CS 760: Machine Learning
Reinforcement Learning

Fred Sala

University of Wisconsin-Madison

Nov. 30, 2021

Announcements

- **Logistics**:
  - Welcome back!
  - HW8 released Thursday (last HW).
- Class roadmap:

Tues., Nov. 30	RL I
Thurs., Dec. 2	RL II
Tues., Dec. 7	RL III
Thurs., Dec 9	Large Language Models
Tues., Dec 14	Fairness & Ethics

Outline

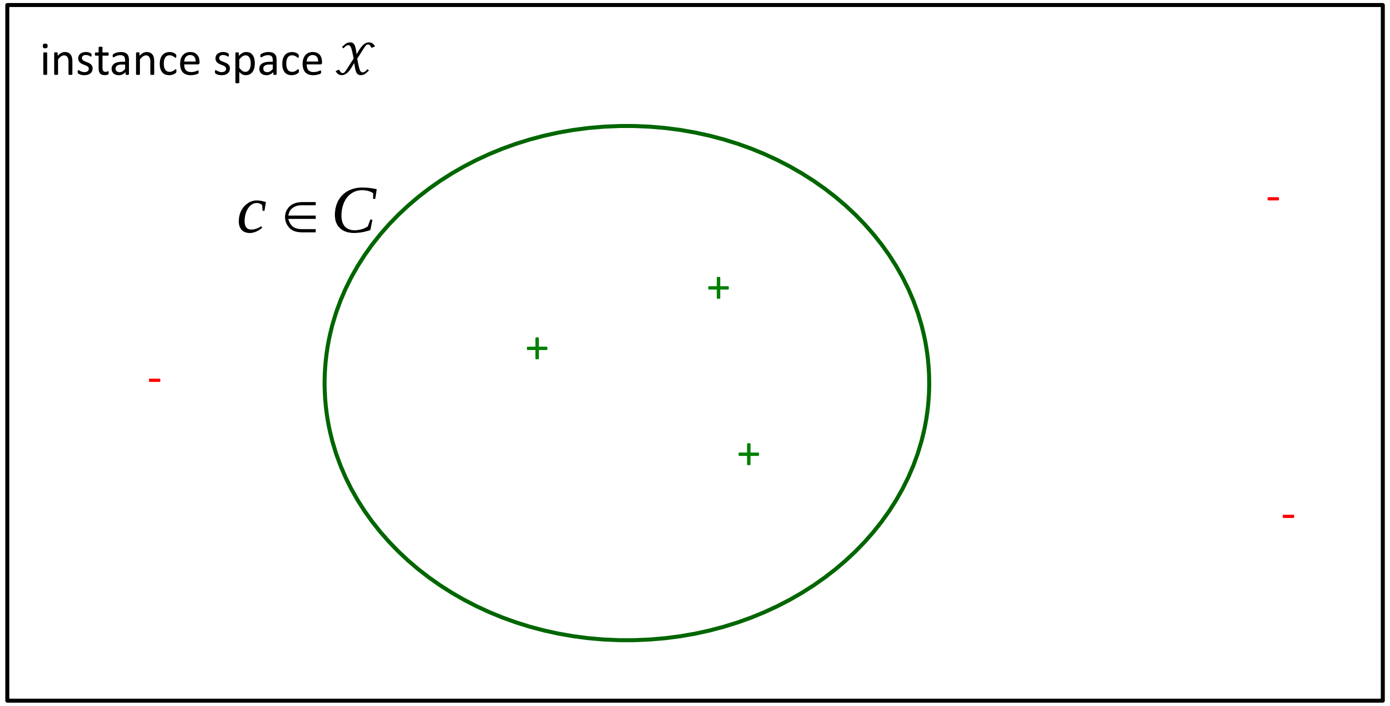
- **Review & PAC Learning Framework**
  - Definition, intuition, sample complexity bounds
- **Intro to Reinforcement Learning**
  - Basic concepts, mathematical formulation, MDPs, policies
- **Valuing and Obtaining Policies**
  - Value functions, Bellman equation, value iteration, policy iteration

Outline

- **Review & PAC Learning Framework**
  - Definition, intuition, sample complexity bounds
- **Intro to Reinforcement Learning**
  - Basic concepts, mathematical formulation, MDPs, policies
- **Valuing and Obtaining Policies**
  - Value functions, Bellman equation, value iteration, policy iteration

PAC Learning Setup

PAC learning is a framework used for theoretical analysis. Basic setting:



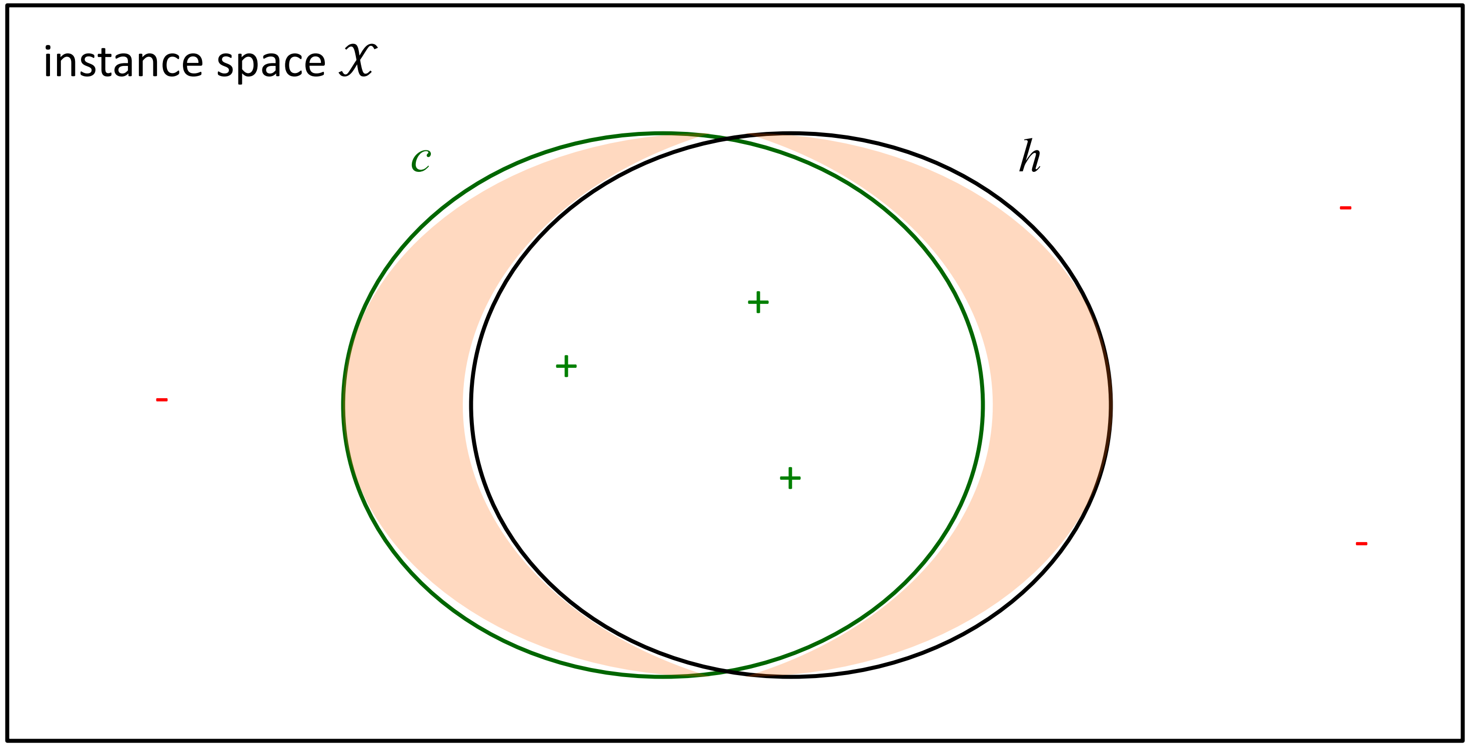
- Set of instances $\mathcal{X}$
- Set of hypotheses (models) H
- Set of possible target concepts C
- Unknown probability distribution $\mathcal{D}$ over instances

