CS 540 Introduction to Artificial Intelligence
Reinforcement Learning I

Yudong Chen
University of Wisconsin-Madison

Dec 2, 2021

Announcements

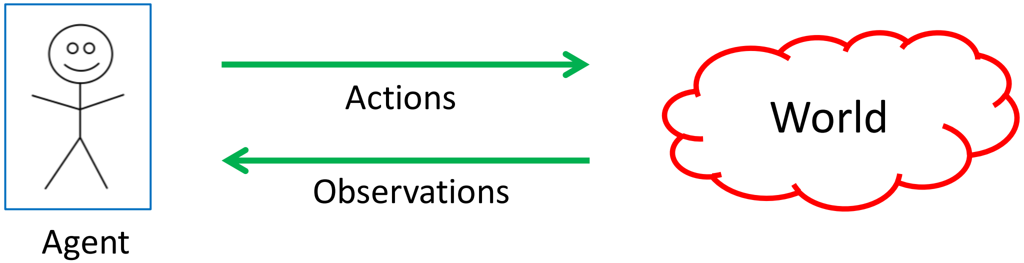
- **Homeworks**:
  - HW9 due next Tue
  - HW10 released
- **Final**: administrative details out soon
- Class roadmap:
  - Today and next Tuesday: Reinforcement Learning

Outline

- Introduction to reinforcement learning
  - Basic concepts, mathematical formulation, MDPs, policies
- Valuing policies
  - Value functions, Bellman equation, value iteration
- Q-learning (time permitted)
  - Q function, SARSA

Back to Our General Model

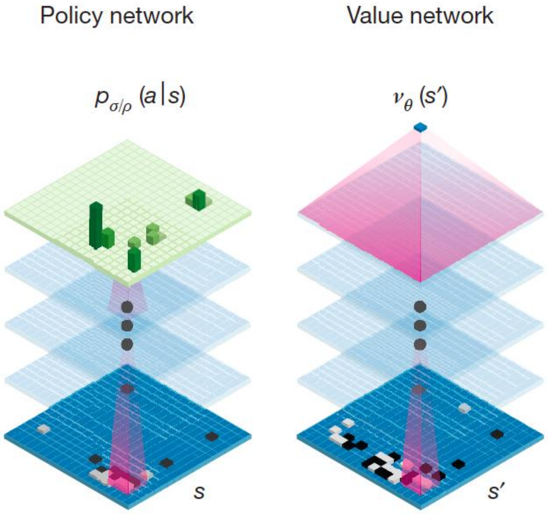
We have an **agent** **interacting** with the **world**



- Agent receives a reward based on state of the world
  - **Goal**: maximize reward / utility **($$$)**
  - Note: **data** consists of actions & observations
    - Compare to unsupervised learning and supervised learning

Examples: Gameplay Agents

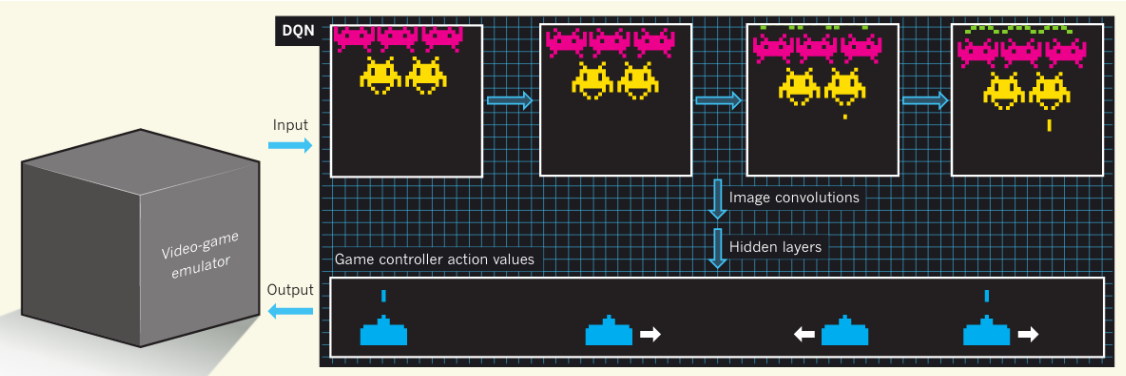
AlphaZero:



