CS 760: Machine Learning
Reinforcement Learning I

Kirthi Kandasamy

University of Wisconsin-Madison

Apr 12/17, 2023

Announcements

- **Homework 7** (last HW) is out, due on May 1.

Outline

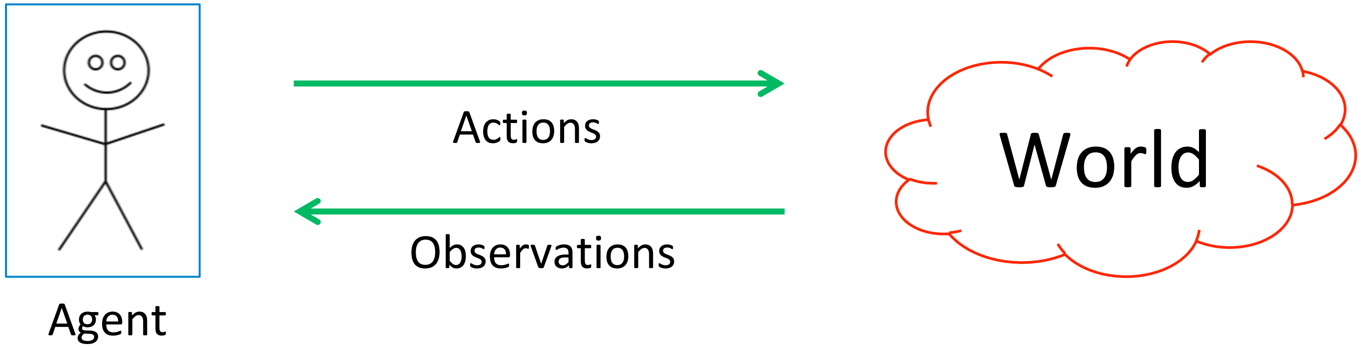
- **Introduction to Reinforcement Learning**
  - Basic concepts, mathematical formulation, MDPs, policies
- **Valuing and Obtaining Policies**
  - Value functions, Bellman equation, value iteration, policy iteration
- **Q Learning**
  - Q function, Q-learning, SARSA, approximation

Outline

- **Introduction to Reinforcement Learning**
  - Basic concepts, mathematical formulation, MDPs, policies
- **Valuing and Obtaining Policies**
  - Value functions, Bellman equation, value iteration, policy iteration
- **Q Learning**
  - Q function, Q-learning, SARSA, approximation

A General Model

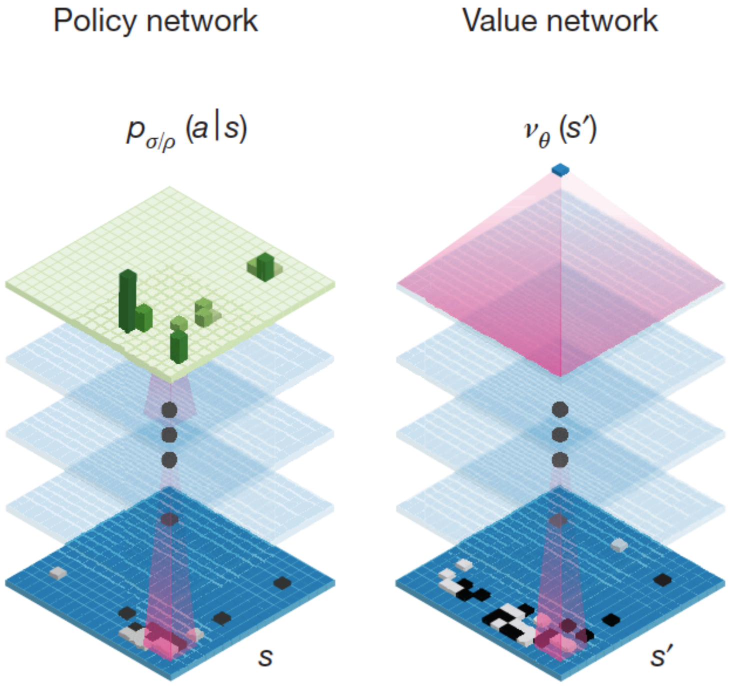
We have an **agent** **interacting** with the **world**



- 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

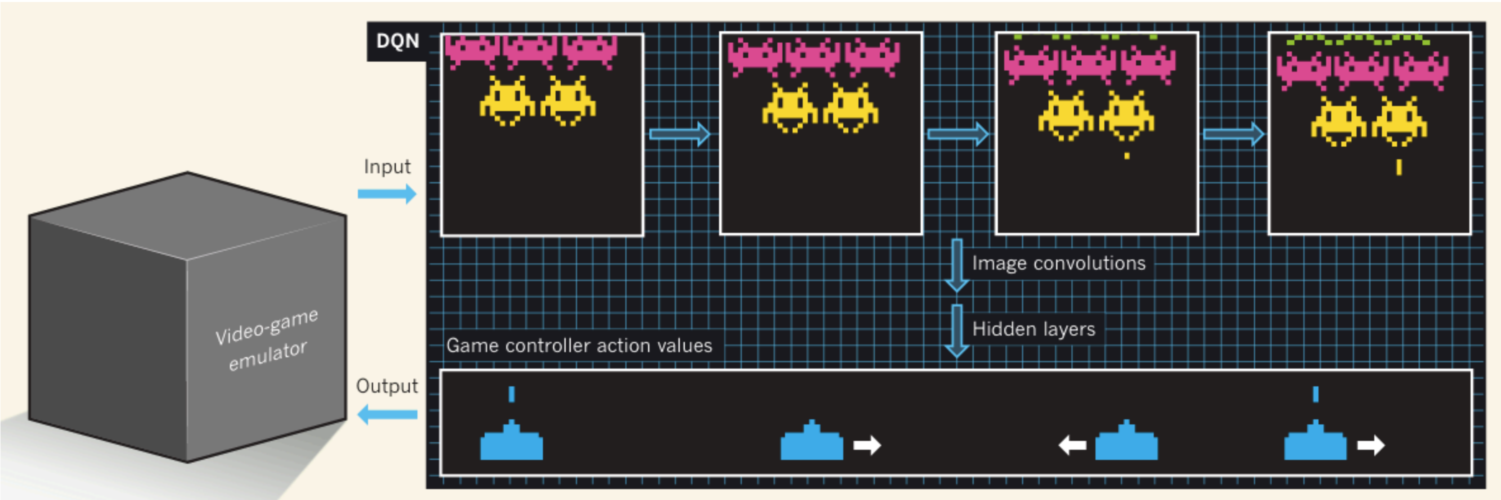
AlphaZero:



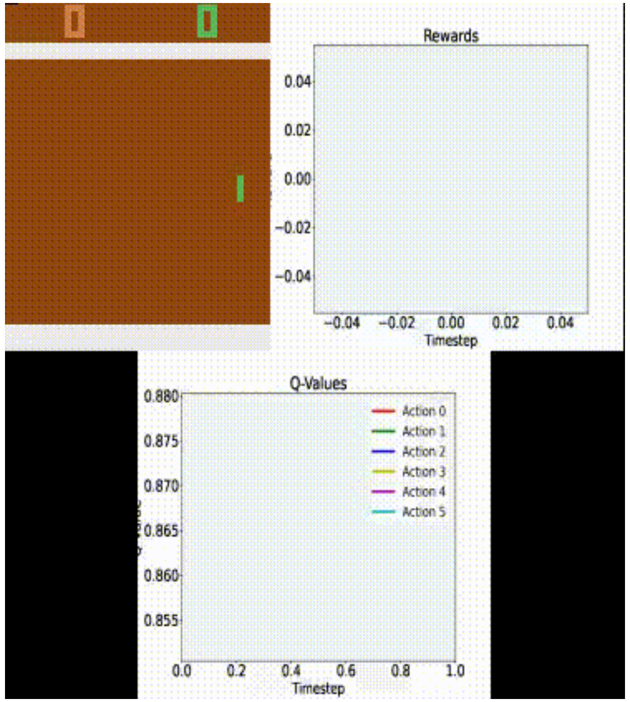
https://deepmind.com/research/alphago/

Examples: Video Game Agents

Pong, Atari



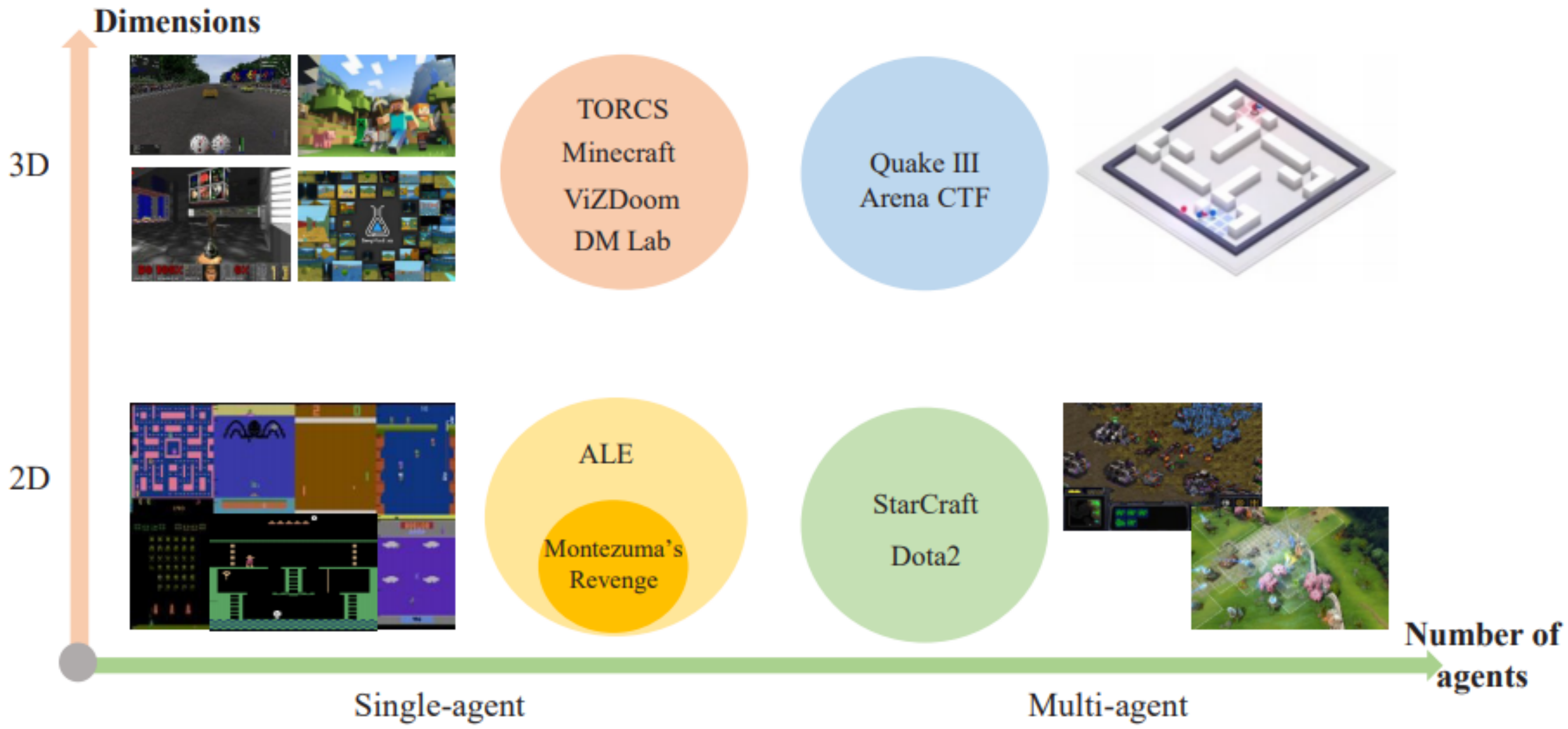
Mnih et al, “Human-level control through deep reinforcement learning”



A. Nielsen

Examples: Video Game Agents

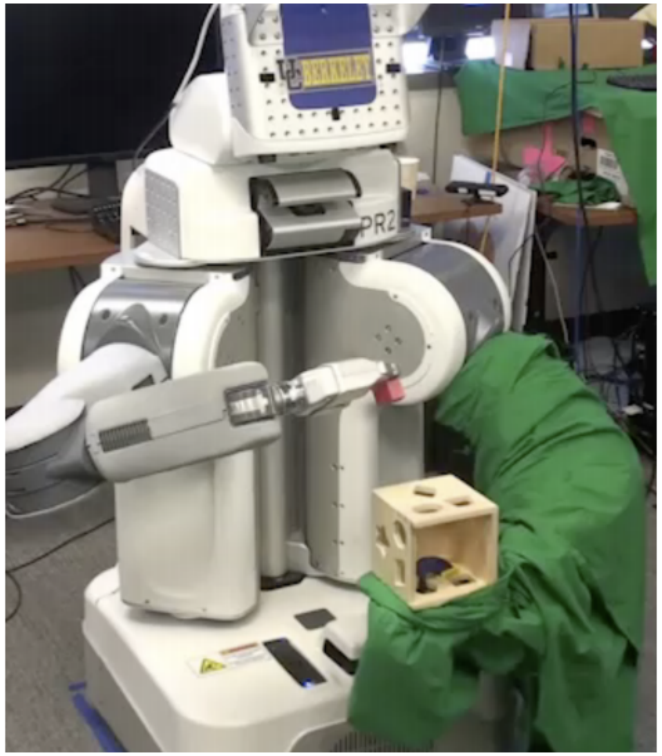
Minecraft, Quake, StarCraft, and more!



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

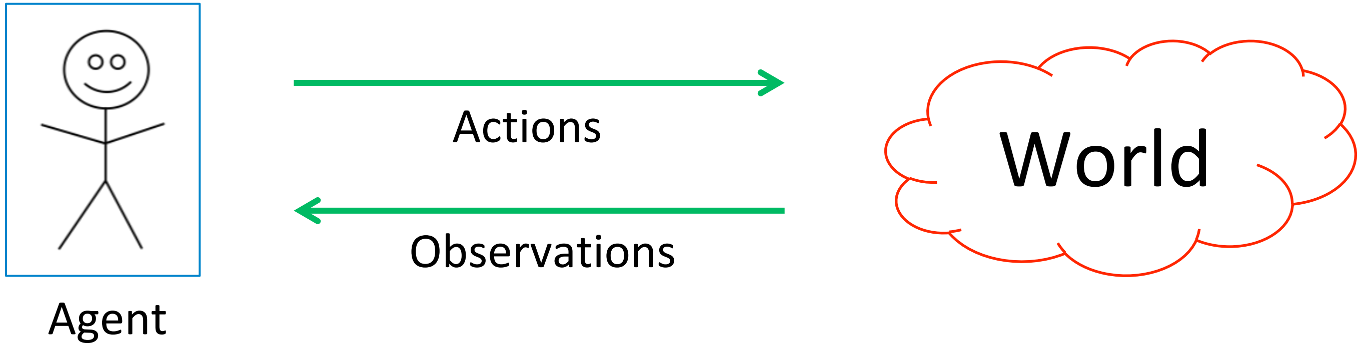
Examples: Robotics

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 a formal 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"

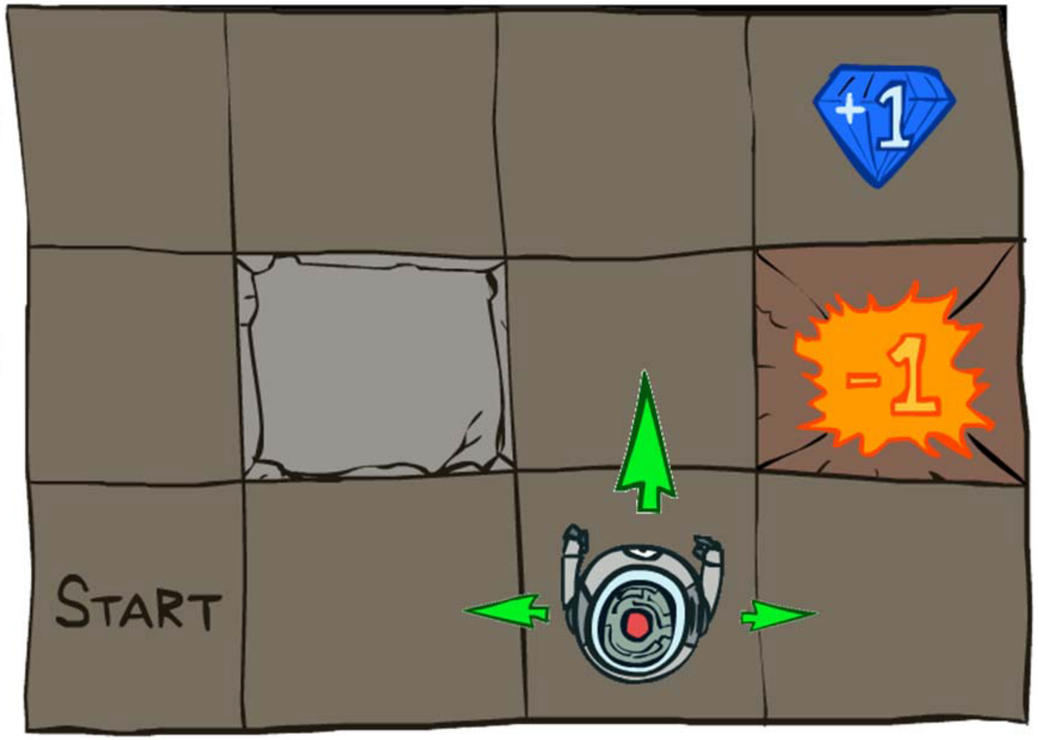
Markov Decision Process (MDP)

