- **Uncertain** / changing **environment**
  - Need for trial & error
- **Actions** may have **long term consequences**
  - Affect the future states or data observed: data is no longer i.i.d.
  - Affect the future reward signals received
- **Reward** may be **delayed**
  - No supervision
- Require a **balance** between **immediate reward** and **long-term reward**
  - Need for exploration & exploitation

Characteristics of RL

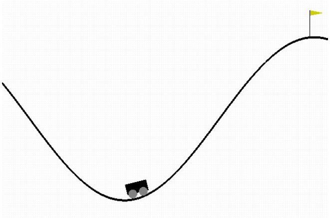
- **Uncertain** / changing **environment**
- **Actions** may have **long term consequences**
- **Reward** may be **delayed**
- Require a **balance** between **immediate reward** and **long-term reward**

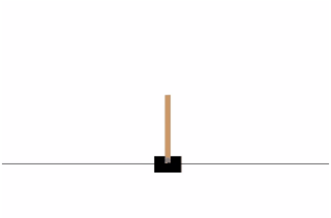
- E.g., Chess
  - current move affects future moves
  - rewarded when you are win at the end of the game
  - getting one piece now vs. winning in the end

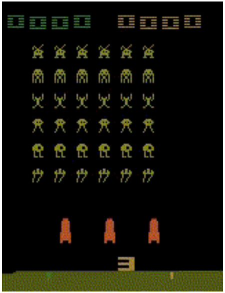
Applications of RL

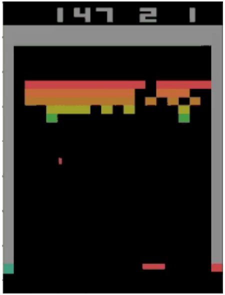


Applications of RL

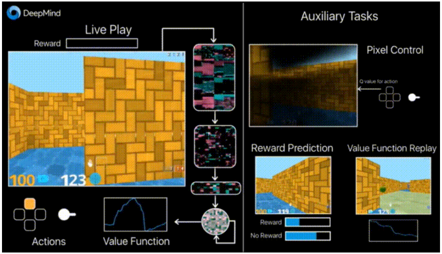


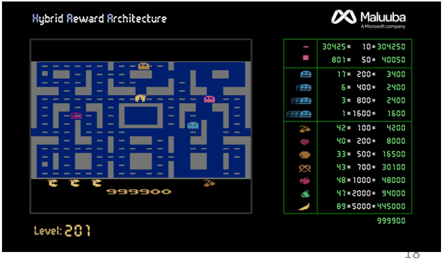




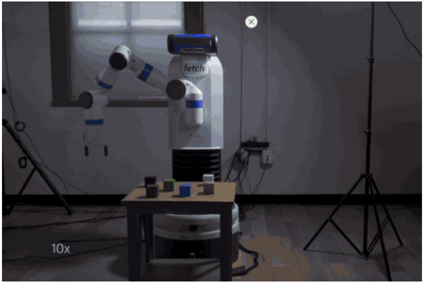


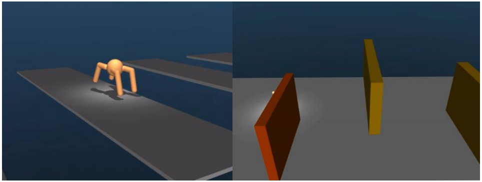
Applications of RL





Applications of RL





Applications of RL

- Playing **games**: Go, Chess, Atari, **poker**, …
- Explore worlds: Labyrinth, 3D worlds, …
- Continuous control: real-world, simulation
- **Recommendation system**
- Robotics
- **Operation research**: warehousing, transportation, scheduling
- **Adaptive treatment design**, **biological modeling**, …

REINFORCEMENT LEARNING

REINFORCEMENT LEARNING

Mathematical Formulation

Value Functions & Bellman Equations

Assumptions

Approaches

REINFORCEMENT LEARNING

Mathematical Formulation

Value Functions & Bellman Equations

Assumptions

Approaches

Modeling an Environment

- **Markov Decision Processes** (MDPs)

  $\langle \boldsymbol{S}, \boldsymbol{A}, \boldsymbol{P}, \boldsymbol{R}, \boldsymbol{\gamma} \rangle$
  - S : a finite **set of states**
  - A : a finite **set of actions**
  - P : a **state transition function**
    - $p(s', r|s, a)$: the probability of observing r and reaching s' after taking a at s.
  - R : a **reward function**
    - $\mathbb{E}[R_{t+1}|S_t = s, A_t = a]$
  - γ : a discount factor $\gamma \in [0,1]$
    - Role and function: later

Modeling an Environment

- **Markov Decision Processes** (MDPs)

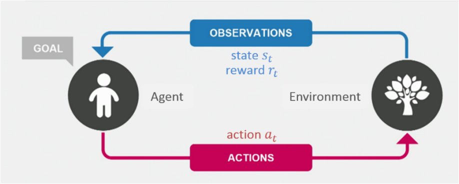
  $\langle \boldsymbol{S}, \boldsymbol{A}, \boldsymbol{P}, \boldsymbol{R}, \boldsymbol{\gamma} \rangle$

  – S : a finite **set of states**

  – A : a finite **set of actions**

  – P : a **state transition function**

    - $p(s', r | s, a)$: the probability of observing r and reaching s' after taking a at s.