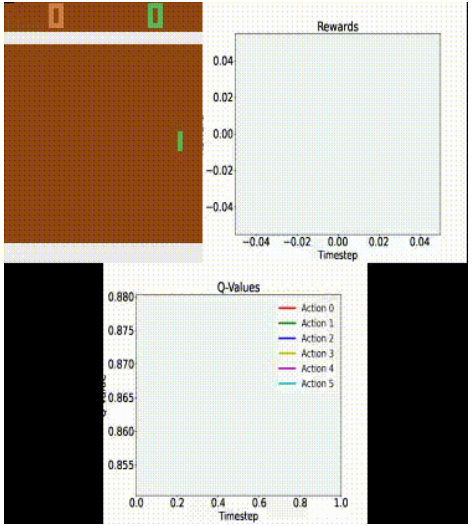
https://deepmind.com/research/alphago/

Examples: Video Game Agents

Pong, Atari



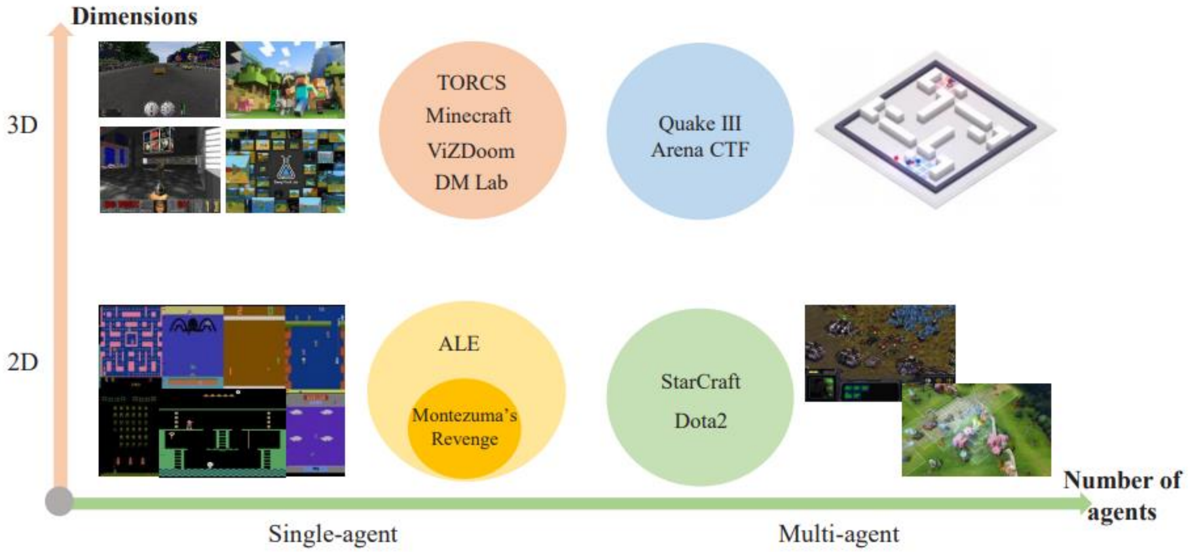
Mnih et al, "Human-level control through deep reinforcement learning"



A. Nielsen

Examples: Video Game Agents

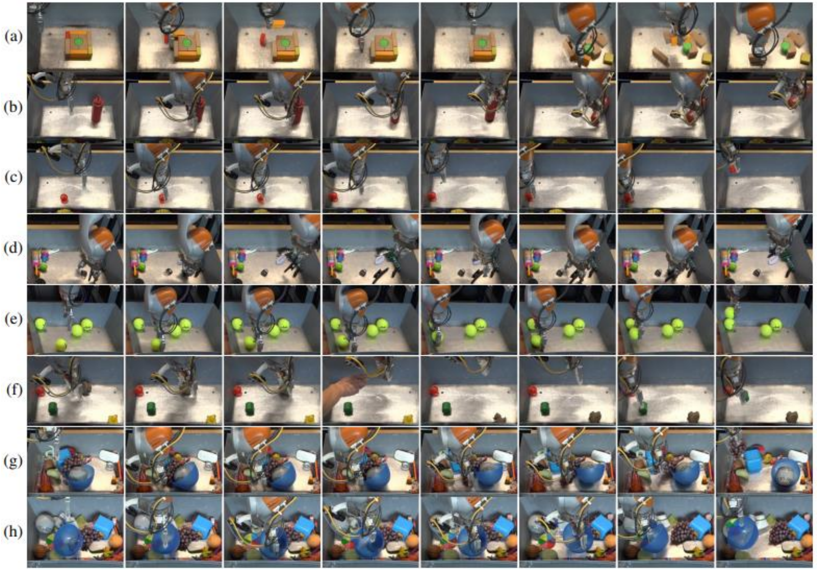
Minecraft, Quake, StarCraft, and more!