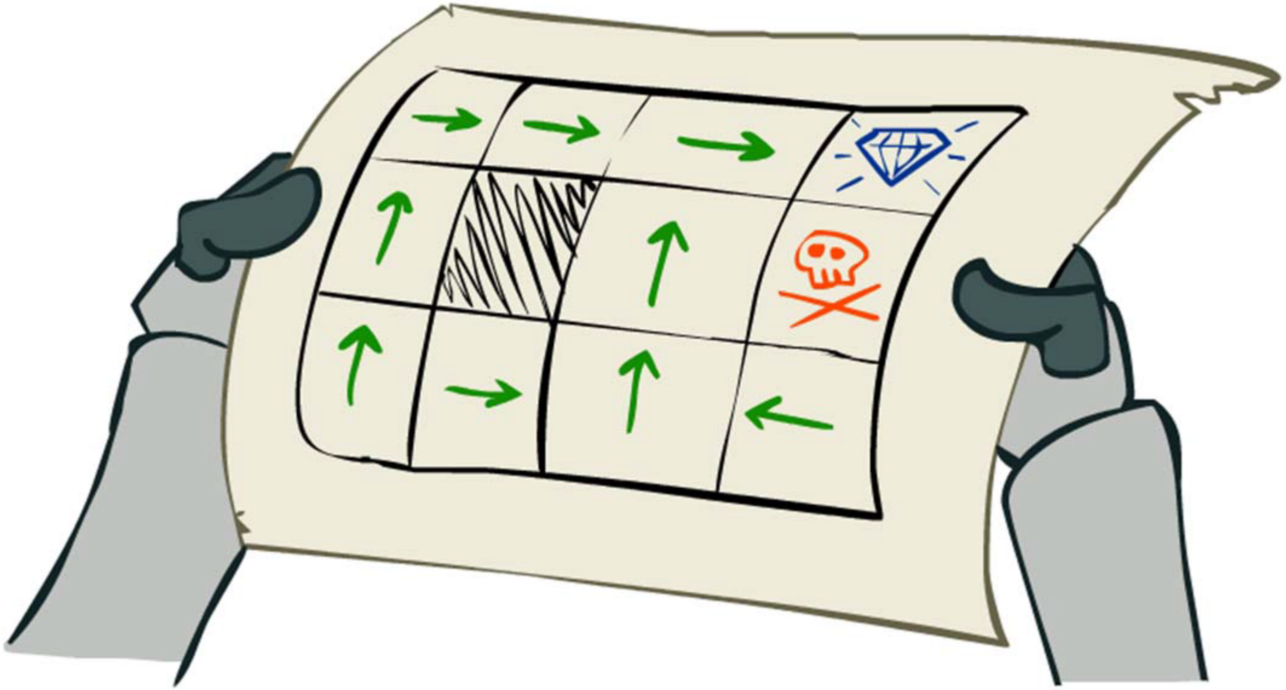
- **State set** S. Initial state s_0. **Action set** A
- **State transition model:** $P(s_{t+1} | s_t, a_t)$
  - Markov assumption: transition probability only depends on s_t and a_t, and not previous actions or states.
  - More generally: $P(r_{t+1}, s_{t+1} \mid s_t, a_t)$
- **Reward function:** $r(s_t)$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

