CS 540 Introduction to Artificial Intelligence
Reinforcement Learning II

Yudong Chen
University of Wisconsin-Madison

Dec 7, 2021

Announcements

- **Couse evaluation survey**

- **Homework**:
  - HW10 due next Tuesday (before last class)

- **Final exam:** Dec 20, 2:45-4:45pm, online

- **Class roadmap:**
  - Today: Reinforcement Learning II
  - Thursday: Review on search, games, RL
  - Next Tuesday: Ethics and Trust in AI

Outline

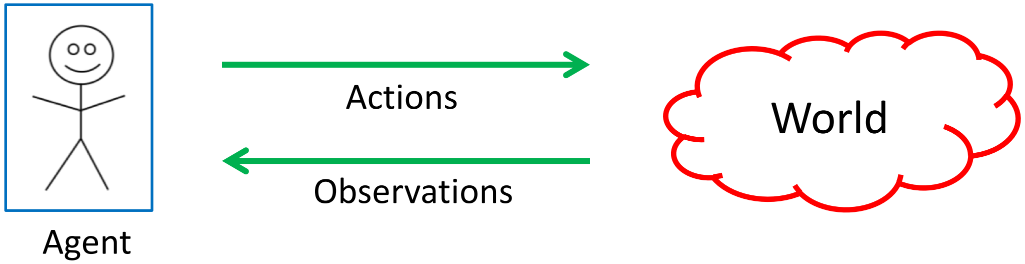
- Review of reinforcement learning
  - MDPs, value functions, value iteration

- Q-learning
  - Q function, SARSA, deep Q-learning

Building The Theoretical Model

Basic setup:

- Set of states, S
- Set of actions A



- Information: at time t, observe state $s_t \in S$. Get reward r_t
- Agent makes choice $a_t \in A$. State changes to s_{t+1}, continue

Goal: find a map from **states to actions** maximize rewards.

A "policy"

Markov Decision Process (MDP)

The formal mathematical model:

- **State set** S. Initial state s_0. **Action set** A
- **State transition model:** $P(s_{t+1} | s_t, a_t)$
  - Markov assumption: transition probability only depends on s_t and a_t, and not previous actions or states.
- **Reward function:** $r(s_t)$
- **Policy**: $\pi(s) : S \rightarrow A$ action to take at a particular state.

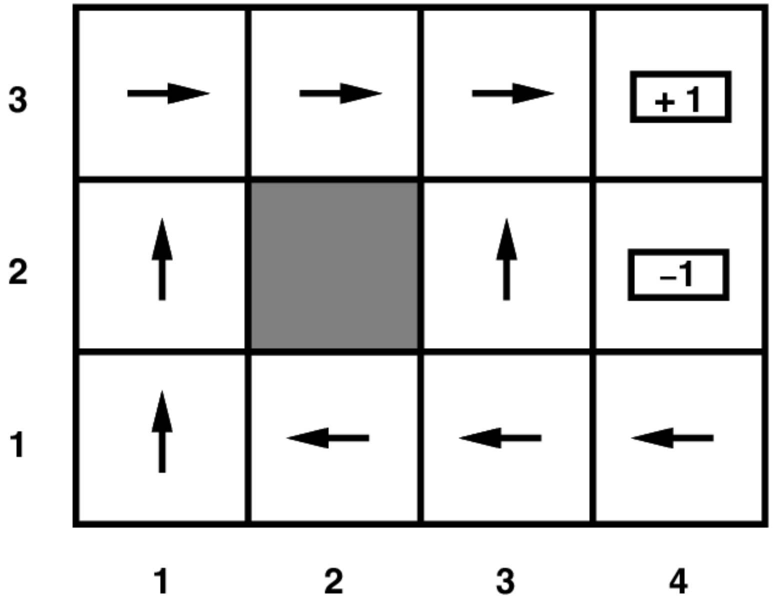
$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots$$

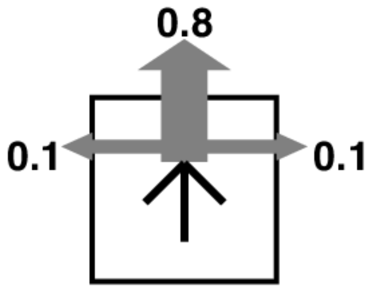
$a_0 = \pi(s_0)$, $a_1 = \pi(s_1)$

$r(s_0)$ $r(s_1)$ $r(s_2)$

Grid World Optimal Policy

Note: (i) Robot is unreliable (ii) Reach target fast





$r(s) = -0.04$ for every non-terminal state

Defining the Optimal Policy

For policy π, **expected utility** over all possible state sequences from s_0 produced by following that policy:

$$V^{\pi}(s_0) = \sum_{\substack{\text{sequences} \\ \text{starting from } s_0}} P(\text{sequence})U(\text{sequence})$$

Called the **value function** (for π, s_0)

Discounting Rewards

One issue: these are infinite series. **Convergence**?