Shao et al, "A Survey of Deep Reinforcement Learning in Video Games"

Examples: Robotics

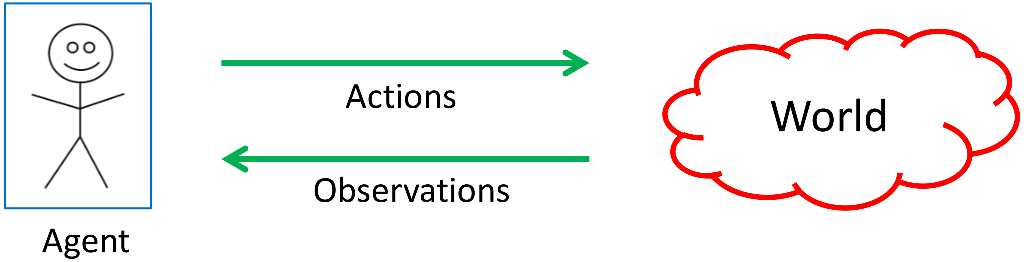
Training robots to perform tasks (e.g., grasp!)



Ibarz et al, " How to Train Your Robot with Deep Reinforcement Learning – Lessons We've Learned "

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)

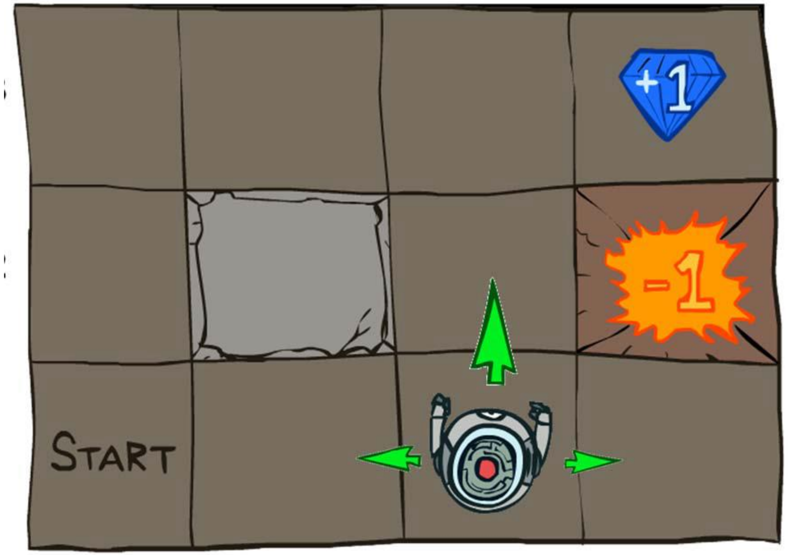
The formal mathematical model:

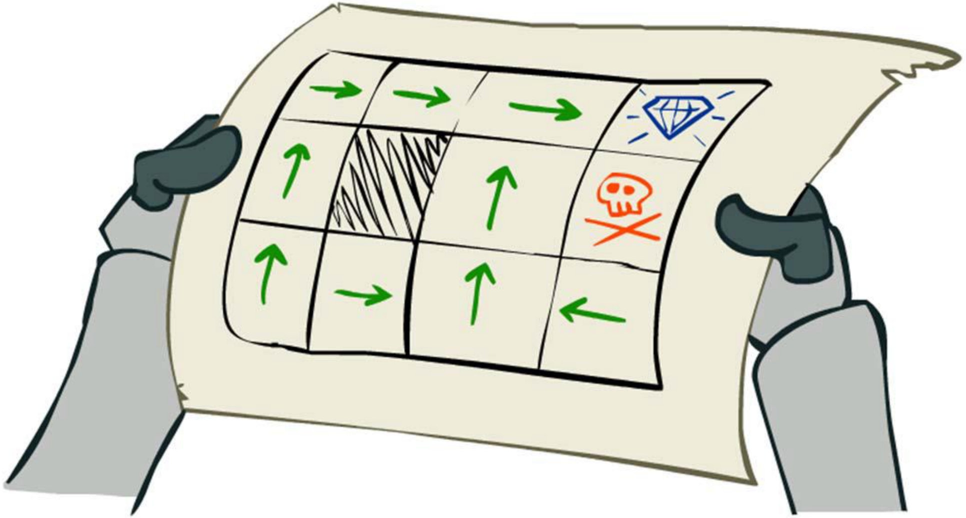
- **State set** S. Initial state s_0. **Action set** A
- **State transition model:** $P(s_{t+1} | s_t, a_t)$
  - Markov assumption: transition probability only depends on s_t and a_t, and not previous actions or states.
- **Reward function:** $r(s_t)$
- **Policy**: $\pi(s) : S \rightarrow A$ action to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