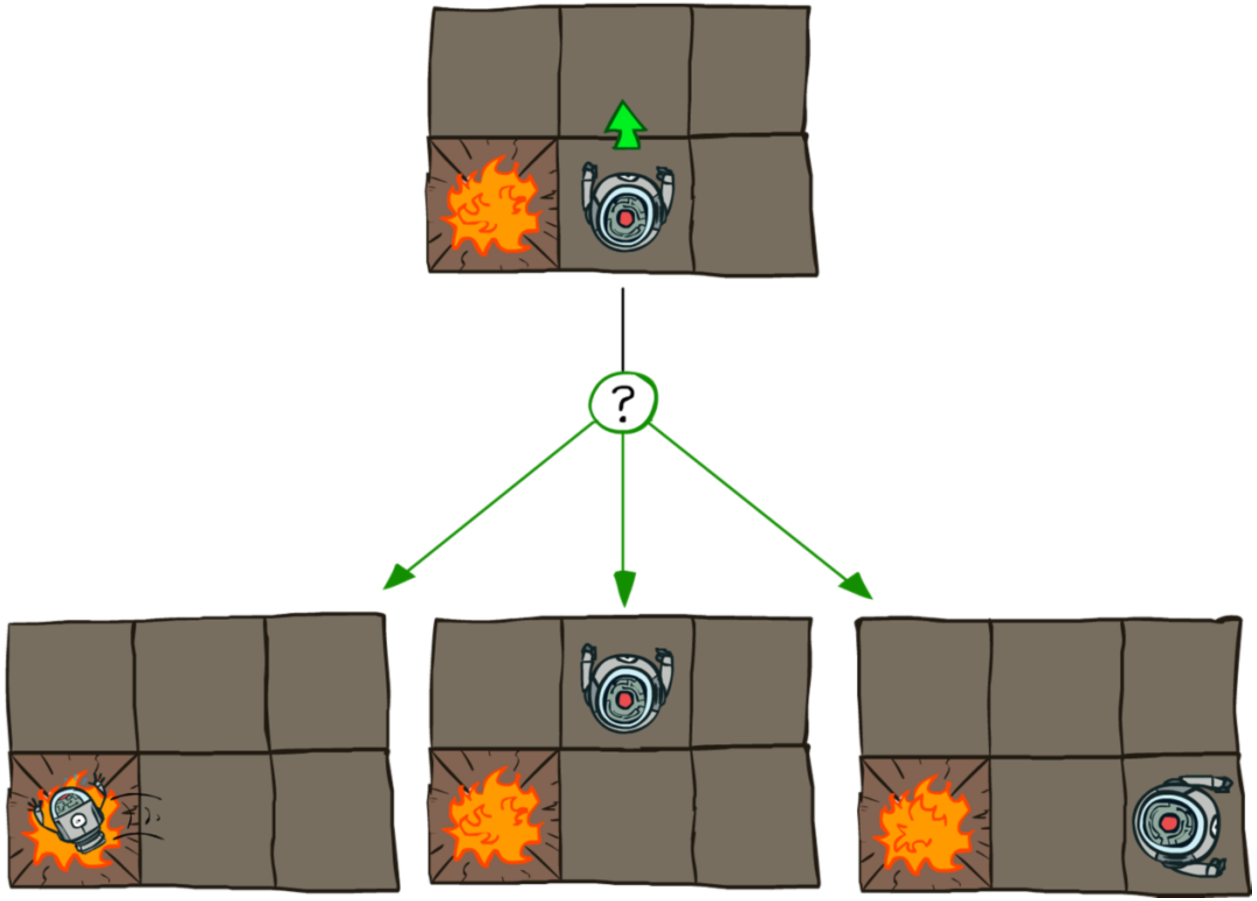


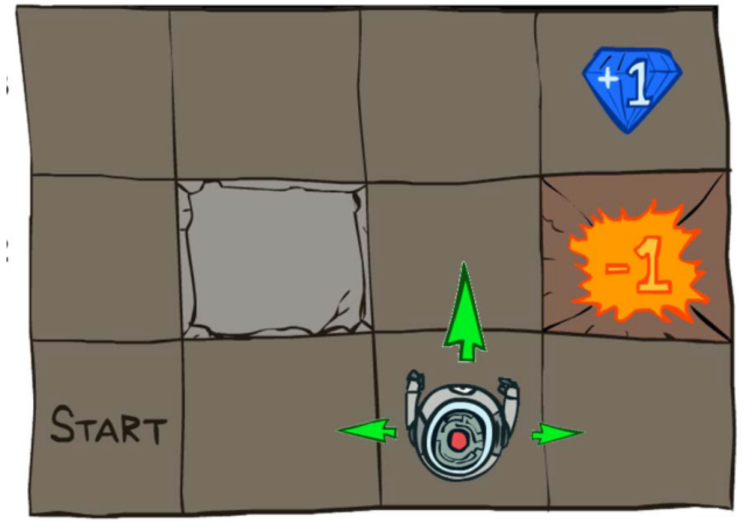


Source: P. Abbeel and D. Klein

Example of MDP: Grid World

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

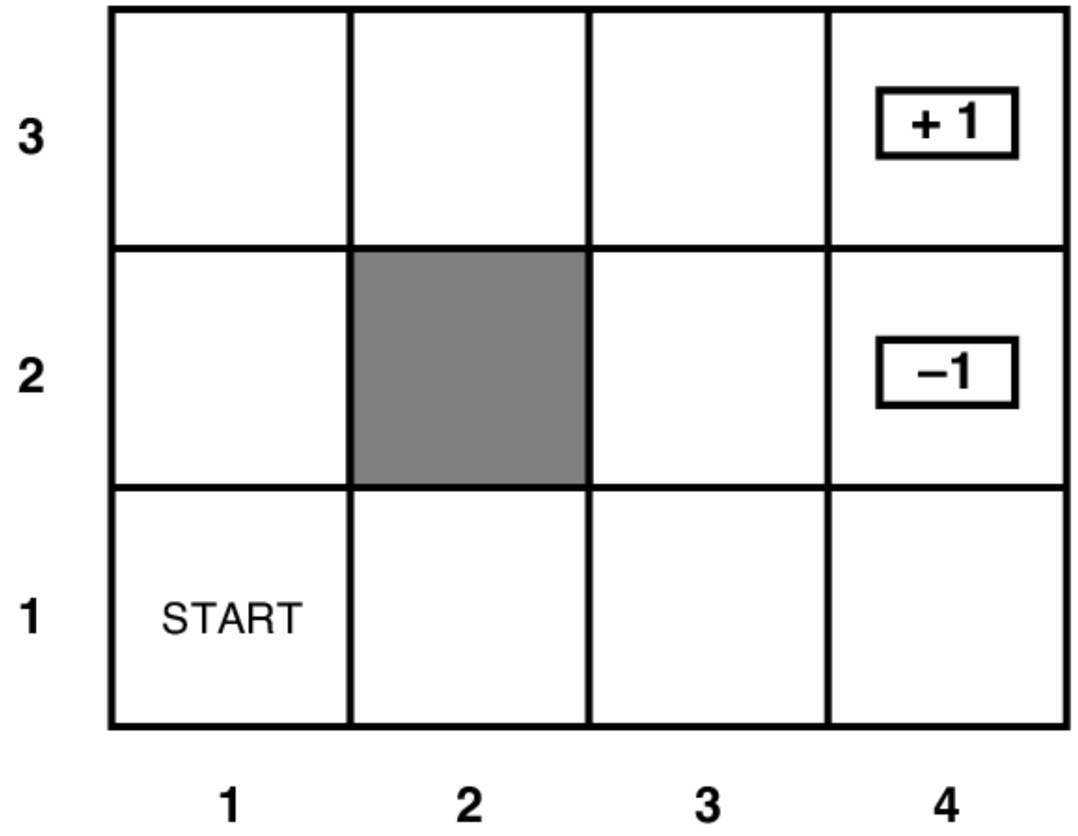


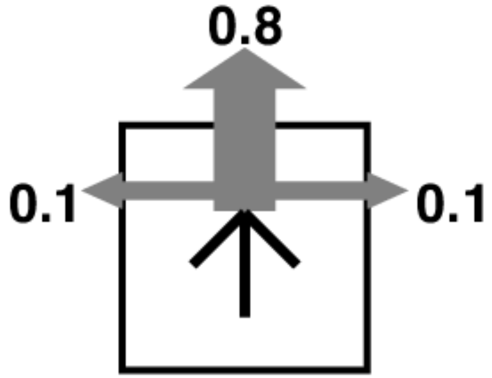


$r(s) = -0.04$ for every non-terminal state

Grid World Abstraction

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

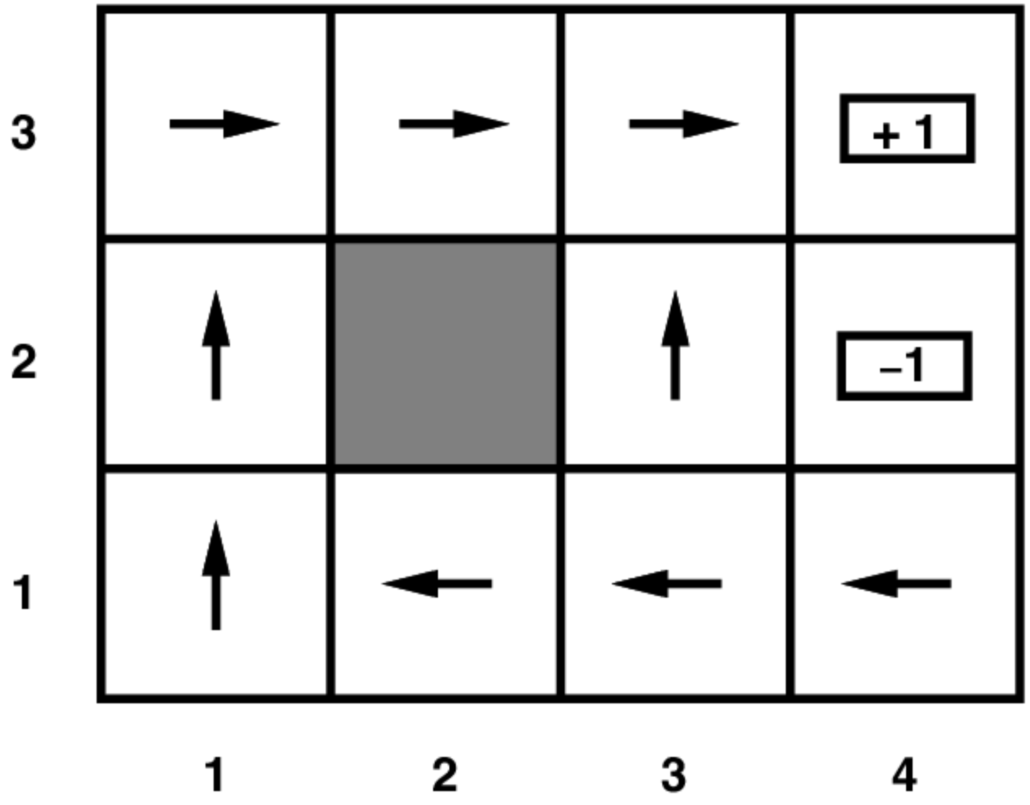


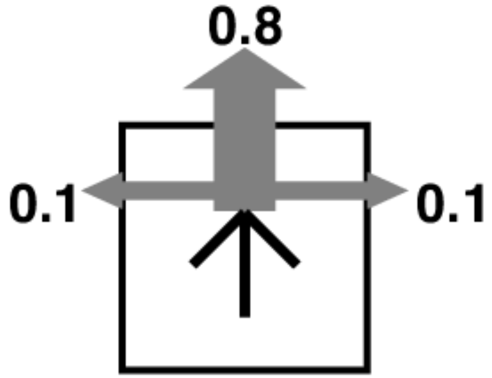


$\boldsymbol{r}(s) = -0.04$ for every non-terminal state

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:** $\pi(s) : S \to A$ 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

Break & Quiz

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

Break & Quiz

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**

Break & Quiz

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

- A. The reward function must output a scalar value (**True**)
- B. The policy maps states to actions (**True: a policy tells you what action to take for each state (Although it could map to a distribution over actions as well)**).
- **C. The probability of next state can depend on current and previous states** (**False: Markov assumption**).

Outline

- **Intro to Reinforcement Learning**
  - Basic concepts, mathematical formulation, MDPs, policies
- **Valuing and Obtaining Policies**
  - Value functions, Bellman equation, value iteration, policy iteration
- **Q Learning**
  - Q function, Q-learning, SARSA, approximation

Values & Policies

For policy π, **the value** starting from produced by following that policy:

$$V^{\pi}(s_0) = \sum_{\substack{\text{sequences } (s_t, a_t, r_t, s_{t+1}) \\ \text{starting from } s_0}} P(\text{sequence})\,U(\text{sequence})$$

U(sequence): sum of rewards when following a sequence

Value: Expected sum of rewards when starting from a state

Called the **value function** (for π)

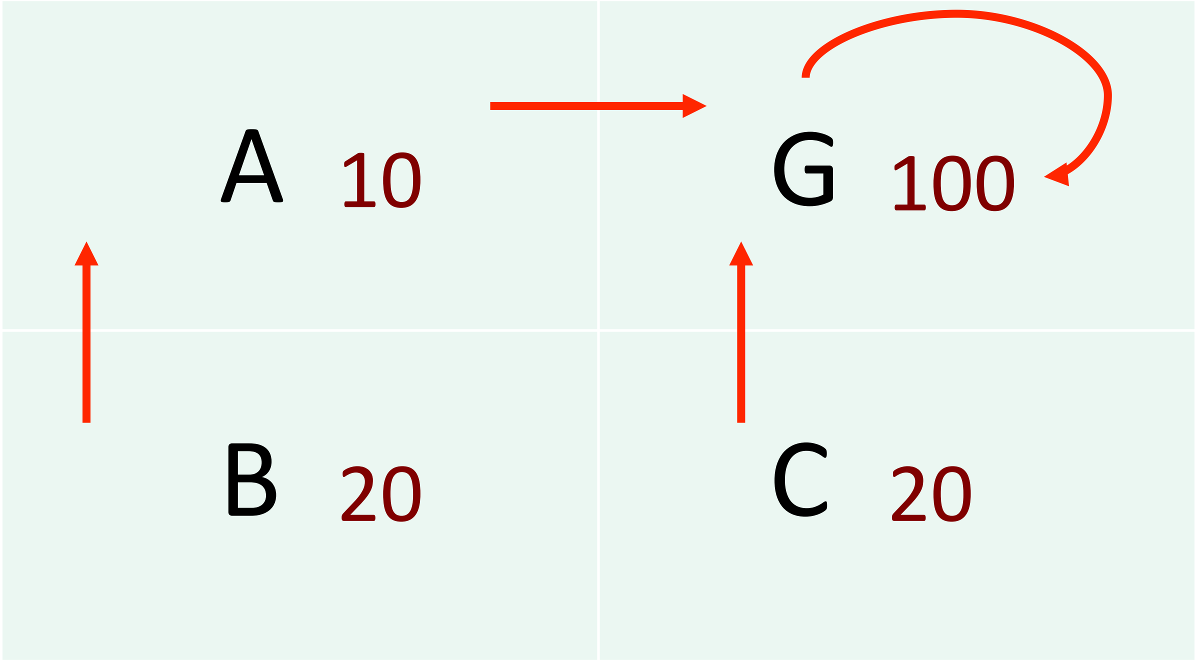
Values & Polices: Discounting Rewards

- If each sequence is finite and the reward at each state is bounded, the value of a policy is also bounded.
- But if it is an infinite series, we usually discount rewards,

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

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

Quiz: Find the value of this policy from all states



Deterministic transitions, γ=0.8, policy shown in red arrow.