Markov property

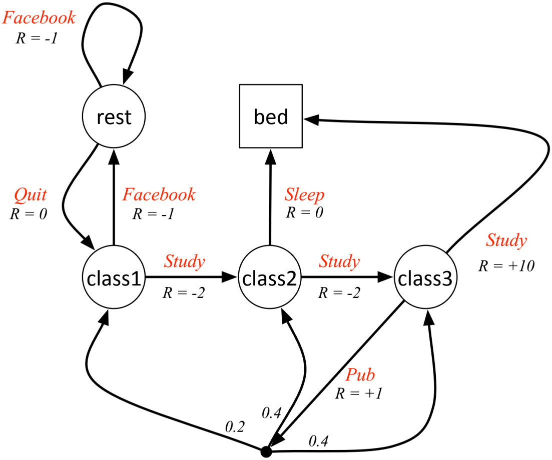
nction

$s, A_t = a]$

  – γ : a discount factor $\gamma \in [0,1]$

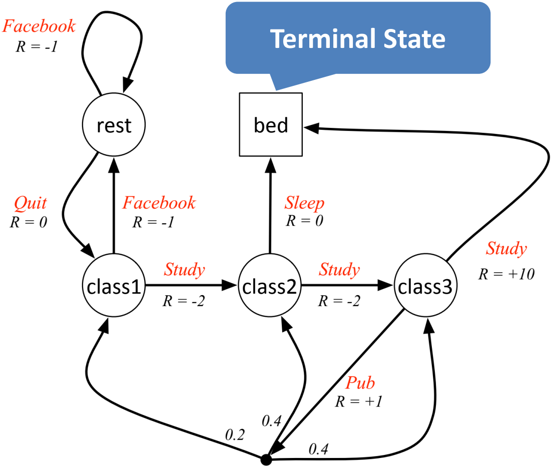
    - Role and function: later

An Example: Student MDP



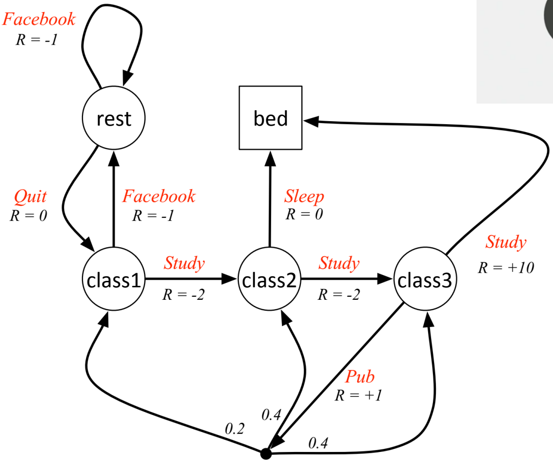
- $\langle S, A, P, R, \gamma \rangle$

An Example: Student MDP



- $\langle S, A, P, R, \gamma \rangle$

An Example: Student MDP



- $\langle S, A, P, R, \gamma \rangle$

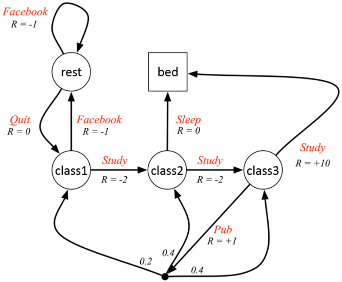
Next

- Covered
  - Formulation of the **environment**

- Next
  - Formulation of **agent's behavior**
  - **Interaction** between **agent** and **environment**

MDP: Policy & Trajectory

- $\langle S, A, P, R, \gamma \rangle$



- **Trajectory**: an agent's history
  - $s_1, a_1, \ldots, s_{t-1}, a_{t-1}, s_t$
  - E.g.,
    - c1, Study, c2, Sleep, bed
    - c1, Facebook, rest, Facebook, rest, Quit, c1, Study, c2, Study, c3, Study, bed
    - c1, Study, c2, Study, c3, Pub, c2, Sleep, bed

MDP: Policy & Trajectory

- $\langle S, A, P, R, \gamma \rangle$

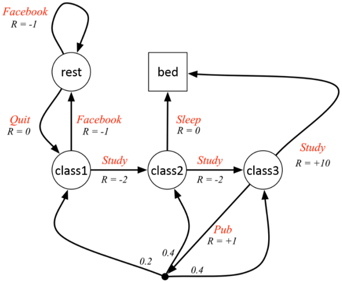
- **Policy**: a function defines an agent's behavior
  - A **deterministic** policy

    $$a_t = \boldsymbol{\pi(s_t)}$$

  - A **stochastic** policy

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

  - E.g.,
    - Study $= \pi(\text{class2})$
    - $\pi(\text{Facebook}|\text{class1}) = 1/3,\ \pi(\text{Study}|\text{class1}) = 2/3$