- Solution

$$U(s_0, s_1 \ldots) = r(s_0) + \gamma r(s_1) + \gamma^2 r(s_2) + \ldots = \sum_{t \geq 0} \gamma^t r(s_t)$$

- Discount factor γ between 0 and 1
  - Set according to how important **present** is VS **future**
  - Note: has to be less than 1 for convergence

Values and Policies

Now that $V^{\pi}(s_0)$ is defined what a should we take?

- First, set V*(s) to be expected utility for **optimal** policy from s
- What's the expected utility of an action?
  - Specifically, action a in state s?

$$\sum_{s'} P(s'|s,a)V^*(s')$$

All the states we could go to

Transition probability

Expected rewards

Obtaining the Optimal Policy

We know the expected utility of an action.

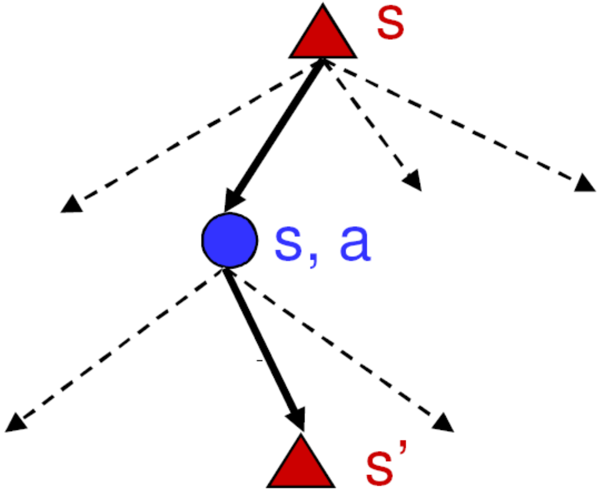
- So, to get the optimal policy, compute

$$\pi^*(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s,a)V^*(s')$$

All the states we could go to

Transition probability

Expected rewards



Credit L. Lazbenik

Bellman Equation

Let's walk over one step for the value function:

$$V^*(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V^*(s')$$

Current state reward

Discounted expected future **rewards**

- Bellman: inventor of dynamic programming

Value Iteration

Q: how do we find $V^*(s)$?

- Why do we want it? Can use it to get the best policy
- Know: reward $r(s)$, transition probability $P(s'|s,a)$
- Also know $V^*(s)$ satisfies Bellman equation (recursion above)

A: Use the property. Start with $V_0(s)=0, \forall s$. Then update

$$V_{i+1}(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a) V_i(s') \qquad \forall s.$$

$$V_{i+1}(s) = r(s) + \gamma \max_a \sum_{s'} P(s'|s,a)V_i(s')$$

$S = \{A, B\}$

$i=0$: $V_0(A) = 0$, $V_0(B) = 0$

$i=1$:

$$V_1(A) = r(A) + \gamma \max_a \Big\{ \underbrace{P(A|A,a)V_0(A)}_{s'=A} + \underbrace{P(B|A,a)V_0(B)}_{s'=B} \Big\}$$

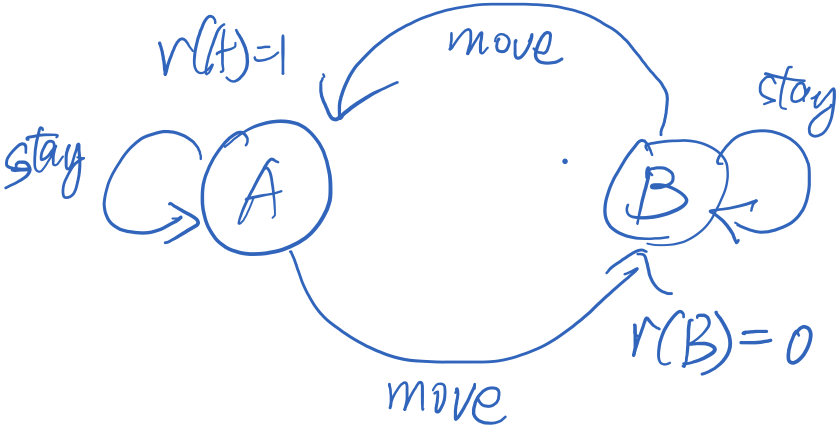
$$V_1(B) = r(B) + \gamma \max_a \Big\{ P(A|B,a)V_0(A) + P(B|B,a)V_0(B) \Big\}$$

$i=2$:

Break & Quiz

Q 1.1 Consider an MDP with 2 states {*A, B*} and 2 actions: **"stay"** at current state and "**move**" to other state. Let **r** be the reward function such that **r**(*A*) = 1, **r**(*B*) = 0. Let γ be the discounting factor. What is the optimal policy $\pi(A)$ and $\pi(B)$? What are $V^*(A)$, $V^*(B)$?