Finding the value of a policy: the Bellman Equation

$$\underbrace{V^{\pi}(s) = r(s)}_{\text{Current state reward}} + \underbrace{\gamma \sum_{s'} P(s'|s,\pi(s))V^{\pi}(s')}_{\text{Discounted expected future rewards}}$$

Current state reward

Discounted expected future **rewards**

Proof: (see board)

- Richard Bellman: inventor of dynamic programming

Value Iteration using the Bellman equation

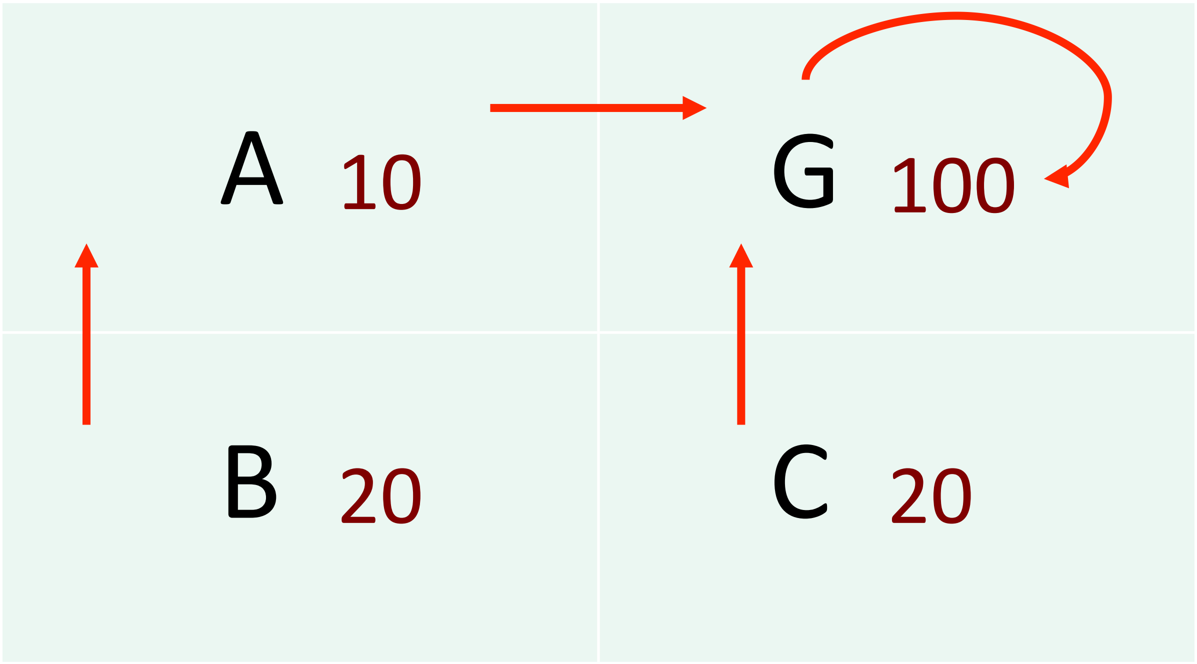
How do we find $V^{\pi}(s)$?

- Know: reward $r(s)$, transition probability $P(s'|s,a)$

Initialize some value function $V_0^{\pi}(s)$ (typically $V_0^{\pi}(s) = 0$).
Then, update

$$V_{i+1}^{\pi}(s) \leftarrow r(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) V_i^{\pi}(s')$$

Quiz: Find value of this policy using the Bellman equation



Deterministic transitions, γ=0.8, policy shown in red arrow.

Obtaining the optimal policy

Now that V^{π} is defined for all policies, how do we define the optimal policy?

- First, set $V^{*}(s)$ to be expected utility for **optimal** policy π^{*} from s. (That is, $V^{*}(s) = V^{\pi^{*}}(s) > V^{\pi}(s)$ for all other policies π.)
- What is the expected utility of a in state s? That is, what is the best you could hope to do, after taking 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.

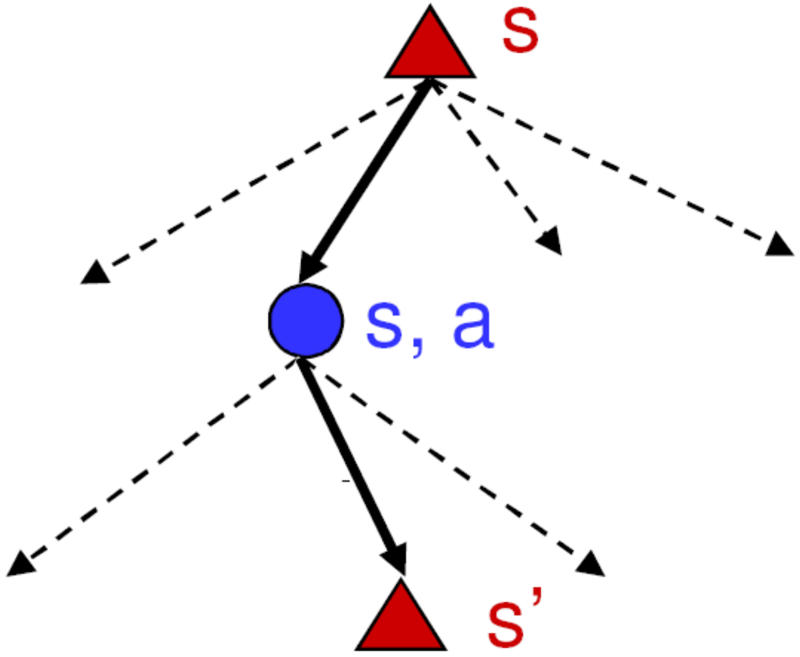
- So, to get the optimal policy, compute

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

All the states we could go to

Transition probability

Expected rewards



Credit L. Lazbenik

Obtaining the optimal policy

Now we can obtain the optimal policy via,

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

- 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 (for the optimal policy)

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

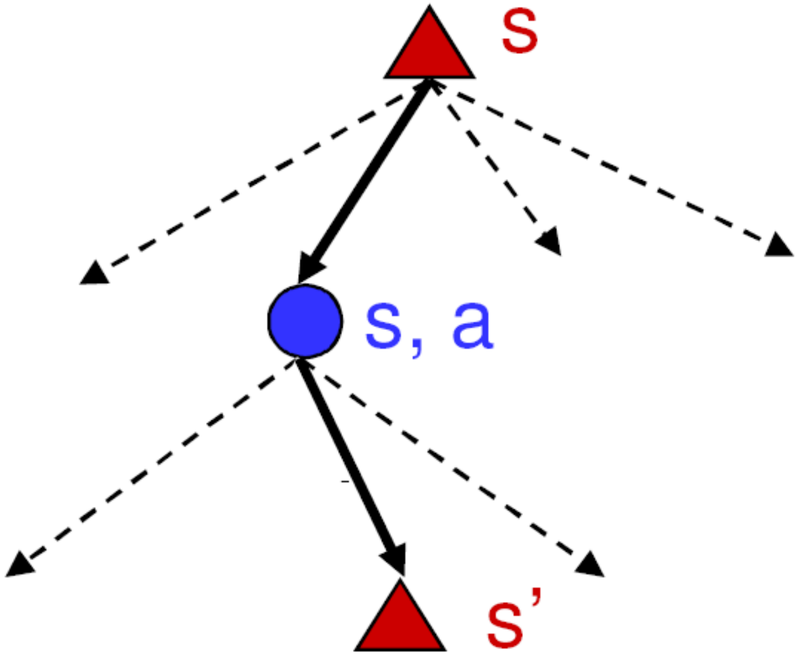
Bellman Equation

Let us walk over one step for the value function:

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

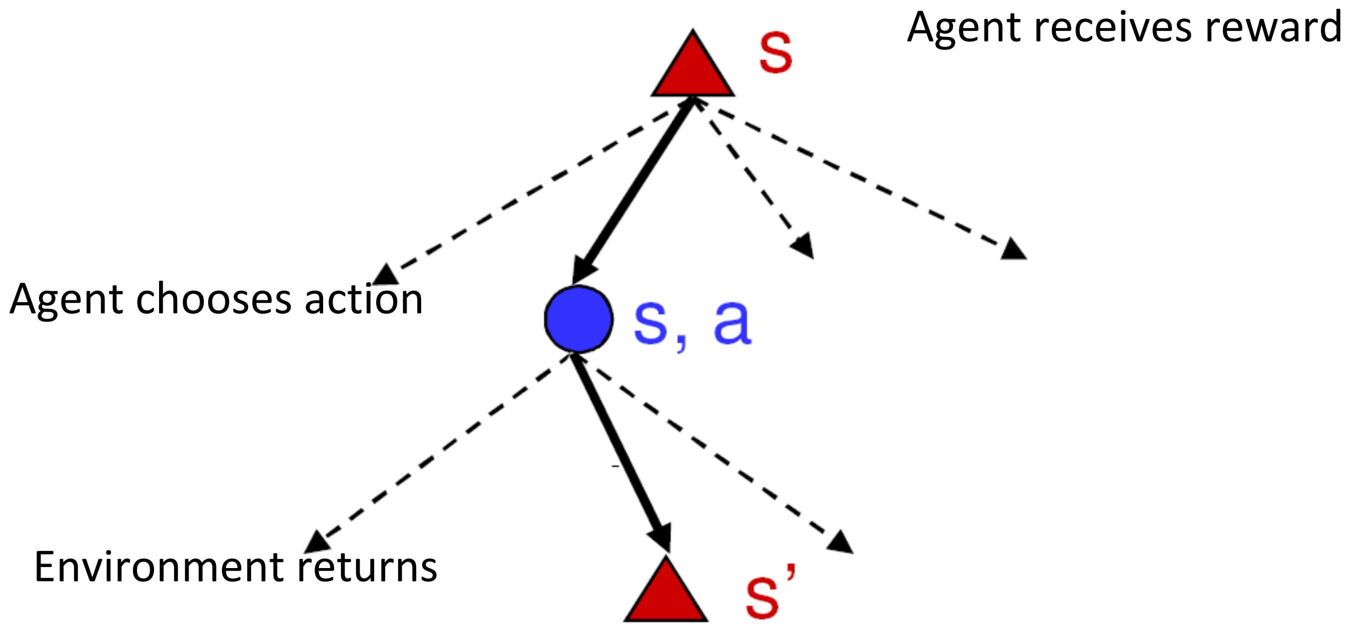
Current state reward

Discounted expected future **rewards**



Credit L. Lazbenik

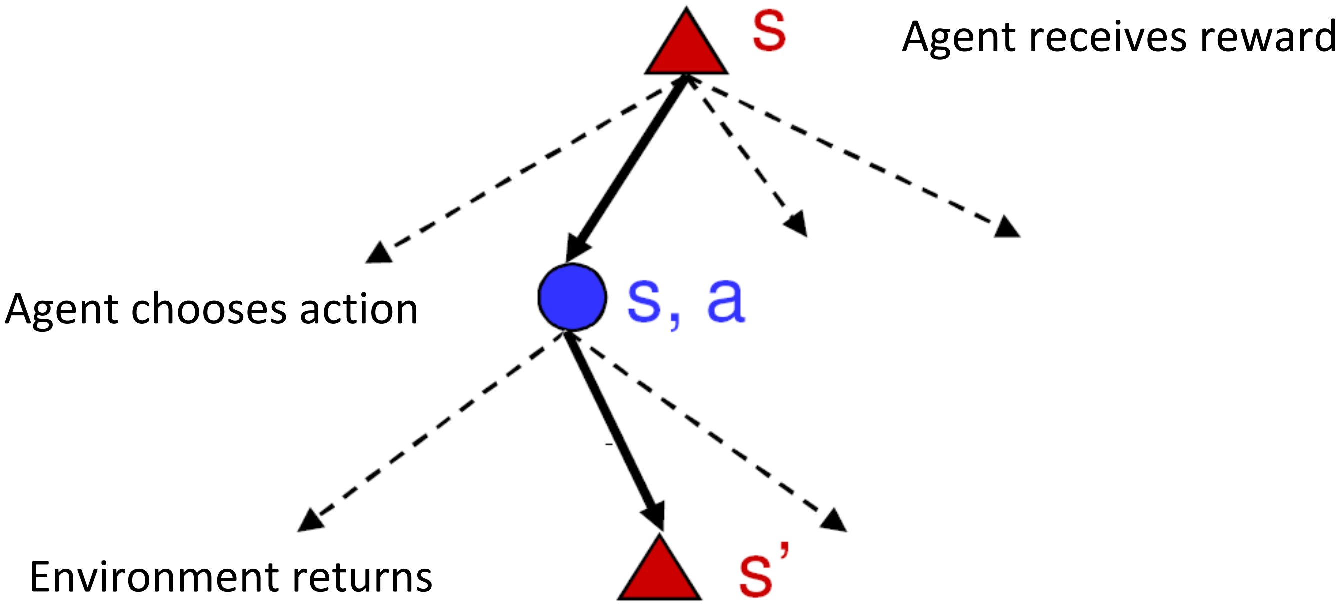
The Bellman equation



- Define state utility $V^*(s')$ as the expected sum of discounted rewards if the agent executes an *optimal* policy starting in state s'

Image source: L. Lazbenik

The Bellman equation

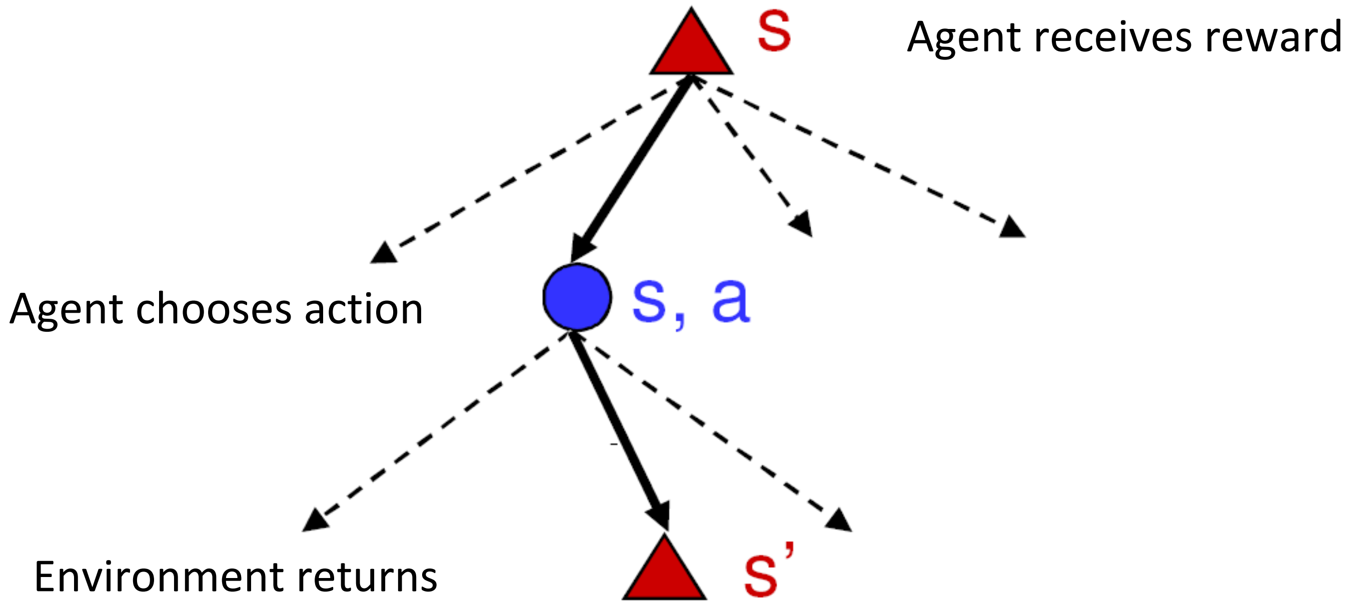


- What is the expected utility of taking action **a** in state **s**?

$$\sum P(s'|s,a)V^*(s')$$

Image source: L. Lazbenik

The Bellman equation

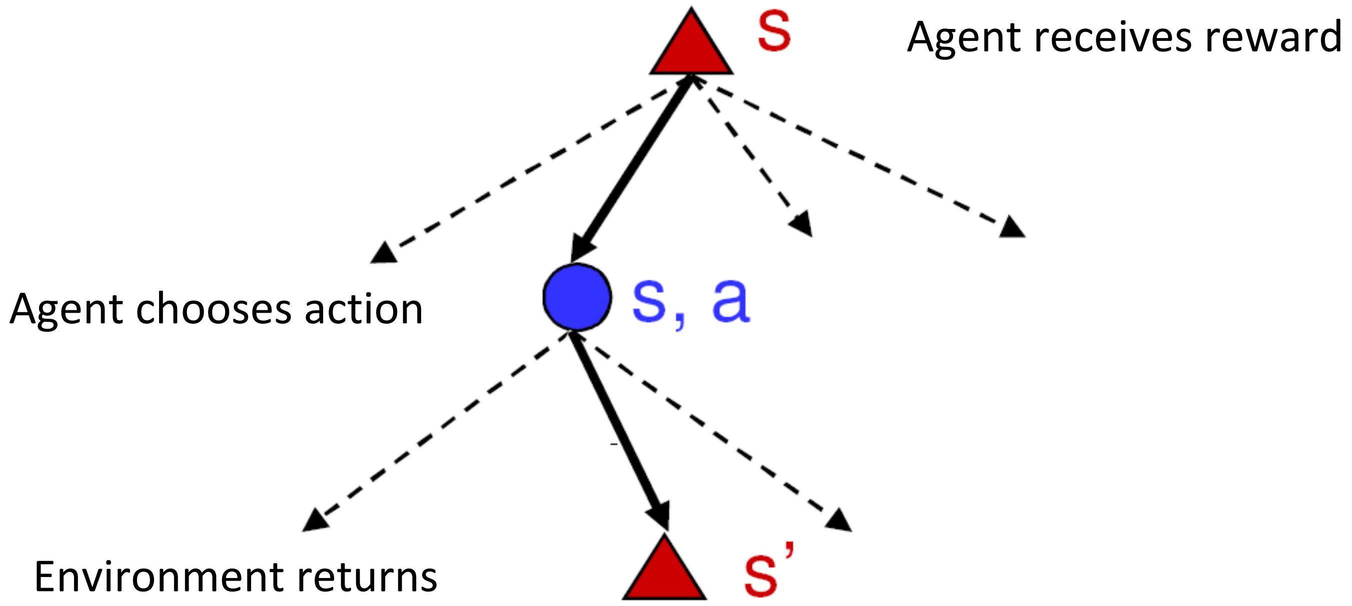


- How do we choose the action?