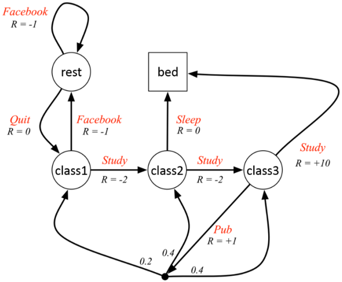


MDP: Reward & Return

- $\langle S, A, P, R, \gamma \rangle$

- **Reward** R_{t+1}
  - A feedback signal
  - How well agent is doing at step t

- **Return** G_t
  - Total discounted reward from step t



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

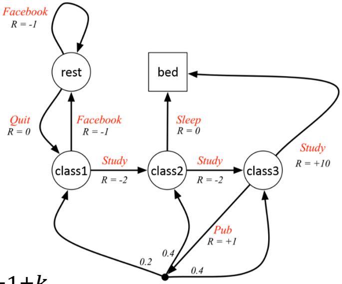
MDP: Reward & Return

- $\langle S, A, P, R, \gamma \rangle$

- Reward R_t

- Return
  - Total discounted reward from step t

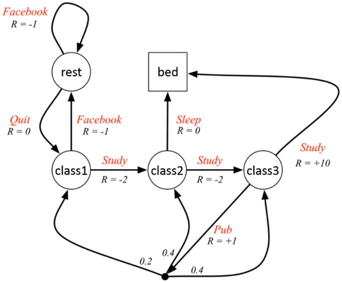
$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$$



- γ: defines present value of future rewards
- γ close to 0: "**myopic**" evaluation
- γ close to 1: " **far-sighted**" evaluation

MDP: Interaction with Environment

- "D" in MDP:
  Decision (policy) from the agent
  - $\boldsymbol{p(s_{t+1}, r_{t+1} | s_t, \pi(a_t|s_t))}$
  - Transition probability is **affected** by **agent's policy π**
  - E.g.,
    - $p(s_{t+1} = c2,\ r_{t+1} = +1 |\ s_t = c3, a_t = \text{Pub})$
      $= 0.4 * \pi(a_t = Pub|s_t = c3)$
  - **Policy π** in turn affects the **trajectory** and the corresponding **return**



MDP: Interaction with Environment

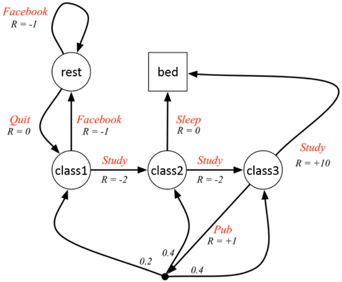
- "D" in MDP:
  Decision (policy) from the agent
  - $p(s_{t+1}, r_{t+1} | \, s_t, \pi(a_t | s_t))$

Nature of environment

Decision from agent

on proba … ed … t's policy

  - E.g.,
    - $p(s_{t+1} = c2,\ r_{t+1} = +1 | \ s_t = c3, a_t = \text{Pub})$
      $= 0.4 * \pi(a_t = Pub | s_t = c3)$



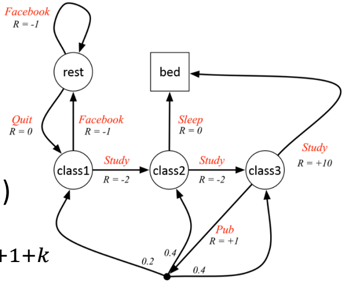
  - **Policy π** in turn affects the **trajectory** and the corresponding **return**

MDP: Goal of Learning

- $\langle S, A, P, R, \gamma \rangle$

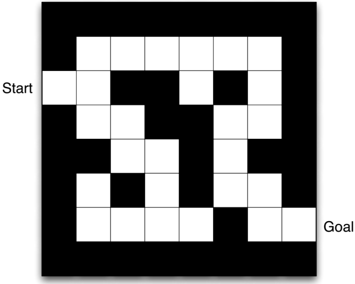
- Policy π
- Return G_t
  - (Total discounted reward from step t)

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



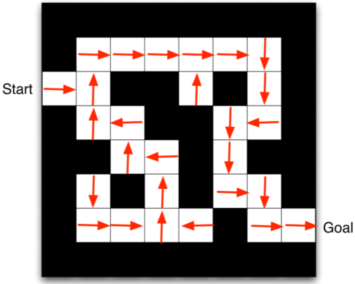
- **Goal** of Learning: find an **optimal π_*** which maximizes the **(expected) return G_t**

One More Example: Maze



- A **grid world** example
- **States** S: Agent's location
- **Actions** A: N, E, S, W
- **Dynamics** P: How actions (directions) change the states (locations)
- **Rewards** R: -1 per step

One More Example: Maze



- A deterministic **policy**:
  arrows are $\boldsymbol{\pi}(\boldsymbol{s})$ for each state s

Next

- Covered
  - Mathematical **formulation**
    - Environment
    - Agent
    - Interaction
  - **Goal** of learning

- Next
  - How to **evaluate** a **policy** $\boldsymbol{\pi}$?
  - How to **learn** an **optimal policy** $\boldsymbol{\pi}_*$?