PAC Learning Setup

We get a set D of training instances $(\boldsymbol{x}, c(\boldsymbol{x}))$ for some target concept c in C

- each instance $\boldsymbol{x}$ is drawn from distribution $\mathcal{D}$
- class label $c(\boldsymbol{x})$ is provided for each $\boldsymbol{x}$

- learner outputs hypothesis h modeling c
- *Goal:* the *true error* of hypothesis h refers to how often h is wrong on future instances drawn from $\mathcal{D}$



PAC Learning: Two Error Types

We have **two** kinds of errors:

True error: (i.e., on any instance from distribution d):

$$error_{\mathcal{D}}(h) \equiv P_{\mathcal{D}}[c(x) \neq h(x)]$$

Empirical error: (I.e., on our dataset)

$$error_{D}(h) \equiv P_{x \in D}[c(x) \neq h(x)] = \frac{\sum_{x \in D} \delta(c(x) \neq h(x))}{|D|}$$

Goal: Can we bound $error_{\mathcal{D}}(h)$ in terms of $error_{\mathrm{D}}(h)$?

PAC Learning Definition

Consider a class C of possible target concepts defined over a set of instances $\mathcal{X}$ of length n, and a learner L using hypothesis space H

- C is **PAC learnable** by L using H if, for all $c \in C$, distributions $\mathcal{D}$ over $\mathcal{X}$, ε such that $0 < \varepsilon < 0.5$, δ such that $0 < \delta < 0.5$,
- The learner L will, with probability at least $(1-\delta)$, output a hypothesis $h \in H$ such that $error_{\mathcal{D}}(h) \leq \varepsilon$ in time that is polynomial in the quantities:

$$1/\varepsilon\ ,\ 1/\delta,\ n,\ \text{size}(c)$$

"**Probably** **Approximately** Correct"

PAC Learning Applications

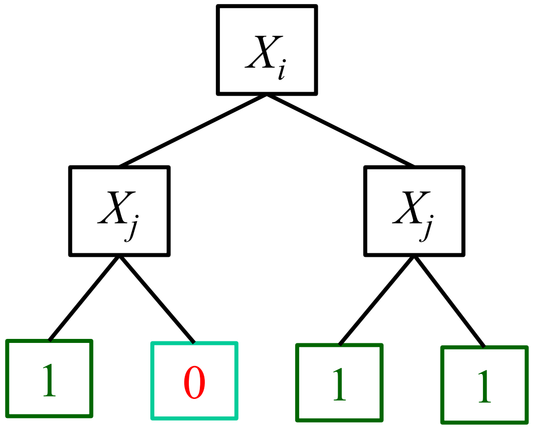
For finite hypothesis classes, the sample complexity (i.e., the m) so that we get a learner that satisfies the above definition is

$$m \geq \frac{1}{e}\left(\ln|H| + \ln\left(\frac{1}{d}\right)\right)$$

Error tolerance

Size of hypothesis class

Probably correct



Can apply to, for example, decision trees of depth 2 for binary feature vectors

- |H| is the number of splits (ie, n choose 2 times 16: # split choices times # leaf labelings)
- For probability ≥ 0.99 with error ≤ 0.05, number of samples we need is:
- Example: for n=100, $m \geq 318$

$$m \geq \frac{1}{.05}\left(\ln\left(8n^2 - 8n\right) + \ln\left(\frac{1}{.01}\right)\right)$$

PAC Learning Discussion

PAC formalizes learning task, allows for non-perfect learning (indicated by ε and δ)

- Requires polynomial computational time

- PAC analysis has been extended to explore a wide range of cases
  - the target concept not in our hypothesis class
  - infinite hypothesis class (VC-dimension theory)
  - noisy training data
  - learner allowed to ask queries
  - restricted distributions (e.g. uniform) over $\mathcal{D}$

- Most analyses are worst case
- Sample complexity bounds are generally not tight

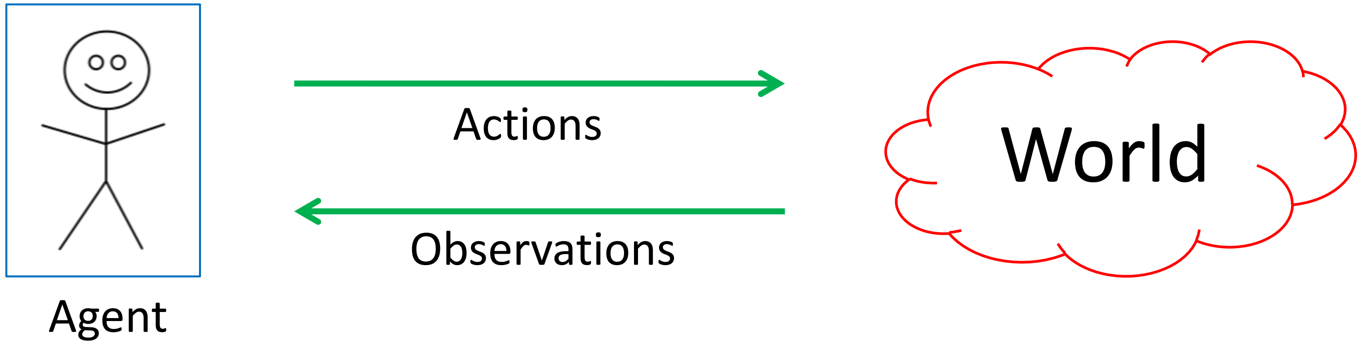
Break & Quiz

Outline

- **Review & PAC Learning Framework**
  - Definition, intuition, sample complexity bounds
- **Intro to Reinforcement Learning**
  - Basic concepts, mathematical formulation, MDPs, policies
- **Valuing and Obtaining Policies**
  - Value functions, Bellman equation, value iteration, policy iteration