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

Image source: L. Lazbenik

The Bellman equation

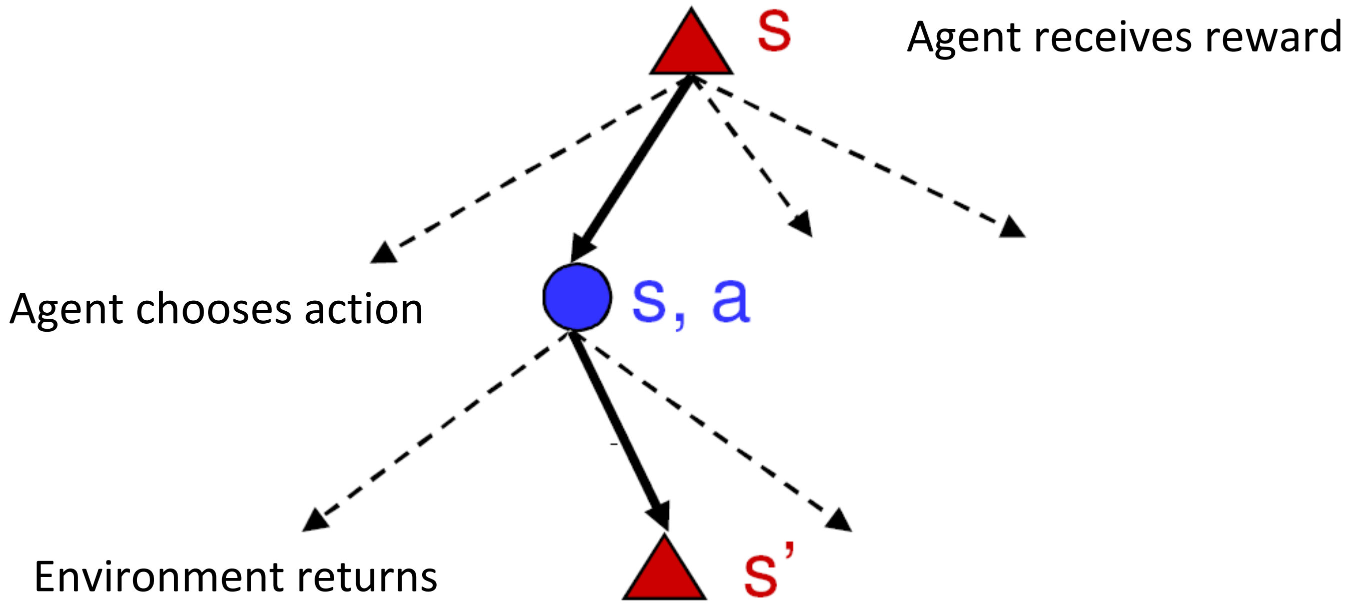


- What is the recursive expression for $V^*(s)$ in terms of $V^*(s')$ - the utilities of its successors?

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

Image source: L. Lazbenik

The Bellman equation

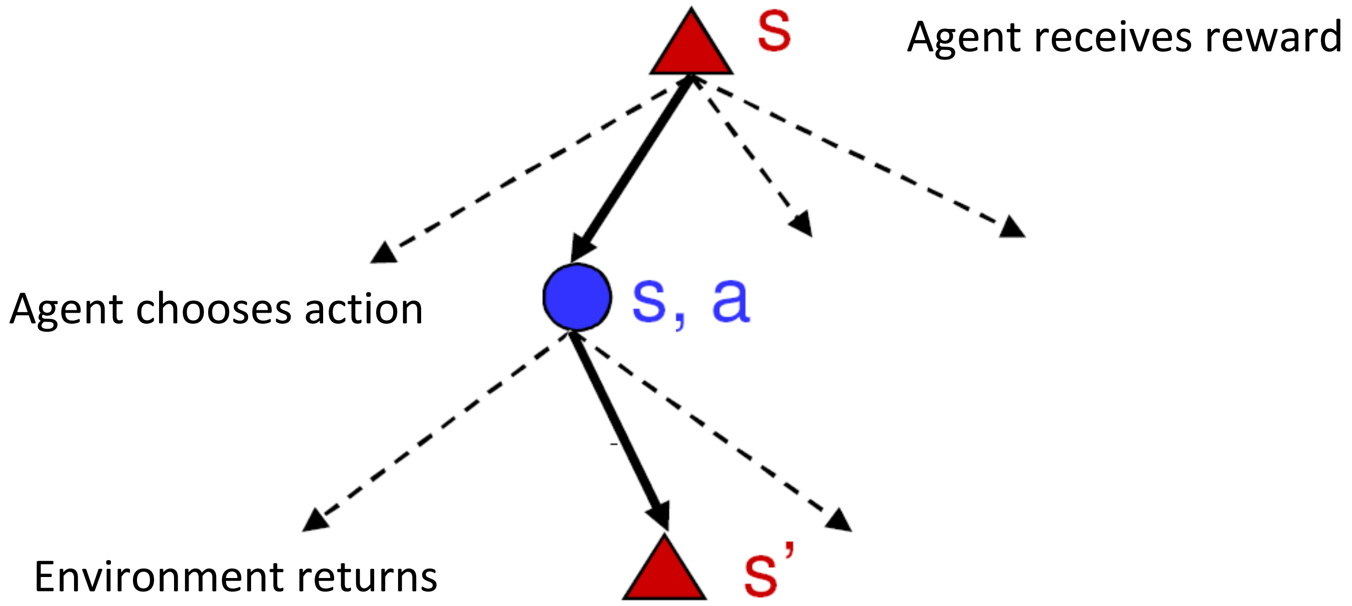


- How do we choose the action?

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

Image source: L. Lazbenik

The Bellman equation



- What is the recursive expression for $V^*(s)$ in terms of $V^*(s')$ - the utilities of its successors?

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

Image source: L. Lazbenik

Value Iteration to find optimal value function

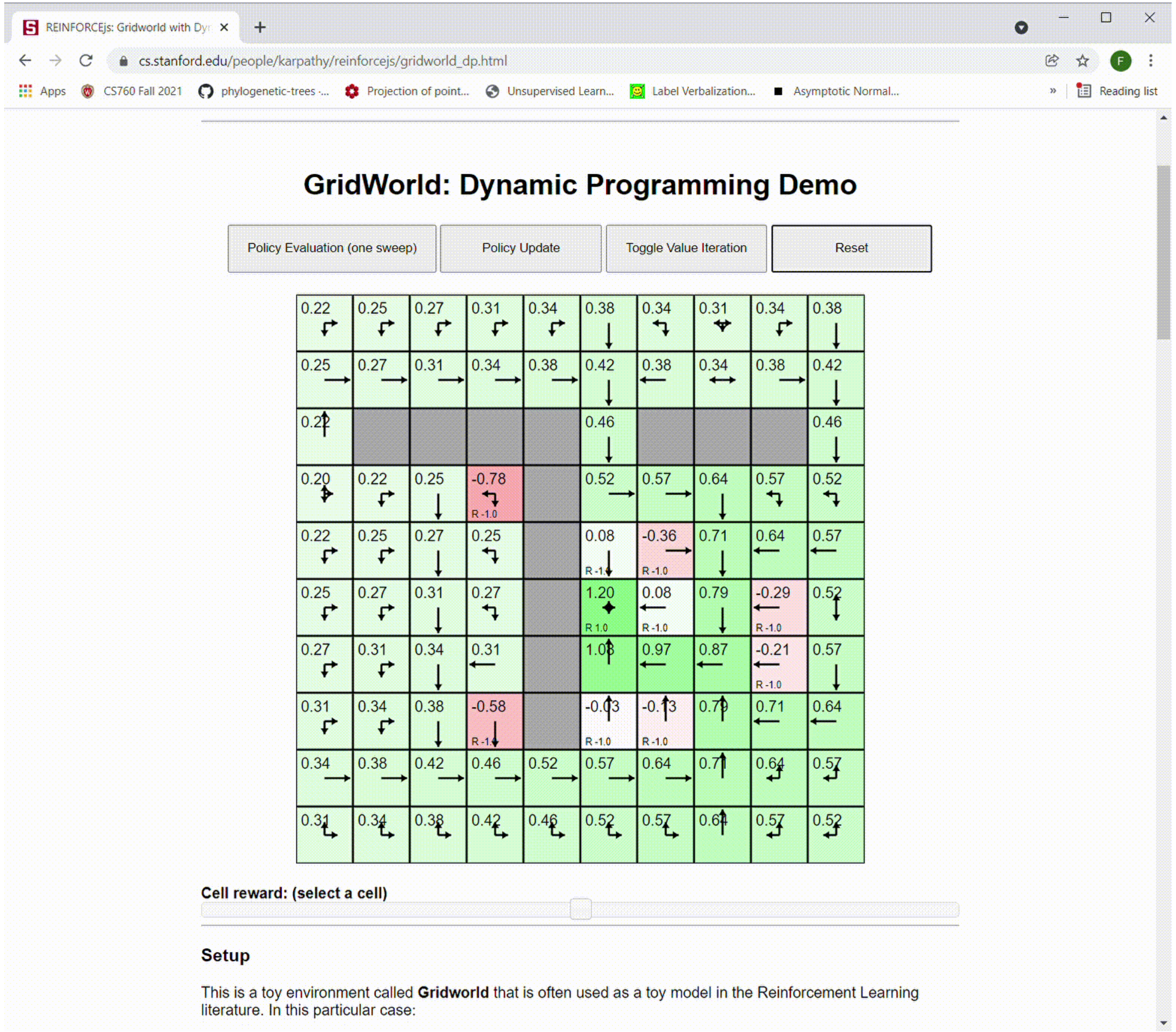
Q: how do we find $V^*(s)$?