REINFORCEMENT LEARNING

Mathematical Formulation

Value Functions & Bellman Equations

Assumptions

Approaches

Evaluating a Policy (I)

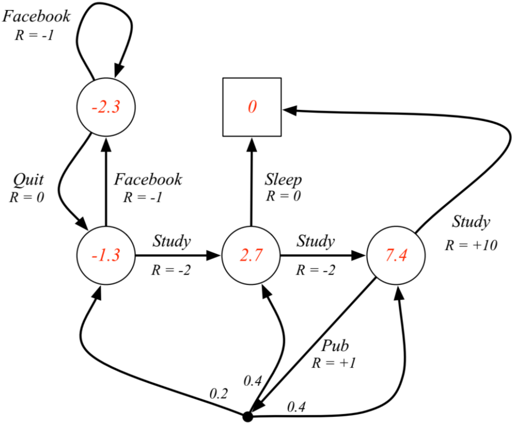
- Given: An MDP $\langle S, A, P, R, \gamma \rangle$ and a policy π

- Definition: **State-value function**

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

  - $p(s_{t+1}, r_{t+1} |\, s_t, \pi(a_t | s_t))$
  - $G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$
  - The **expected return** of π from state s

Student MDP Example



- $v_\pi(s)$ for $\pi(a|s) = 0.5, \gamma = 1$

- $v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$
- $G_t = R_{t+1} + \gamma R_{t+2} + \cdots$
  $= \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$

Evaluating a Policy (I)

- Expanding the state-value function:

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

Evaluating a Policy (I)

- Expanding the state-value function:

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

Evaluating a Policy (I)

- Expanding the state-value function:

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

Evaluating a Policy (I)

- Expanding the state-value function:

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

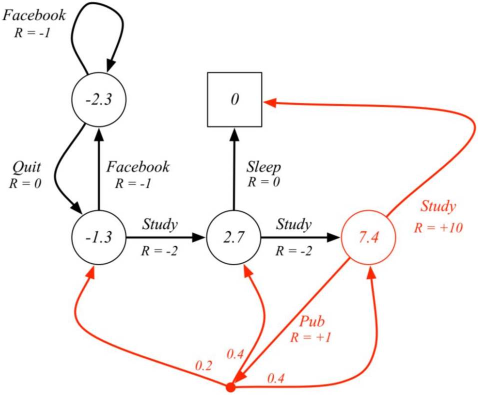
Bellman Equation for MDP (I)

- The **Bellman equation** for v_π

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

- Bellman equation decomposes the value function into two parts:
  - **Immediate reward** R_{t+1}
  - **Discounted value** $\gamma v_\pi(S_{t+1})$
  - Usage: an iterative way to compute $v_\pi(s)$, given $v_\pi(S_{t+1})$ is known

Student MDP Example



- MDP with $\pi(a|s) = 0.5, \gamma = 1$
- $v_\pi(s)$
  $= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$
  - Previously:
    $v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$
  - Now: compute $v_\pi(s)$ iteratively

$v_\pi(c3) = 7.4$
$= [0.5 * (10 + 0)] + [0.5 * (1 + 0.2 * -1.3 + 0.4 \ * 2.7 + 0.4 * 7.4)]$

Next

- Covered
  - **State-value function** $v_\pi(s)$
    - Evaluate expected return of π from state s
  - **Bellman equation** for v_π
    - Compute v_π iteratively

- Next
  - A **second way** to **evaluate** a policy $\boldsymbol{\pi}$
  - **Finding** an optimal policy $\boldsymbol{\pi}_*$

Evaluating a Policy (II)

- Given: An MDP $\langle S, A, P, R, \gamma \rangle$ and a policy π

- Definition: **Action-value function**

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

  - The **expected return** of **first taking** action $\boldsymbol{a}$ **then** following $\boldsymbol{\pi}$
  - $G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$

- Recall: State-value function $v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$

Bellman Equation for MDP (II)

- The **Bellman equation** for $q_\pi(s,a)$

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

  - Usage: later

- Recall: Value functions & their Bellman equations
  - **State-value** function

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

  - **Action-value** function

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

Next

- Covered
  - Value functions and **Bellman equations** to evaluate a policy
  - **State-value** function

    $$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \end{aligned}$$
  - **Action-value** function

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

- Next
  - **Finding** an optimal policy $\boldsymbol{\pi}_*$

Optimal Policy

- Definition: Partial ordering over policies
  $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s), \forall s$

- Theorem
  - There **exists** an **optimal policy**: $\boldsymbol{\pi_*} \geq \pi, \forall \pi$
  - All optimal policies achieve the **optimal value**:
    $v_{\pi_*}(s) = \boldsymbol{v_*(s)}$ and $q_{\pi_*}(s,a) = \boldsymbol{q_*(s,a)}$

Optimal Value Functions

- Definitions
  - **Optimal state-value** function

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

    - The **maximum** state-value function **over all policies**
  - **Optimal action-value** function

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

    - The **maximum** action-value function **over all policies**
- Specify the best possible performance in the MDP