A General Model

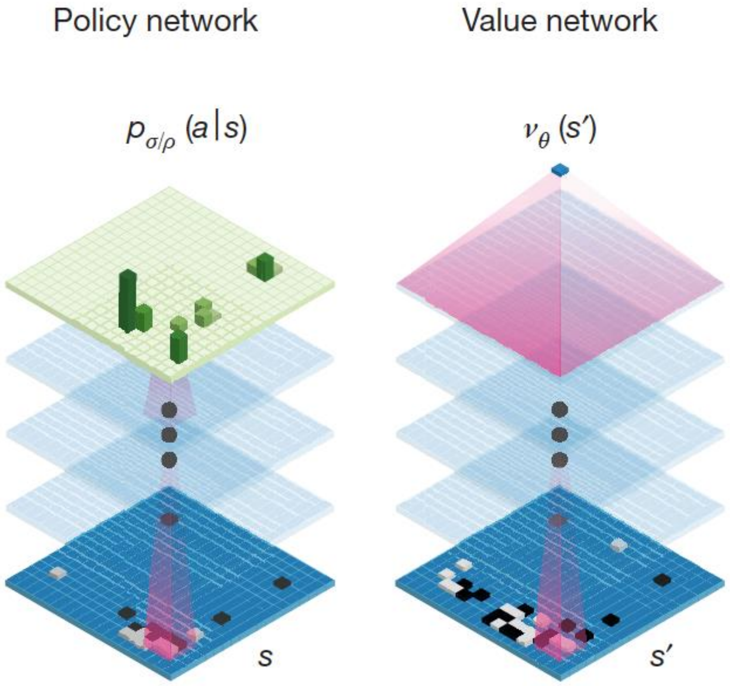
We have an **agent** **interacting** with the **world**



- 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

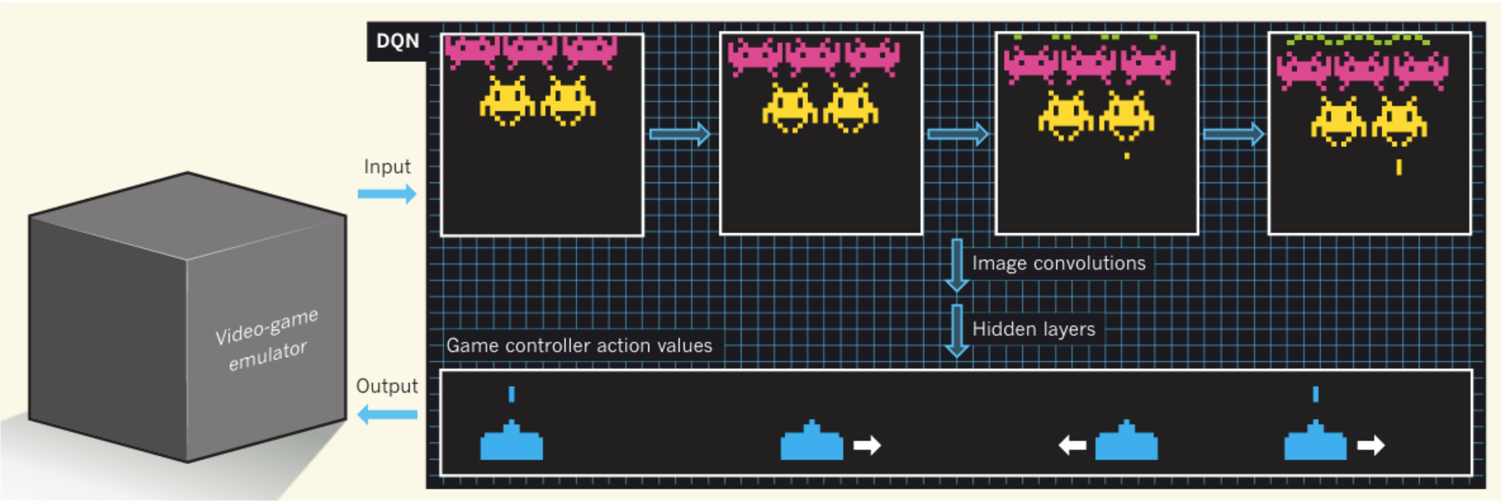
AlphaZero:

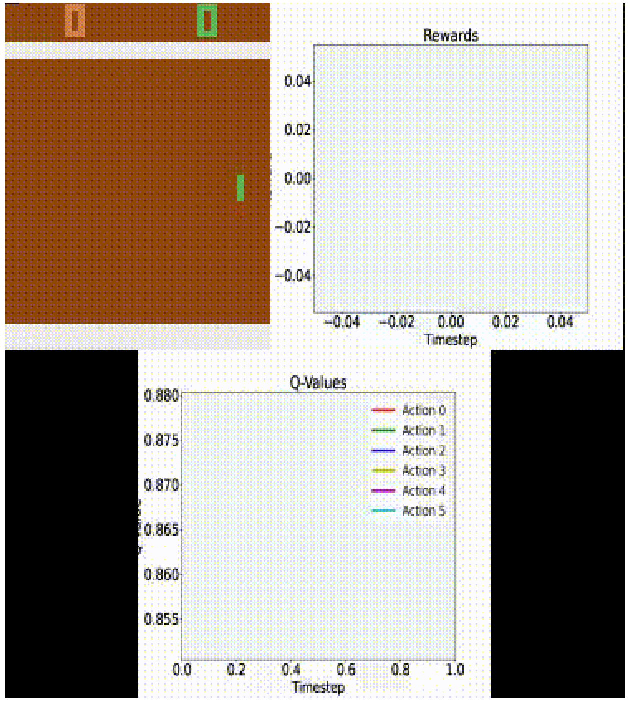


https://deepmind.com/research/alphago/

Examples: Video Game Agents

Pong, Atari



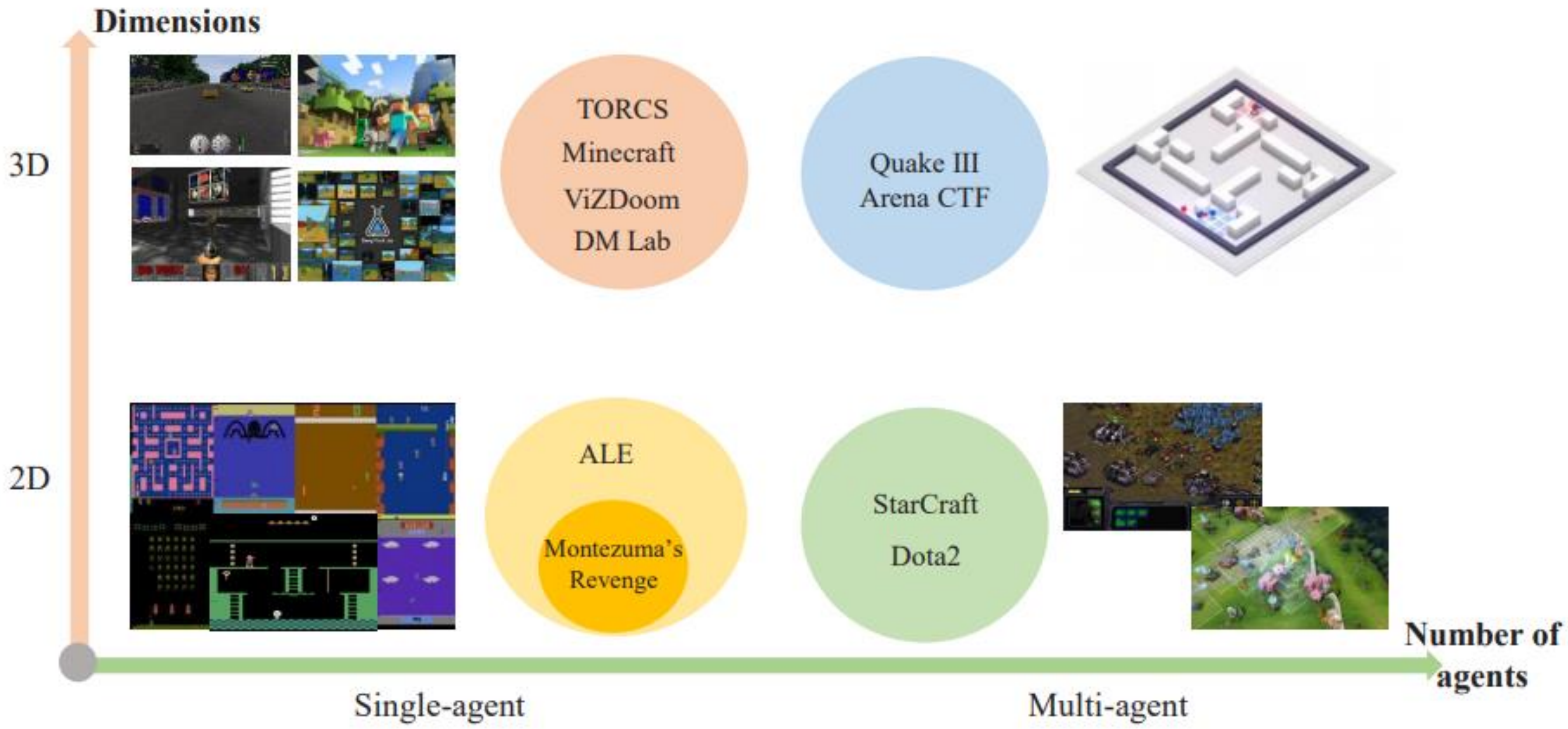


Mnih et al, “Human-level control through deep reinforcement learning”

A. Nielsen

Examples: Video Game Agents

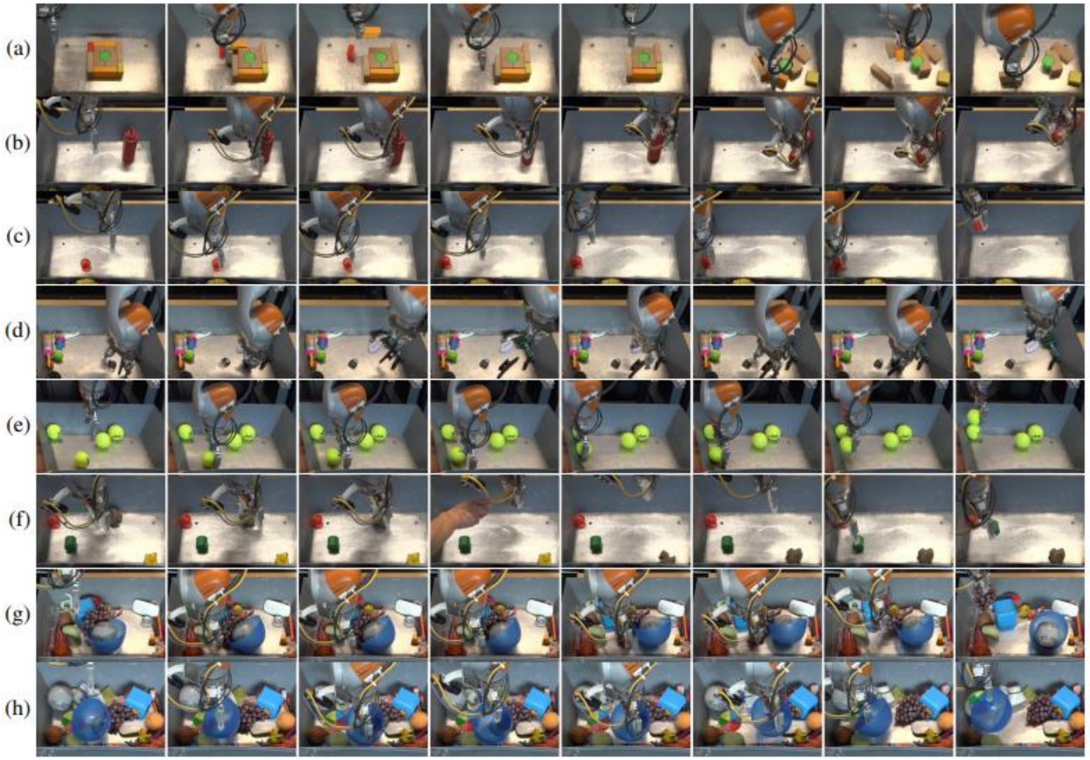
Minecraft, Quake, StarCraft, and more!