- A. Stay, Stay, $1/(1-\gamma)$, 1
- B. Stay, Move, $1/(1-\gamma)$, $1/(1-\gamma)$
- C. Move, Move, $1/(1-\gamma)$, 1
- D. Stay, Move, $1/(1-\gamma)$, $\gamma/(1-\gamma)$



$V^*(A) = 1 + \gamma + \gamma^2 + \dots = \frac{1}{1-\gamma}$

$V^*(B) = 0 + \gamma + \gamma^2 + \dots = \frac{\gamma}{1-\gamma}$

Break & Quiz

Q 1.1 Consider an MDP with 2 states {*A, B*} and 2 actions: **"stay"** at current state and "**move**" to other state. Let **r** be the reward function such that **r**(*A*) = 1, **r**(*B*) = 0. Let γ be the discounting factor. What is the optimal policy $\pi(A)$ and $\pi(B)$? What are $V^*(A)$, $V^*(B)$?

- A. Stay, Stay, 1/(1-γ), 1
- B. Stay, Move, 1/(1-γ), 1/(1-γ)
- C. Move, Move, 1/(1-γ), 1
- **D. Stay, Move, 1/(1-γ), γ/(1-γ)**

Break & Quiz

Q 1.1 Consider an MDP with 2 states {*A*, *B*} and 2 actions: **"stay"** at current state and "**move**" to other state. Let **r** be the reward function such that **r**(*A*) = 1, **r**(*B*) = 0. Let γ be the discounting factor. What is the optimal policy $\pi(A)$ and $\pi(B)$? What are $V^*(A)$, $V^*(B)$?

- A. Stay, Stay, 1/(1-γ), 1
- B. Stay, Move, 1/(1-γ), 1/(1-γ)
- C. Move, Move, 1/(1-γ), 1
- **D. Stay, Move, 1/(1-γ), γ/(1-γ)** Note: want to stay at A, if at B, move to A. Starting at A, sequence A,A,A,… rewards 1, γ, γ^2,…. Start at B, sequence B,A,A,… rewards 0, γ, γ^2,…. Sums to 1/(1-γ), γ/(1-γ).

Q-Learning

$Q(s,a)$

$\max_a \left\{ r(s) + \gamma \sum_{s'} P(s'|s,a)\, V^*(s') \right\}$

What if we don't know transition probability $P(s'|s,a)$?

from data or experience.

- Need a way to learn to act without it.
- **Q-learning**: get an action-value function $Q(s,a)$ that tells us the value of doing a in state s
- Note: $V^*(s) = \max_a Q(s,a)$
- Now, we can just do $\pi^*(s) = \arg\max_a Q(s,a)$
  - But need to estimate Q!

Q-Learning Iteration

How do we get Q(*s*,*a*)?

- Similar iterative procedure

new estimate of value (s_t, a_t)

old estimate of value of s_t, a_t

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r(s_t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

$$= (1-\alpha)\, Q(s_t, a_t) + \alpha[r(s_t) + \gamma \max_a Q(s_{t+1}, a)]$$

Learning rate

Idea: combine old value and new estimate of future value.

Note: We are using a policy to take actions; based on Q!

Exploration Vs. Exploitation

General question!

- **Exploration:** take an action with unknown consequences
  - **Pros**:
    - Get a more accurate model of the environment
    - Discover higher-reward states than the ones found so far
  - **Cons**:
    - When exploring, not maximizing your utility
    - Something bad might happen
- **Exploitation:** go with the best strategy found so far
  - **Pros**:
    - Maximize reward as reflected in the current utility estimates
    - Avoid bad stuff
  - **Cons**:
    - Might also prevent you from discovering the true optimal strategy

Q-Learning: Epsilon-Greedy Policy

How to **explore**?

- With some 0<ε<1 probability, take a random action at each state, or else the action with highest Q(*s*,*a*) value.

exploit

w.p. $1-\varepsilon$

$$a = \begin{cases} \text{argmax}_{a \in A} Q(s, a) & \text{uniform}(0,1) > \epsilon \\ \text{random } a \in A & \text{otherwise} \end{cases}$$

w.p. ε

explore.