Finding an Optimal Policy

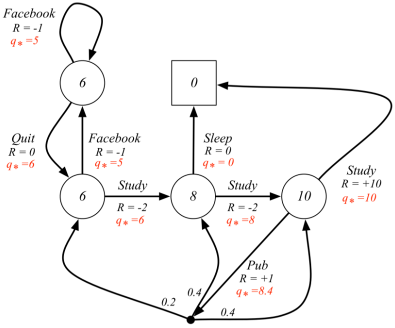
- An **optimal** policy:

$$\pi_*(a|s) = 1 \text{ if } a = \boldsymbol{\operatorname{argmax}_a q_*(s,a)}$$

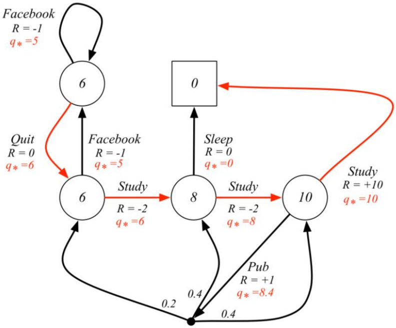
  - $q_*(s,a) = \max_\pi q_\pi(s,a)$
  - One immediately has the optimal policy if $q_*(s,a)$ is known.

Student MDP Example

$q_*(s, a)$ for $\gamma = 1$



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



$\pi_*(a|s) = 1$ if $a = \underset{a}{\operatorname{argmax}}\, q_*(s, a)$

Bellman Optimality Equations

- How to compute $q_*(s, a)$?
  - Applying **Bellman optimality equations**

- Similarly, **optimal value functions** also have the corresponding **Bellman optimality equations**

$$v_*(s) = \max_a \mathbb{E}[\boldsymbol{R_{t+1}} + \boldsymbol{\gamma v_*(S_{t+1})}|S_t = s, A_t = a]$$

$$q_*(s, a) = \mathbb{E}[\boldsymbol{R_{t+1}} + \boldsymbol{\gamma \max_{a'} q_*(S_{t+1}, a')}\,|S_t = s, A_t = a]$$

$$= \mathbb{E}[\boldsymbol{R_{t+1}} + \boldsymbol{\gamma v_*(S_{t+a})}|S_t = s, A_t = a]$$

Solving an Optimal Policy

- Approach #1:
  Can we simply solve the Bellman optimality equations to obtain $q_*(s, a)$?

$$\boldsymbol{q_*(s, a)} = \mathbb{E}[R_{t+1} + \gamma \max_{a'} \boldsymbol{q_*(S_{t+1}, a')} \,|S_t = s, A_t = a]$$

- No. Since
  - Bellman optimality equation is non-linear
  - In general **no closed form solution**

Finding an Optimal Policy

- Approach #2:
  Second form of $q_*(s,a)$:

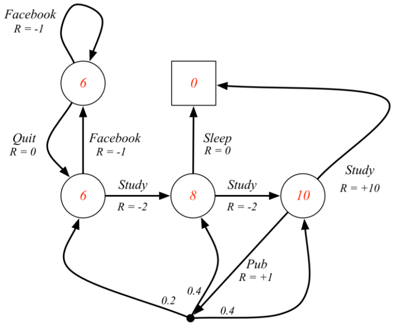
$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma \boldsymbol{v}_*(\boldsymbol{S}_{\boldsymbol{t+a}}) | S_t = s, A_t = a]$$

- Recall:

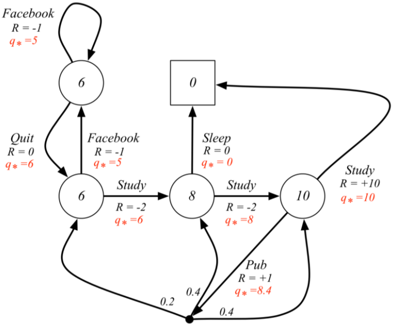
$$\begin{aligned} q_*(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \, | S_t = s, A_t = a] \\ &= \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+a}) | S_t = s, A_t = a] \end{aligned}$$

Student MDP Example

$v_*(s)$ for $\gamma = 1$



$q_*(s,a)$ for $\gamma = 1$

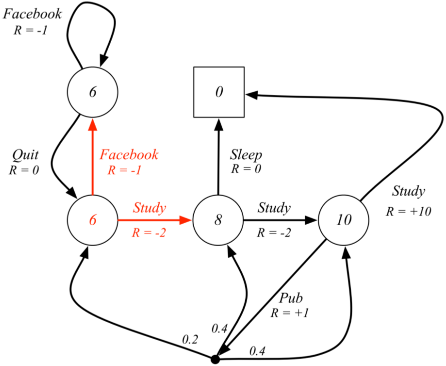


$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma \boldsymbol{v_*(S_{t+a})} | S_t = s, A_t = a]$$

Student MDP Example

$$v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$$

$$v_*(c1) = \max\{-1 + 6, -2 + 8\} = 6$$



Finding an Optimal Policy

- How to compute $v_*(s)$?