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

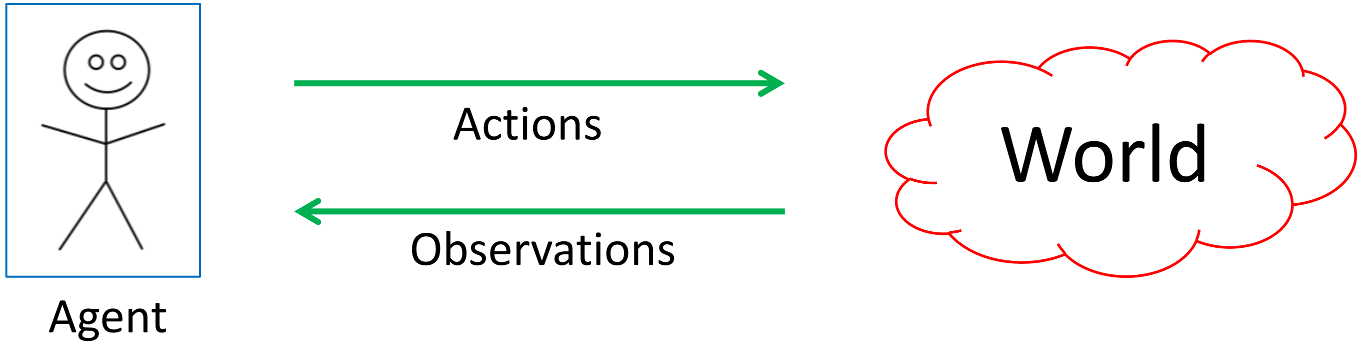
Examples: Robotics

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)

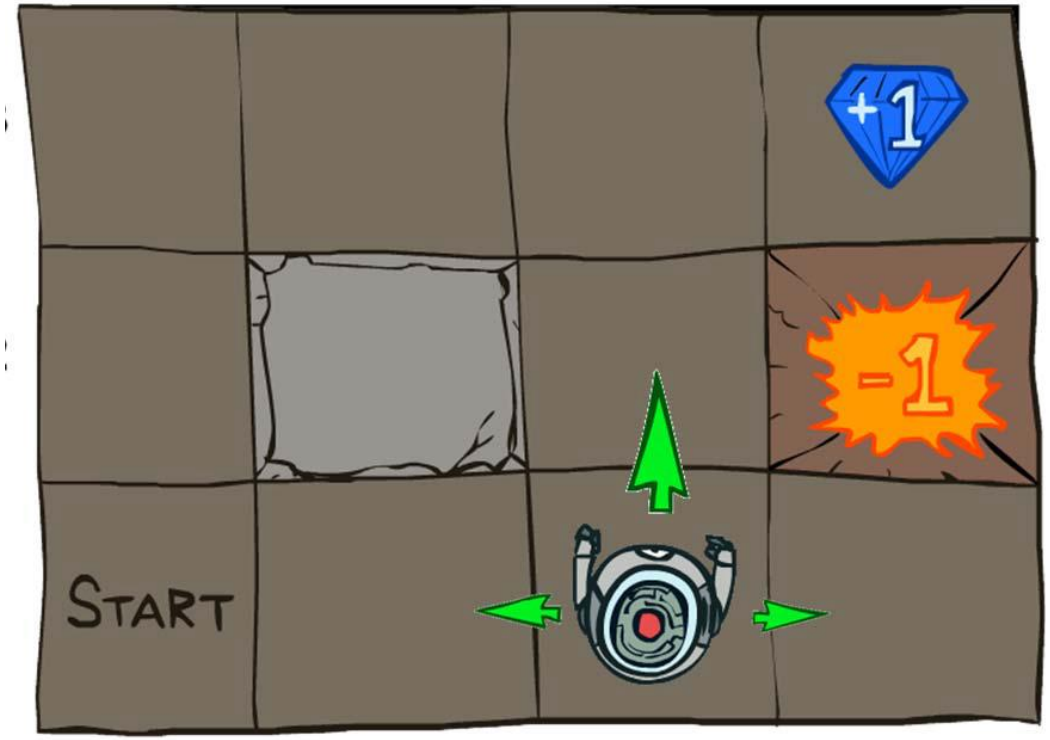
The formal mathematical model:

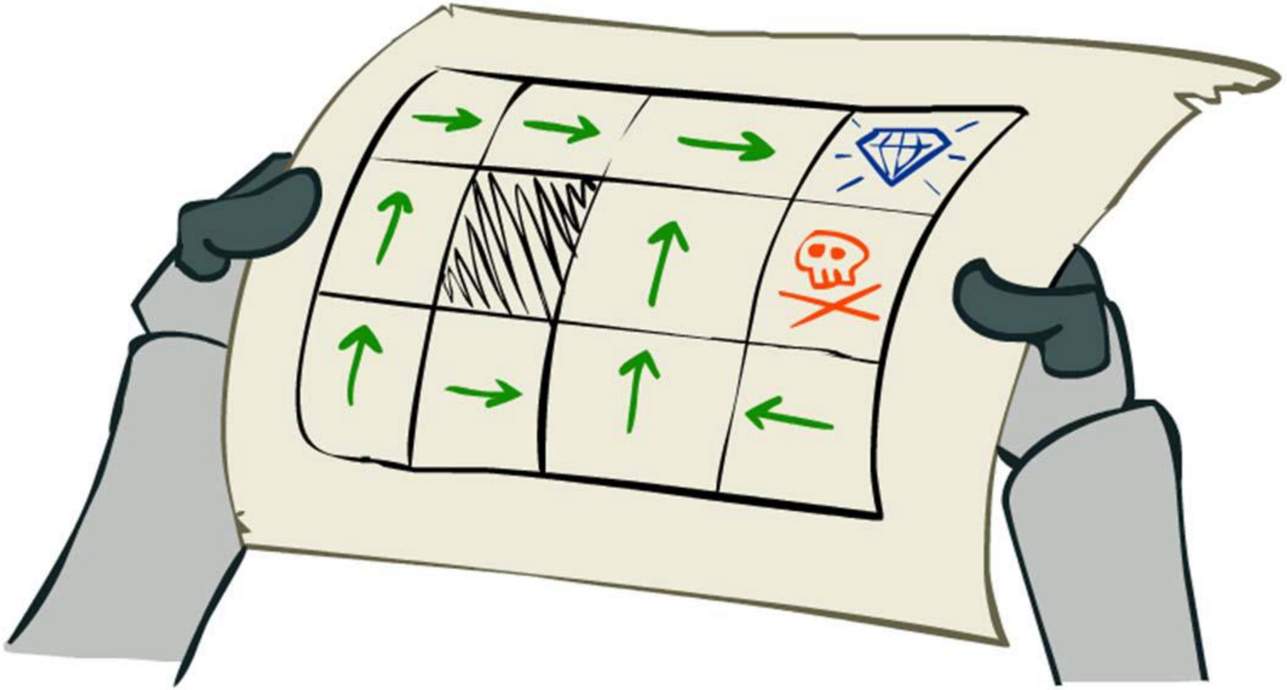
- **State set** S. Initial state s_0. **Action set** A
- **State transition model**: $P(s_{t+1}|s_t, a_t)$
  - Markov assumption: transition probability only depends on s_t and a_t, and not previous actions or states.
- **Reward function:** $r(s_t)$
- **Policy**: $\pi(s) : S \rightarrow A$ action to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