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

Example of MDP: Grid World

Robot on a grid; goal: find the best policy

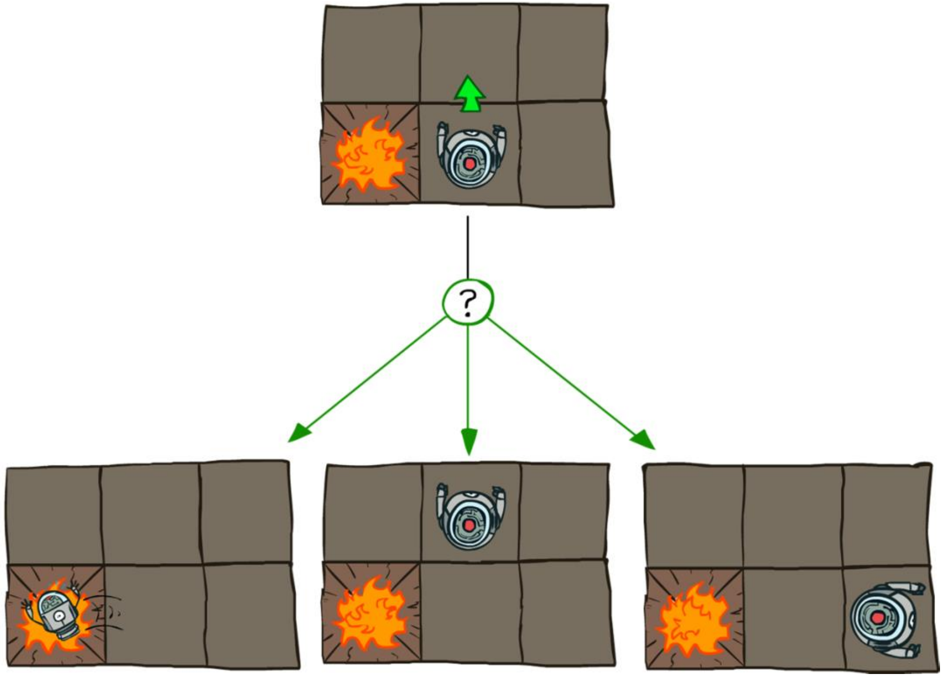


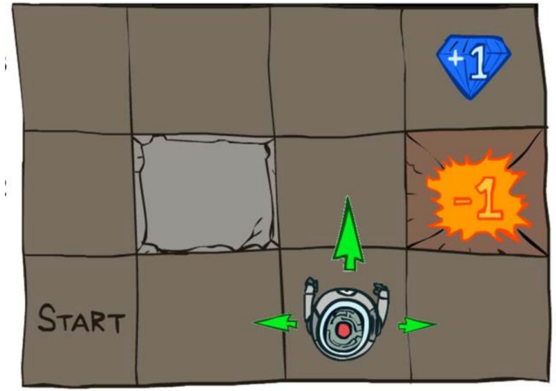


Source: P. Abbeel and D. Klein

Example of MDP: Grid World

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

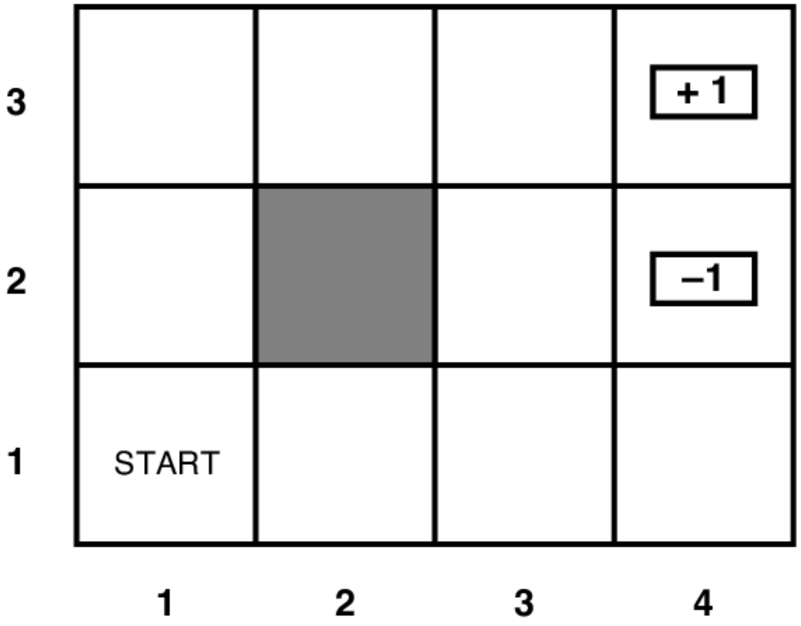


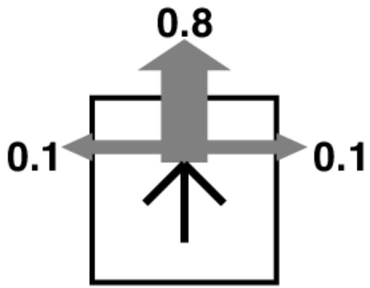


$r(s) = -0.04$ for every non-terminal state

Grid World Abstraction

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

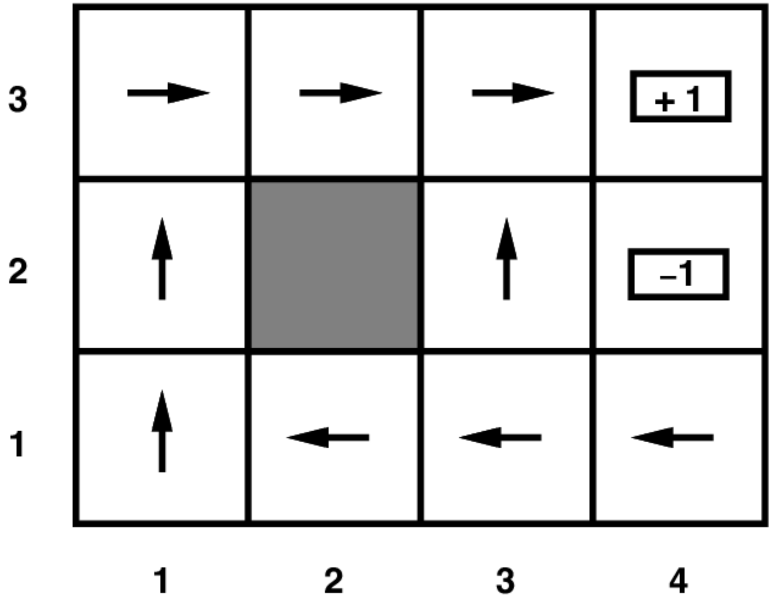


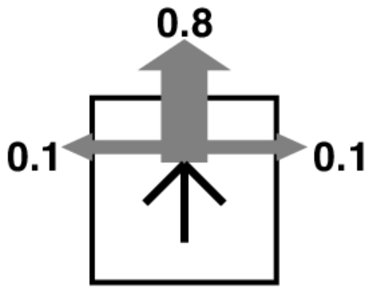


$r(s) = -0.04$ for every non-terminal state

Grid World Optimal Policy

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





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

Back to MDP Setup

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.

How do we find the best policy?

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

Break & Quiz

Q 1.1 Which of the following statement about MDP is **not** true?

- A. The reward function must output a scalar value
- B. The policy maps states to actions
- C. The probability of next state can depend on current and previous states
- D. The solution of MDP is to find a policy that maximizes the cumulative rewards

Value Function

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

From Value to Policy

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

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

All the states we could go to

Transition probability

Expected rewards

Slight Problem...