- Recall:
  $v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$

Next

- Covered
  - Definition of the **optimal policy** $\pi_*(s)$
  - **Bellman optimality equations**
  - Finding $\pi_*(s)$ given either $q_*(s, a)$ or $v_*(s)$
- Next
  - **Iterative** approaches inspired by **Bellman equations**

REINFORCEMENT LEARNING

Mathematical Formulation

Value Functions & Bellman Equations

Assumptions: Known Model

Approaches: Dynamic Programming

Value Iteration

- Recall: Bellman optimality equation

$$\boldsymbol{v}_*(\boldsymbol{s}) = \max_a \mathbb{E}[R_{t+1} + \gamma \boldsymbol{v}_*(\boldsymbol{S_{t+1}}) | S_t = s, A_t = a]$$

  - $\langle S, A, P, R, \gamma \rangle$ is given
  - Previously: If $\boldsymbol{v}_*(\boldsymbol{S_{t+1}})$ is known, RHS above computes $v_*(s)$

Value Iteration

- Recall: Bellman optimality equation

$$\boldsymbol{v_*(s)} = \max_a \mathbb{E}[R_{t+1} + \gamma \boldsymbol{v_*(S_{t+1})} | S_t = s, A_t = a]$$

  - $v_*(s)$ **stores** and **reuses** values
  - **Recursive decomposition** of Bellman equation

- **Dynamic Programming** applies
  - Break a problem down into subproblems
  - Combine solutions to subproblems

Value Iteration

- Recall: Bellman optimality equation

$$\boldsymbol{v_*(s)} = \max_a \mathbb{E}[R_{t+1} + \gamma \boldsymbol{v_*(S_{t+1})} | S_t = s, A_t = a]$$

  - $v_*(s)$ **stores** and **reuses** values
  - **Recursive decomposition** of Bellman equation

- Idea: turning **Bellman optimality equation** into **iterative updates**:

$$v_{t+1}(s) \leftarrow \max_a \mathbb{E}[R_{t+1} + \gamma v_t(S_{t+1}) | S_t = s, A_t = a]$$

Value Iteration

- Recall: Bellman optimality equation

$$\boldsymbol{v}_*(\boldsymbol{s}) = \max_a \mathbb{E}[R_{t+1} + \gamma \boldsymbol{v}_*(\boldsymbol{S}_{t+1}) | S_t = s, A_t = a]$$

  - $v_*(s)$ **stores** and **reuses** values
  - **Recursive decomposition** of Bellman equation

- Idea: turning **Bellman optimality equation** into **iterative updates**:

$$v_{t+1}(s) \leftarrow \max_a \mathbb{E}[R_{t+1} + \gamma v_t(S_{t+1}) | S_t = s, A_t = a]$$

Combine solutions

Value Iteration

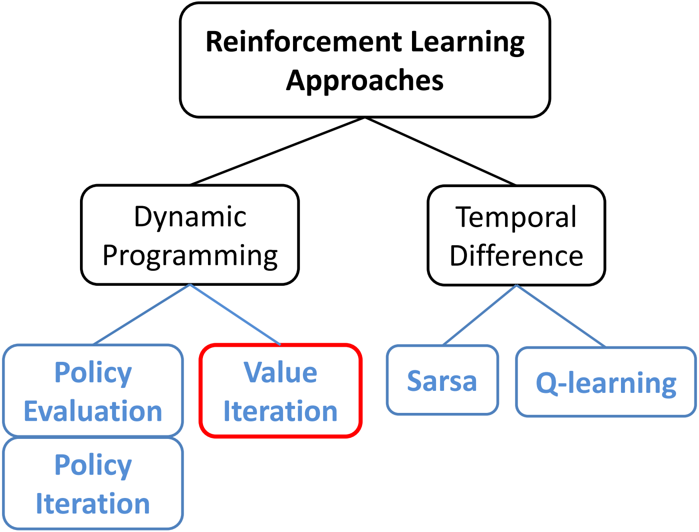
- $v_{t+1}(s) \leftarrow \max_a \mathbb{E}[R_{t+1} + \gamma v_t(S_{t+1}) | S_t = s, A_t = a]$

Initialize array V arbitrarily (e.g., $V(s) = 0$ for all $s \in \mathcal{S}^+$)

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi \approx \pi_*$, such that
  $\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

Reinforcement Learning Approaches

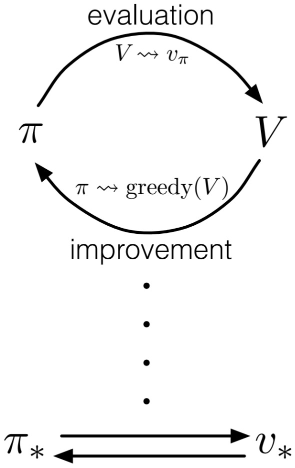


Next

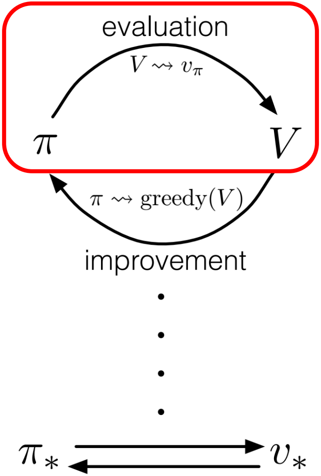
- Covered
  - The first **iterative method**, **Value Iteration**, to learn $v_*(s)$ so that we can use $v_*(s)$ to find out the optimal policy $\pi_*(s)$.

- Next
  - Introduce another iterative method called **Policy Iteration**
    - The **foundation** of many state-of-the-art RL algorithms.