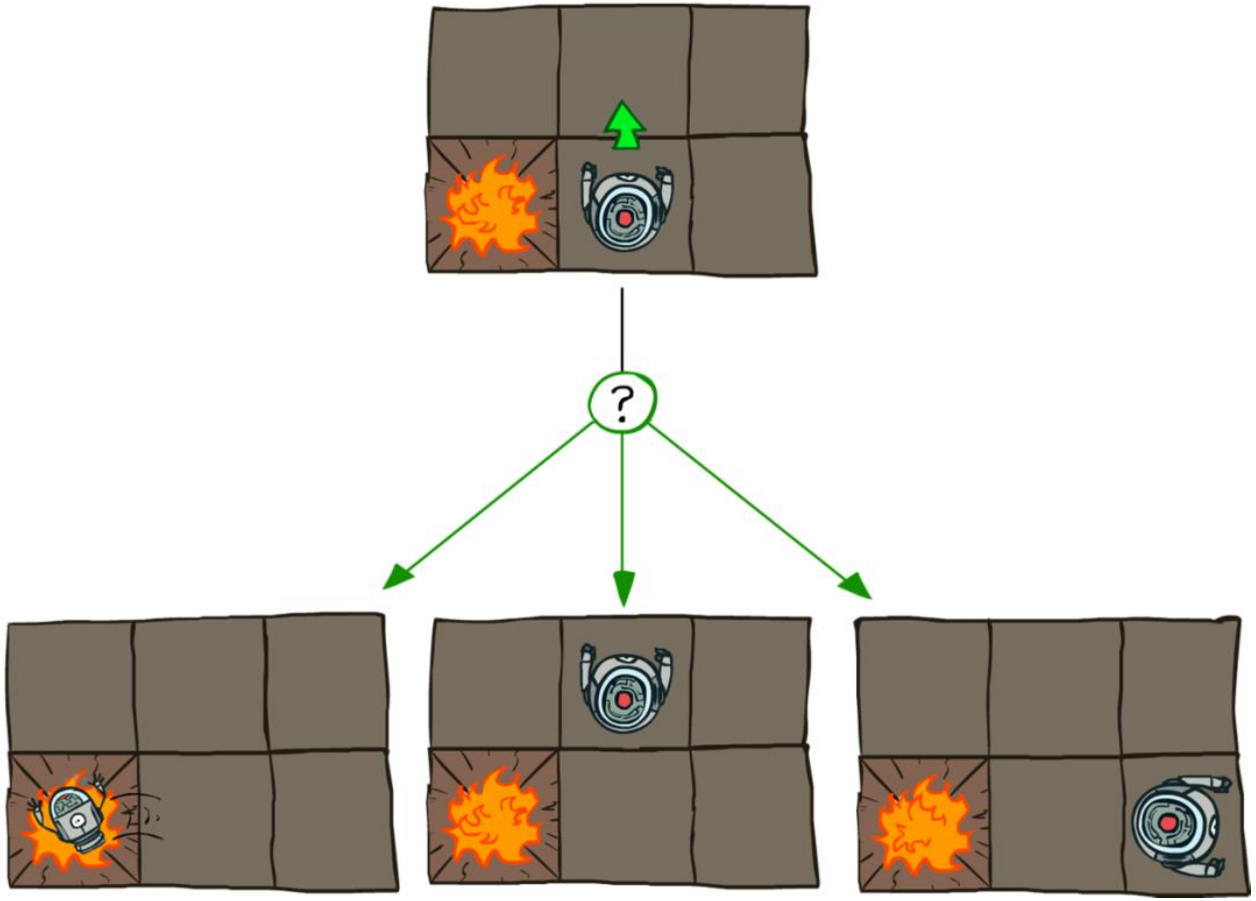


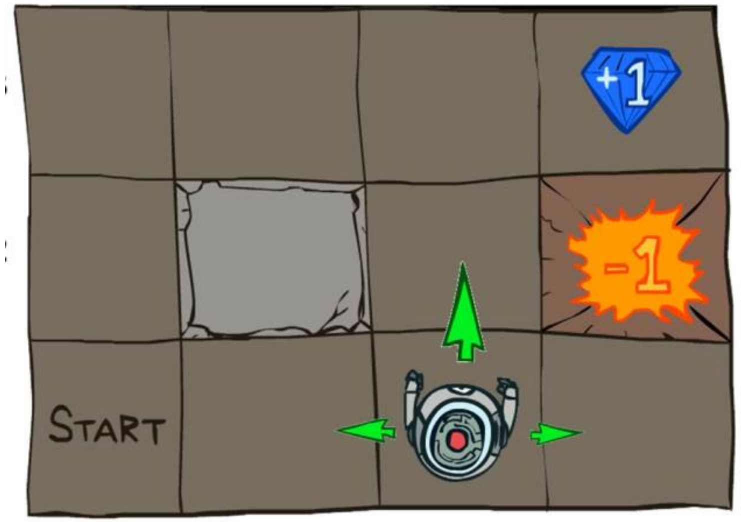


Source: P. Abbeel and D. Klein

Example of MDP: Grid World

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

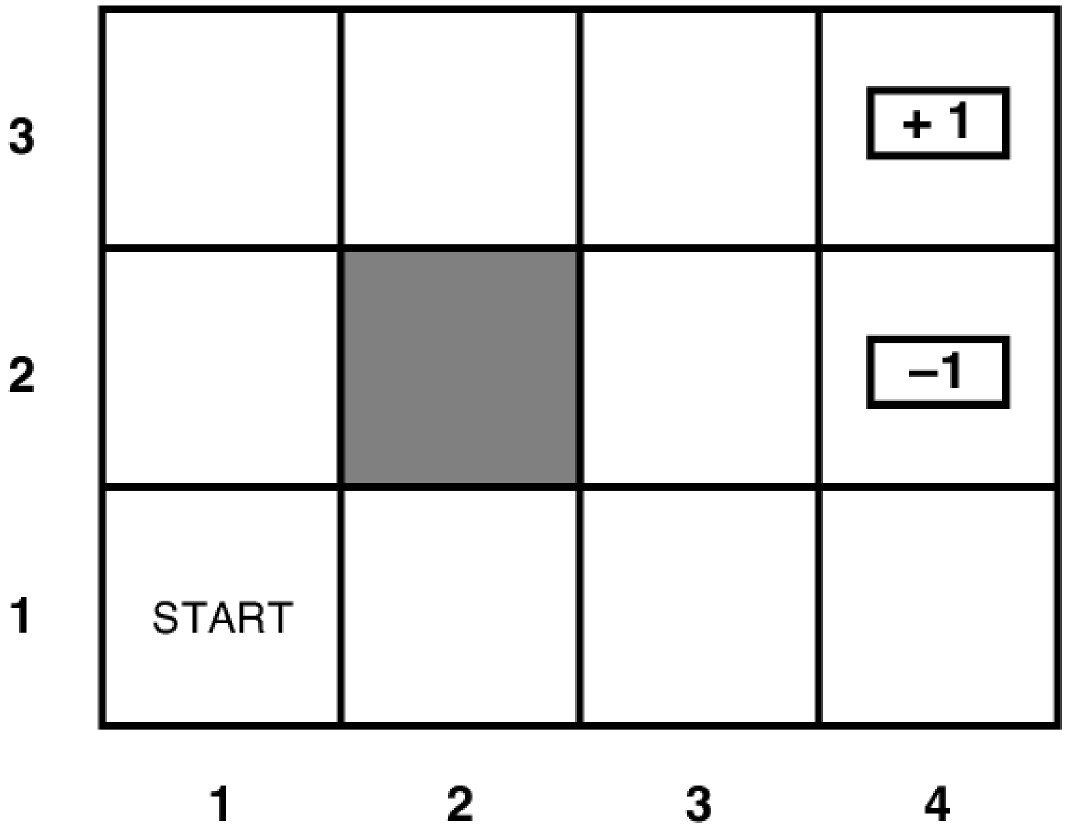


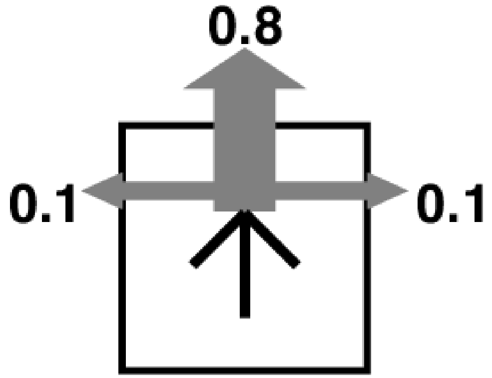


$r(s) = -0.04$ for every non-terminal state

Grid World Abstraction

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

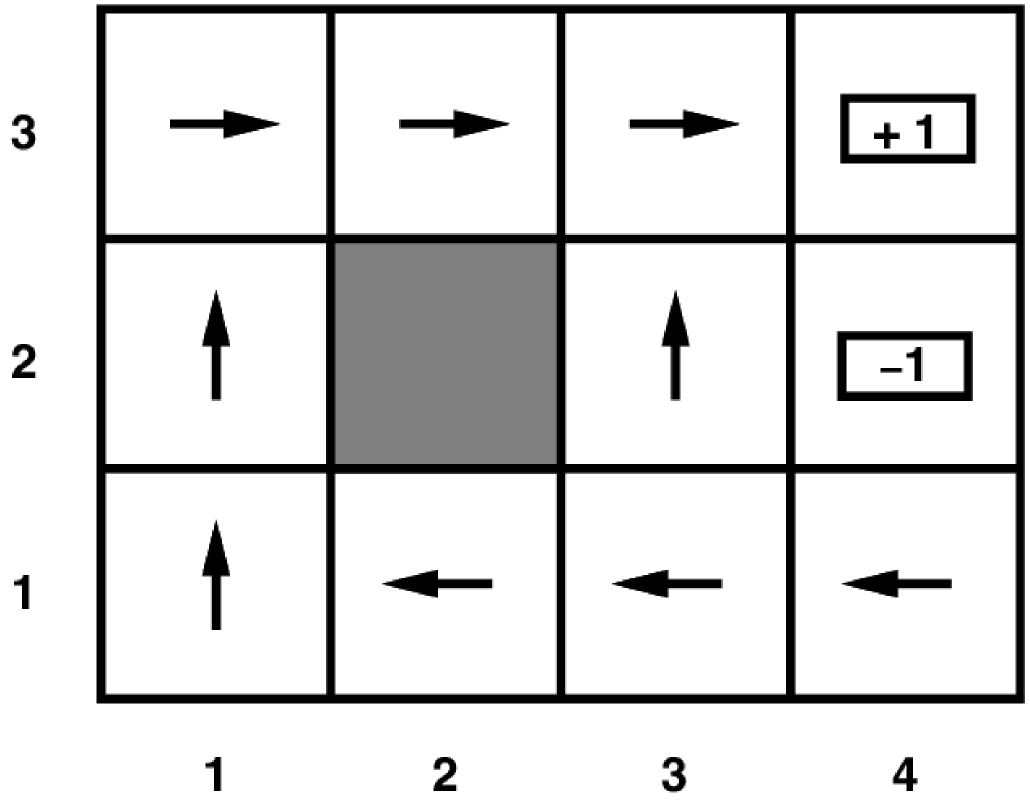


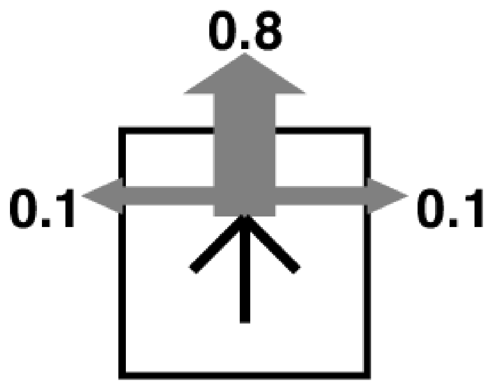


$r(s) = -0.04$ for every non-terminal state

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

Outline

- **Review & PAC Learning Framework**
  - Definition, intuition, sample complexity bounds
- **Intro to Reinforcement Learning**
  - Basic concepts, mathematical formulation, MDPs, policies
- **Valuing and Obtaining Policies**
  - Value functions, Bellman equation, value iteration, policy iteration

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

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.