- Why? 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 Bellman Equation. 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

Break & Quiz

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

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. $1/(1-\gamma)$
- **C. $1/(1-\gamma^2)$**
- D. 1

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)** (States: A,B,A,B,… rewards 1,0, γ^2,0, γ^4,0, …)
- D. 1

Outline

- **Intro to Reinforcement Learning**
  - Basic concepts, mathematical formulation, MDPs, policies
- **Valuing and Obtaining Policies**
  - Value functions, Bellman equation, value iteration, policy iteration
- **Q Learning**
  - Q function, Q-learning, SARSA, approximation

Planning vs Learning

So far we have assumed that the transition probability $P(s'|s,a)$ is known?

What if it is unknown?

Q-function (Action value function)

Q(s,a) tells us the value of doing action a in state s.

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

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

- Note: $V^*(s) = \max_a Q(s,a)$
- Now, we can just do $\pi^*(s) = \mathrm{argmax}_a\, Q(s,a)$

Learning the Q-function

What is wrong with the following strategy?

- Initialize $Q_0(s,a) = 0$ for all states and actions. Then,

$$Q_{i+1}(s,a) \leftarrow r(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q_i(s',a')$$

The transition probabilities $P(s'|s,a)$ are unknown!

Instead, use monte-carlo approximations (i.e from data) by observing that

$$Q(s,a) = r(s) + \gamma \mathbb{E}_{S'} \left[\max_{a'} Q(S',a') \middle| s,a \right]$$

Q-Learning

Offline setting: Estimating Q(s,a) from a given dataset (i.e sequences of the form s_0, a_0, s_1, a_1, s_2, a_2, …).

- 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.

Q-Learning: Making decisions while learning

Online setting: Make decisions while simultaneously learning.

- Make good decisions from the partially learned Q function
- But also update the Q function based on new data

- Update rule (the same)

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

- But what action do you choose on round t? Can you always simply choose:

$$a_t = \arg\max_a Q_{t-1}(s, a)$$

Exploration Vs. Exploitation

- **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 $0<\varepsilon<1$ 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

An alternative:

- Just use the next action in the update rule, 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**)
- Can use with epsilon-greedy policy

Q-Learning Details

We have assumed known deterministic rewards so far r(s). Q-Learning works even if rewards are unknown and/or stochastic

“Model-free”: we do not try to estimate transitions P(s’|s,a).

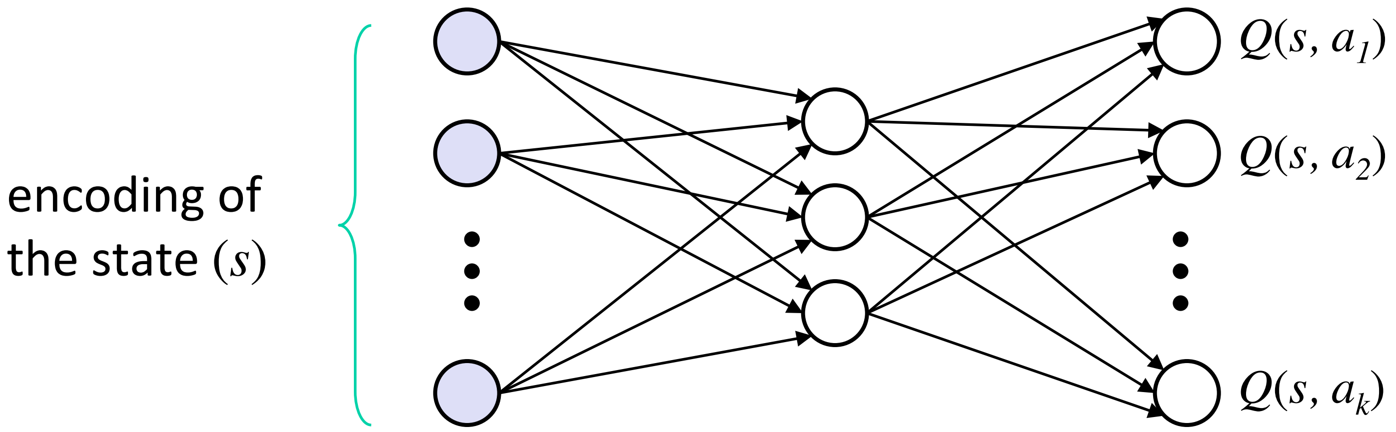
Note: if we have a **terminal** state, the process ends

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

Q-Learning – Compact Representations

Q-table can be quite large… might not even fit memory

- Solution: use some other representation for a more compact version. E.g: neural networks.

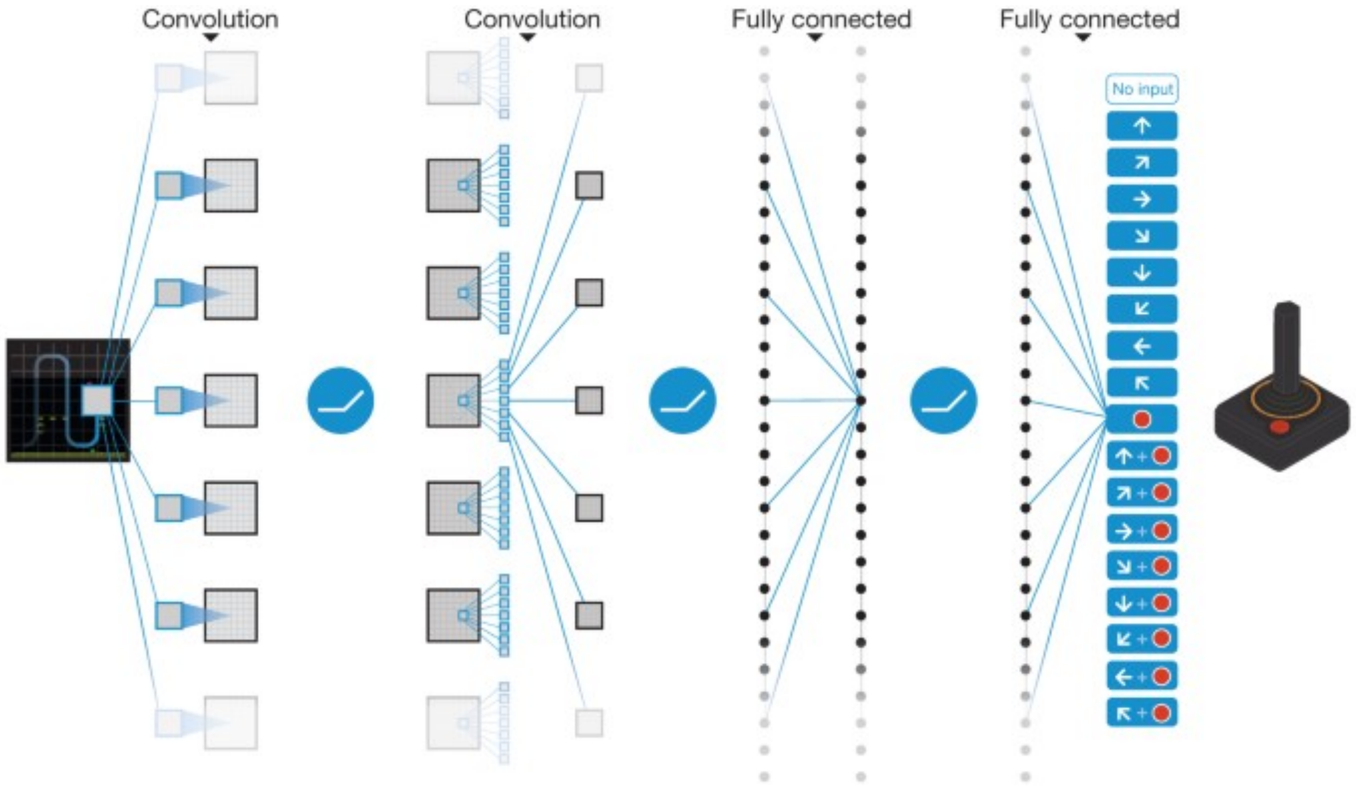


each input unit encodes a property of the state (e.g., a sensor value)

or could have <u>one net</u> for <u>each</u> possible action

Deep Q-Learning

How do we get Q(s,a)?



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

Break & Quiz

When the actions and states are discrete, for Q learning to converge to the true Q function, we must

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

Break & Quiz

When the actions and states are discrete, for Q learning to converge to the true Q function, we must

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

Break & Quiz

When the actions and states are discrete, for Q learning to converge to the true Q function, we must

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

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