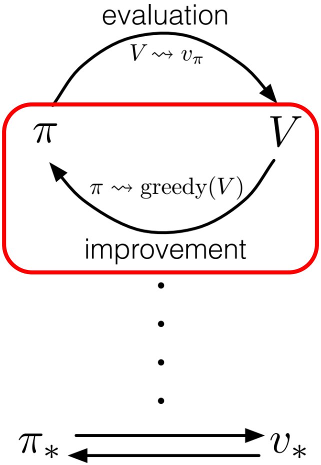
Learning Iteratively



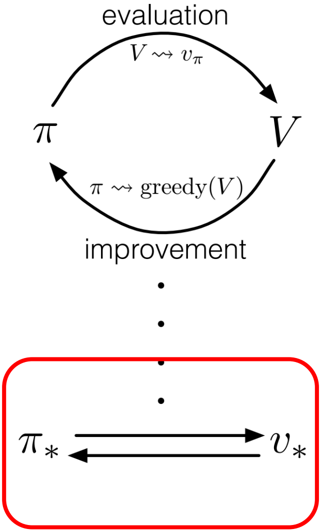
Learning Iteratively



Learning Iteratively



Learning Iteratively



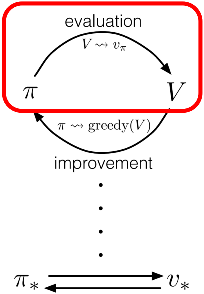
Policy Evaluation

- Bellman equation

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

  - $v_\pi(s)$ stores and reuses solutions
  - Recursive decomposition of Bellman equation

- Dynamic Programming applies
  - Break a problem down into subproblems
  - Combine solutions to subproblems



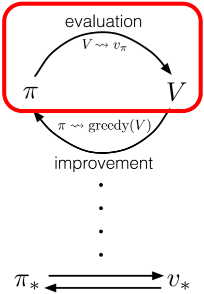
Policy Evaluation

- **Iterative** policy **evaluation**

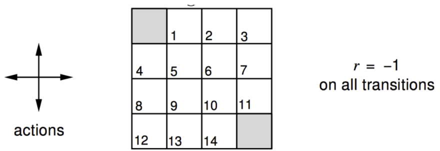
$$v_\pi(s) \approx v_{t+1}(s) \leftarrow \mathbb{E}_\pi[R_{t+1} + \gamma v_t(S_{t+1}) | S_t = s]$$

  - Stores and reuses solutions
  - Recursive decomposition of Bellman equation

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)



Iterative Policy Evaluation: Small Gridworld



- 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

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

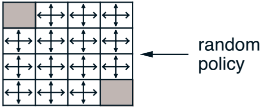
Iterative Policy Evaluation: Small Gridworld

v_k for the Random Policy

Greedy Policy w.r.t. v_k

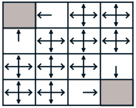
$k = 0$

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



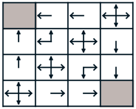
$k = 1$

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



Iterative Policy Evaluation: Small Gridworld

$k = 3$

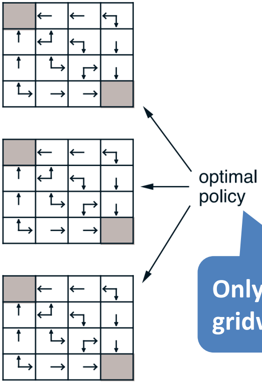
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

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



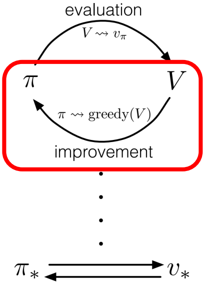
Only work for small gridworld!

Policy Improvement

- **Improve** a given policy π by acting greedily:

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

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      old-action $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
      If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$



Policy Improvement

- Improve a given policy π by acting greedily:

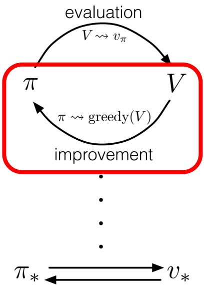
$$\pi' = \arg\max_a q_\pi(s, a)$$

- $\Rightarrow$ improve the value from any state s over one step:

$$q_\pi(s, \pi'(s)) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

- $\Rightarrow$ improve the value function

$$v_{\pi'}(s) \geq v_\pi(s)$$

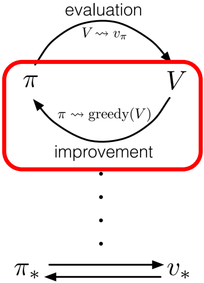


Policy Improvement

- When improvement stops:

$$q_\pi(s, \pi'(s)) = \max_a q_\pi(s, a) = v_\pi(s)$$

- $\Rightarrow$ the Bellman optimality equation $v_*(s) = \max_a q_*(s, a)$ is satisfied.
- $\Rightarrow$ $v_\pi(s) = v_*(s)$, so π is **optimal**