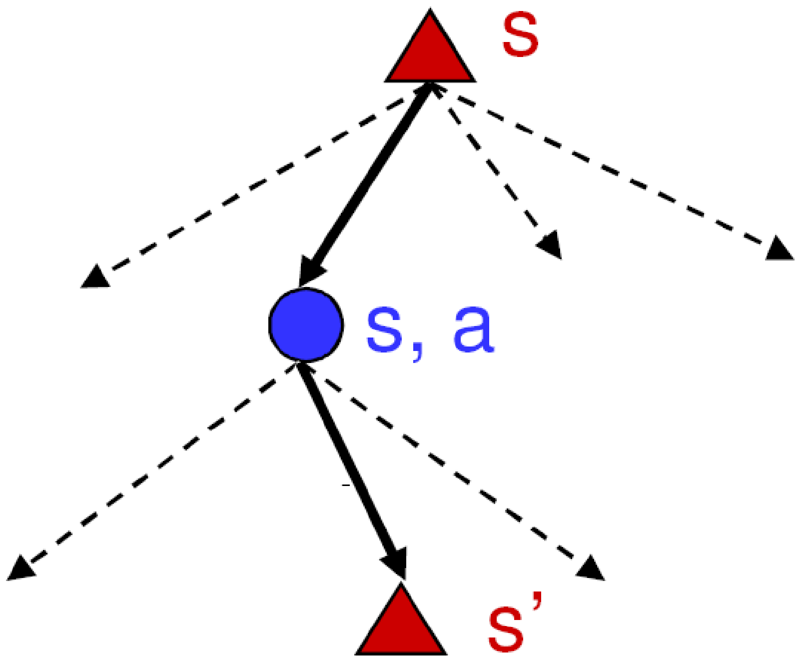
- So, to get the optimal policy, compute

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

All the states we could go to

Transition probability

Expected rewards



Credit L. Lazbenik

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:

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

Current state reward

Discounted expected future **rewards**

- Bellman: inventor of dynamic programming

Value Iteration

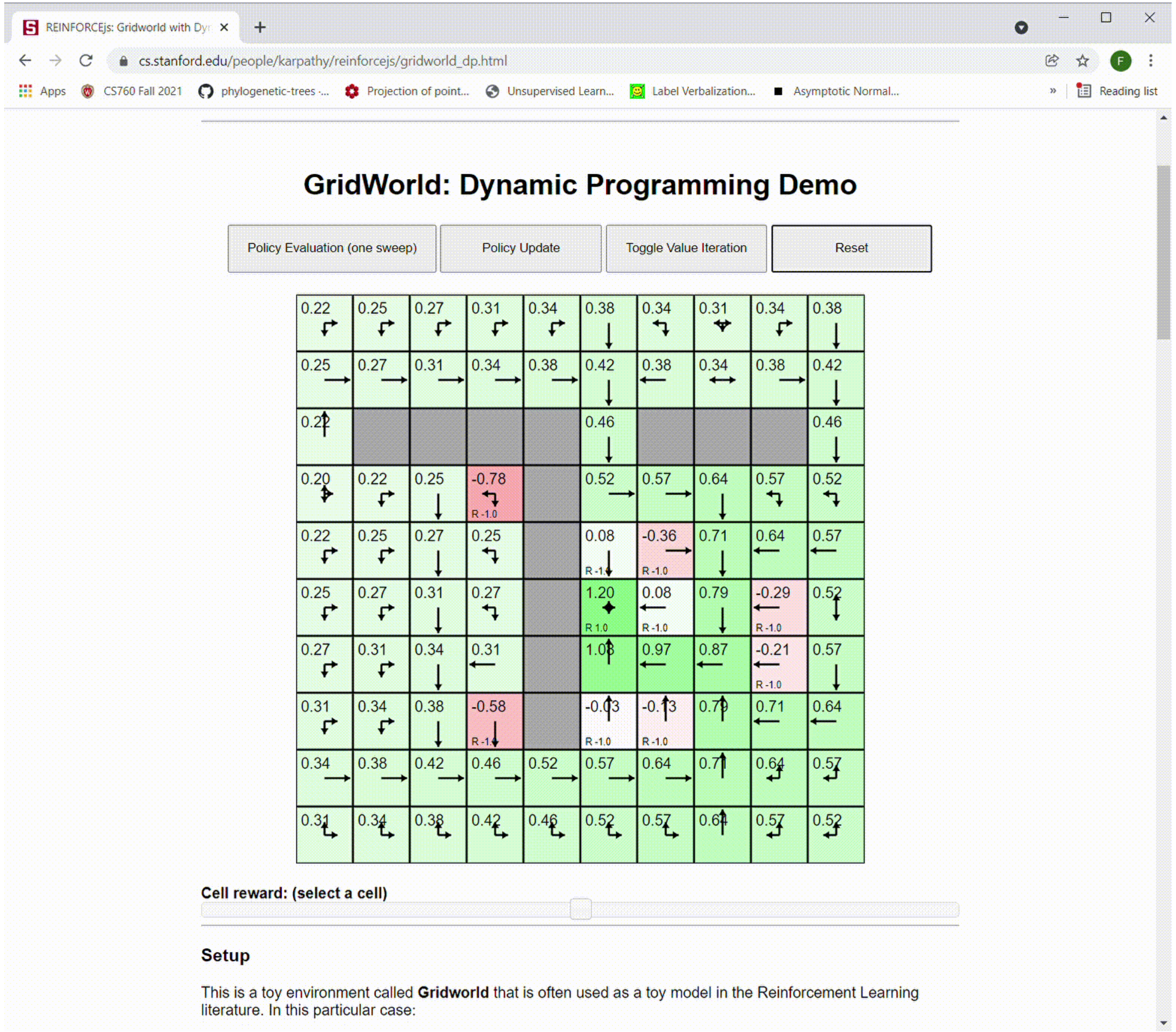
Q: how do we find $V^*(s)$?

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

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

Value Iteration: Demo



Source: Karpathy

Policy Iteration

With value iteration, we estimate V*

- Then get policy (i.e., indirect estimate of policy)

- Could also try to get policies directly

- This is **policy iteration.** Basic idea:
  - Start with random policy π
  - Use it to compute value function V^{π} (for that policy)
  - Improve the policy: obtain π'

Policy Iteration: Algorithm

Policy iteration. Algorithm

- Start with random policy π
- Use it to compute value function V^π : a set of linear equations

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

- Improve the policy: obtain π'

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

- Repeat

Thanks Everyone!

Some of the slides in these lectures have been adapted/borrowed from materials developed by Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Elad Hazan, Tom Dietterich, Pedro Domingos, Jerry Zhu, Yingyu Liang, Volodymyr Kuleshov