Q-Learning: SARSA

An alternative:

Learning rate

- Just use the next action, no max over actions:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r(s_t) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

compare to Q learning

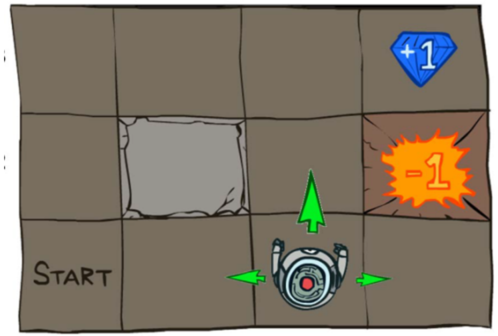
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \max_a [r(s_t) + \gamma (Q s_{t+1}, a) - Q(s_t, a_t)]$$

- Called state–action–reward–state–action (**SARSA**)
- Can use with epsilon-greedy policy

Q-Learning Details

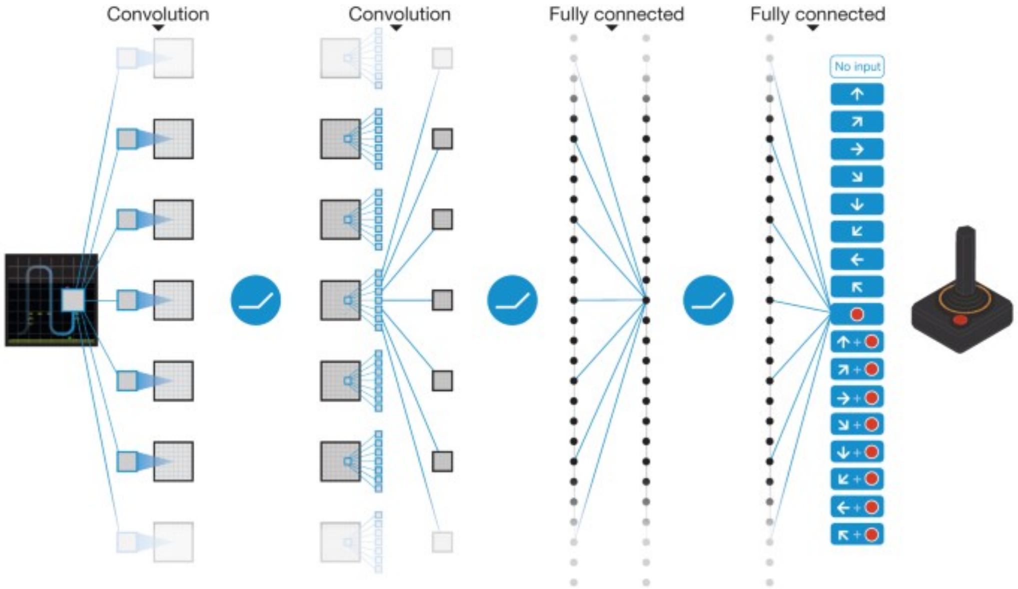
Note: if we have a **terminal** state, the process ends

- An **episode**: a sequence of states ending at a terminal state
- Want to run on many episodes
- Slightly different Q-update for terminal states (see homework!)



Deep Q-Learning

How do we get Q(*s*,*a*)?



Mnih et al, "Human-level control through deep reinforcement learning"

Break & Quiz

Q 2.1 For Q learning to converge to the true Q function, we must

- A. Visit every state and try every action
- B. Perform at least 20,000 iterations.
- C. Re-start with different random initial table values.
- D. Perform exploitation instead of exploration.

Break & Quiz

Q 2.1 For Q learning to converge to the true Q function, we must

- **A. Visit every state and try every action**
- B. Perform at least 20,000 iterations.
- C. Re-start with different random initial table values.
- D. Perform exploitation instead of exploration.

Break & Quiz

Q 2.1 For Q learning to converge to the true Q function, we must

- **A. Visit every state and try every action**
- B. Perform at least 20,000 iterations. (No: this is dependent on the particular problem, not a general constant).
- C. Re-start with different random initial table values. (No: this is not necessary in general).
- D. Perform exploitation instead of exploration. (No: insufficient exploration means potentially unupdated state action pairs).

Summary of RL

- Reinforcement learning setup
- Mathematical formulation: MDP
- Value functions & the Bellman equation
- Value iteration
- Q-learning

Acknowledgements: Based on slides from Fred Sala, Yin Li, Jerry Zhu, Svetlana Lazebnik, Yingyu Liang, David Page, Mark Craven, Pieter Abbeel, Dan Klein