Now we can get the optimal policy by doing

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

- So we need to know $V^*(s)$.
  - But it was defined in terms of the optimal policy!
  - So we need some other approach to get $V^*(s)$.
  - Need some other **property** of the value function!

Bellman Equation

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

$$V^*(s) = r(s) + \gamma \underbrace{\max_a \sum_{s'} P(s'|s,a)V^*(s')}_{\text{Discounted expected future } \textbf{rewards}}$$

Current state reward

- Bellman: inventor of dynamic programming

Value Iteration

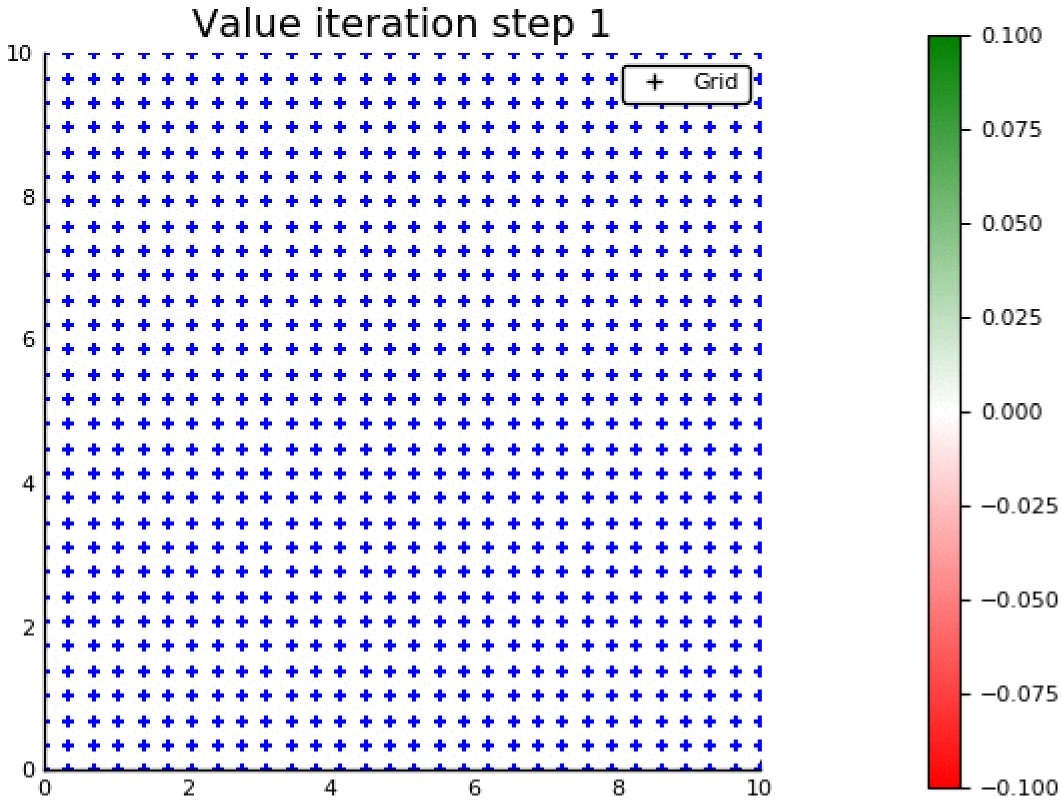
Q: how do we find $V^*(s)$?

- 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 $V_0(s)=0$. Then, update

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

Value Iteration: Demo



Source: POMDPBGallery Julia Package

Break & Quiz

Q 2.1 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. Let π: π(A) = π(B) = **move** (i.e., an "always move" policy). What is the value function $V^{\pi}(A)$?

- A. 0
- B. 1 / (1 -γ)
- C. 1 / (1 -γ^2)
- D. 1

Q-Learning

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

- 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

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

Learning rate

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

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.

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

Q-Learning: SARSA

Using the epsilon-greedy policy, an alternative:

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

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

Learning rate

- Called state–action–reward–state–action (**SARSA**)

Break & Quiz

Q 3.1 What is the main difficulty in applying Q-learning with a discrete Q-table to an environment with continuous numerical observations?

- A. We may not converge to the correct Q(s,a) values.
- B. It would take a potentially intractable amount of memory to represent every possible Q(s,a) value in the discrete Q-table.
- C. Q-learning with a discrete Q-table can be applied to these environments with no issues.

Summary

- Reinforcement learning setup
- Mathematica 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