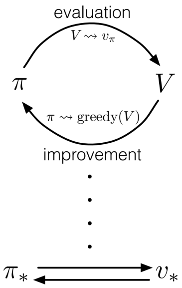
Policy Iteration

- Policy Evaluation + Policy Improvement

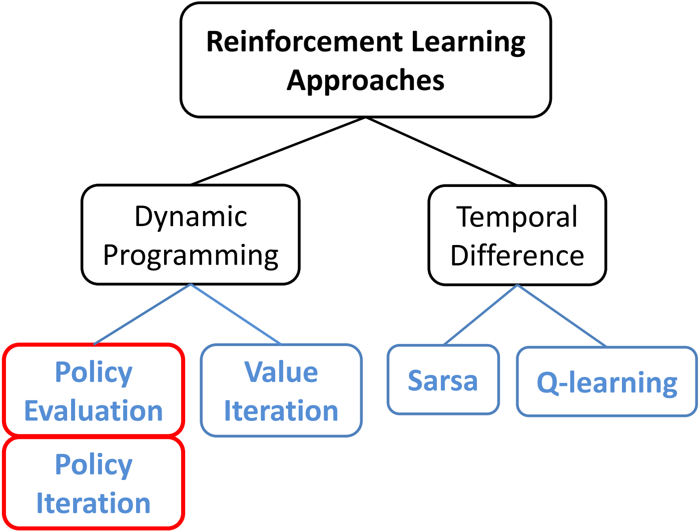
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
       $\Delta \leftarrow 0$
       For each $s \in \mathcal{S}$:
           $v \leftarrow V(s)$
           $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\big[r + \gamma V(s')\big]$
           $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
       old-action $\leftarrow \pi(s)$
       $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
       If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2



Reinforcement Learning Approaches



Next

- Covered
  - **Iterative methods** for finding **optimal $\boldsymbol{\pi}_*$** when $\langle S, A, P, R, \gamma \rangle$ is **known**
    - Policy Iteration
    - Value Iteration

- Next
  - What if P and R of $\langle S, A, P, R, \gamma \rangle$ is **unknown**?

REINFORCEMENT LEARNING

Mathematical Formulation

Value Functions & Bellman Equations

Assumptions: Unknown Model

Approaches: Temporal Difference

Temporal Difference Learning

- Now: P and R **unknown**
  - Policy Iteration & Value Iteration **don't work**

- Idea: **Temporal difference** update
  - **Current estimate** q_t and **new trajectory** s, a, r, s'
  - $(1-\alpha) \cdot \boldsymbol{q_t}(s,a) + \alpha \cdot \left(\boldsymbol{r} + \boldsymbol{\gamma q_t}(\boldsymbol{s'}, \boldsymbol{a'})\right)$
    $= \boldsymbol{q_t}(s,a) + \alpha \cdot \left[\left(\boldsymbol{r} + \boldsymbol{\gamma q_t}(\boldsymbol{s'}, \boldsymbol{a'})\right) - \boldsymbol{q_t}(s,a)\right]$
  - $\boldsymbol{q_{t+1}}(s,a) \leftarrow \boldsymbol{q_t}(s,a) + \alpha \cdot \left[\left(\boldsymbol{r} + \boldsymbol{\gamma q_t}(\boldsymbol{s'}, \boldsymbol{a'})\right) - \boldsymbol{q_t}(s,a)\right]$

Temporal Difference Learning

Sarsa: An on-policy TD control algorithm

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

- ϵ–greedy sampling on $q_t(s',a)$:

$$a' = \begin{cases} \max_{a'} q_t(s',a') & \text{with probability } 1-\epsilon \\ \text{uniformly at random with probability } \epsilon \end{cases}$$

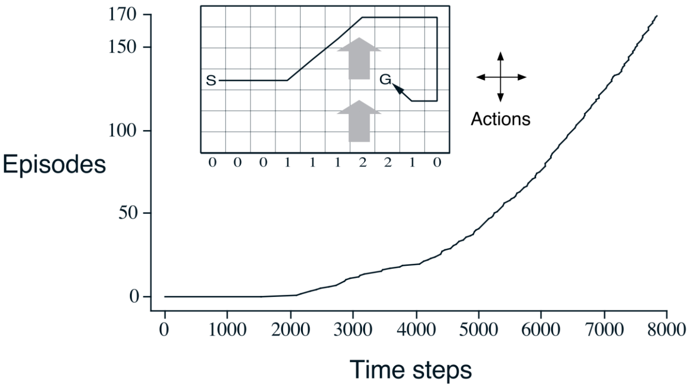
Temporal Difference Learning

Q-learning: An off-policy TD control algorithm

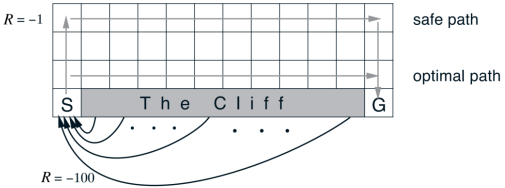
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ϵ-greedy)
        Take action A, observe R, S'
        $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
        $S \leftarrow S'$
    until S is terminal

- Choose a' such that $q_t(s', a') = \max_a q_t(s', a)$
- $q_{t+1}(s,a) \leftarrow q_t(s,a) + \alpha \cdot \left(r + \gamma \max_a q_t(s',a) - q_t(s,a)\right)$

Sarsa Example: Windy Gridworld



Example: Cliff Walking



Reinforcement Learning Approaches