CS 540 Introduction to Artificial Intelligence
Reinforcement Learning I
University of Wisconsin-Madison

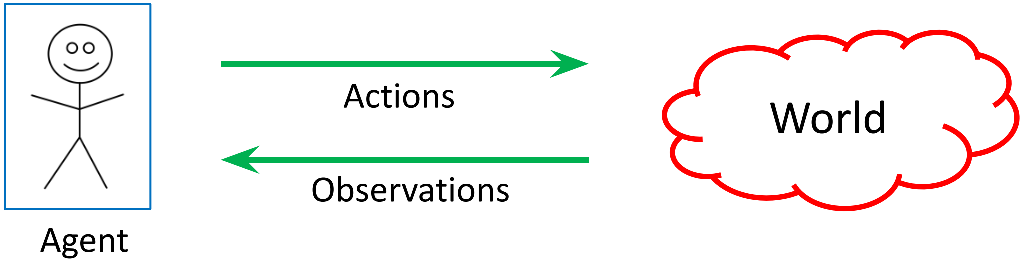
Spring 2022

Outline

- Introduction to reinforcement learning
  - Basic concepts, mathematical formulation, MDPs, policies
- Valuing policies
  - Value functions, Bellman equation, value iteration
- Q-learning

Back to Our General Model

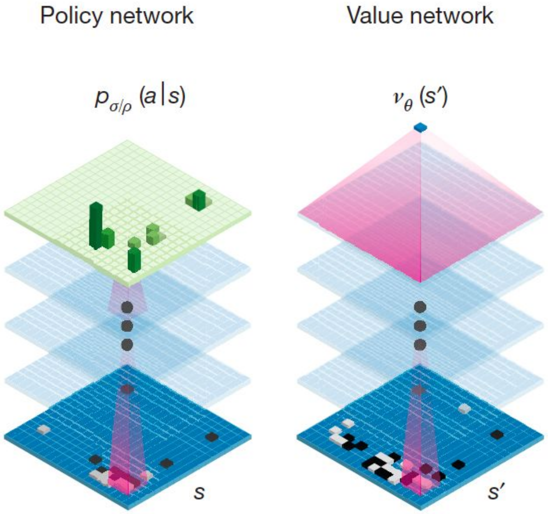
We have an **agent** **interacting** with the **world**



- 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

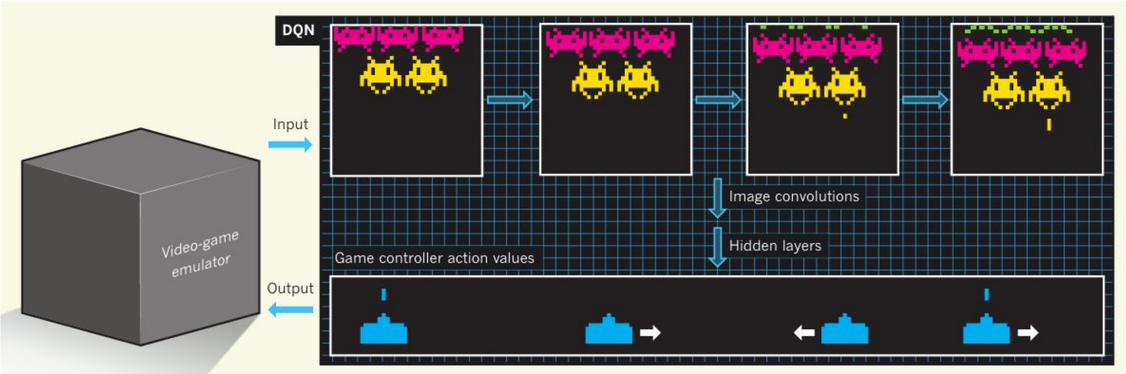
AlphaZero:



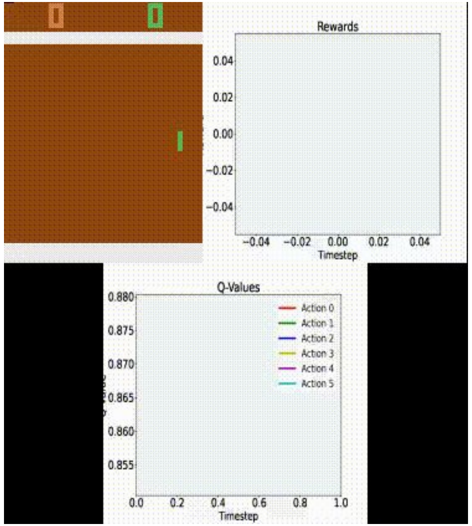
https://deepmind.com/research/alphago/

Examples: Video Game Agents

Pong, Atari



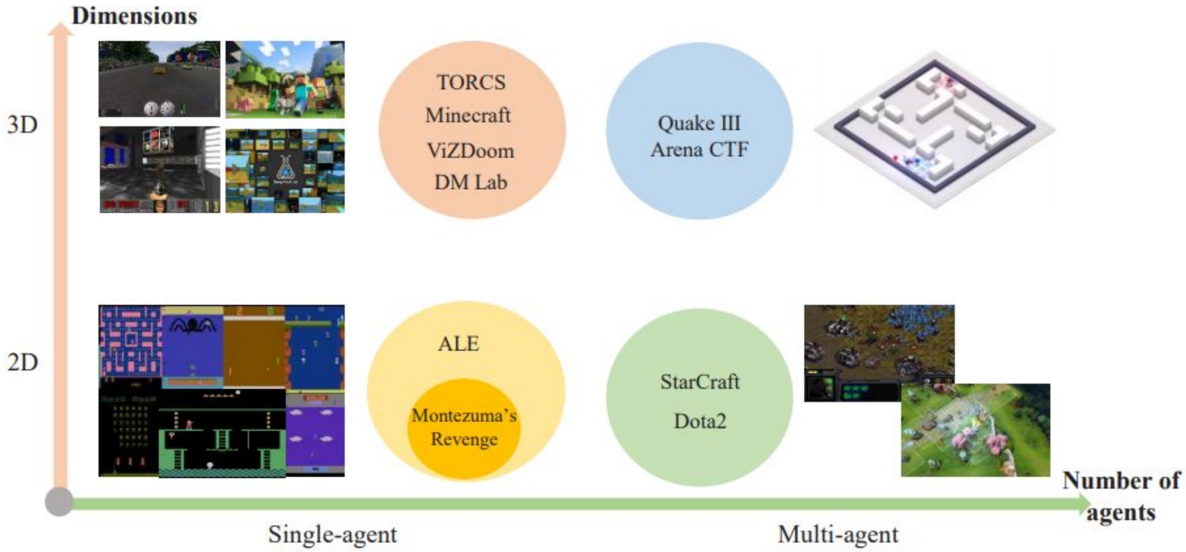
Mnih et al, “Human-level control through deep reinforcement learning”



A. Nielsen

Examples: Video Game Agents

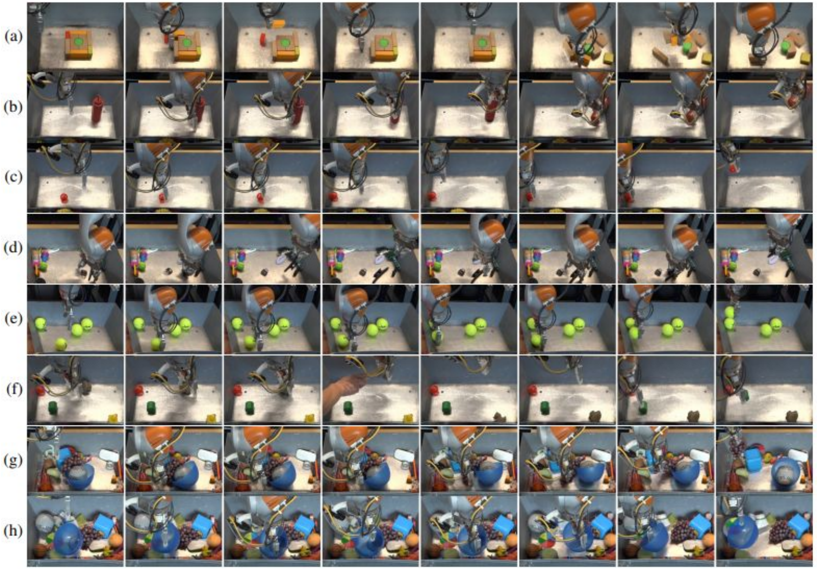
Minecraft, Quake, StarCraft, and more!



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

Examples: Robotics

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

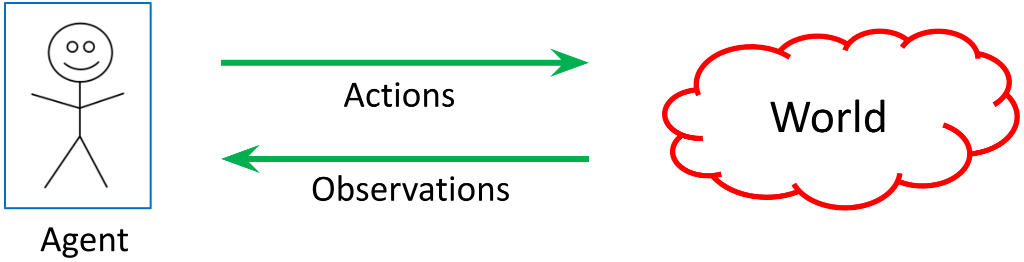


Ibarz et al, " How to Train Your Robot with Deep Reinforcement Learning – Lessons We’ve Learned "

Building The Theoretical Model

Basic setup:

- Set of states, S
- Set of actions A
- Interaction:
  - At time *t*, observe state $s_t \in S$.
  - Agent makes choice $a_t \in A$.
  - Gets reward r_t, state changes to s_{t+1}, continue



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

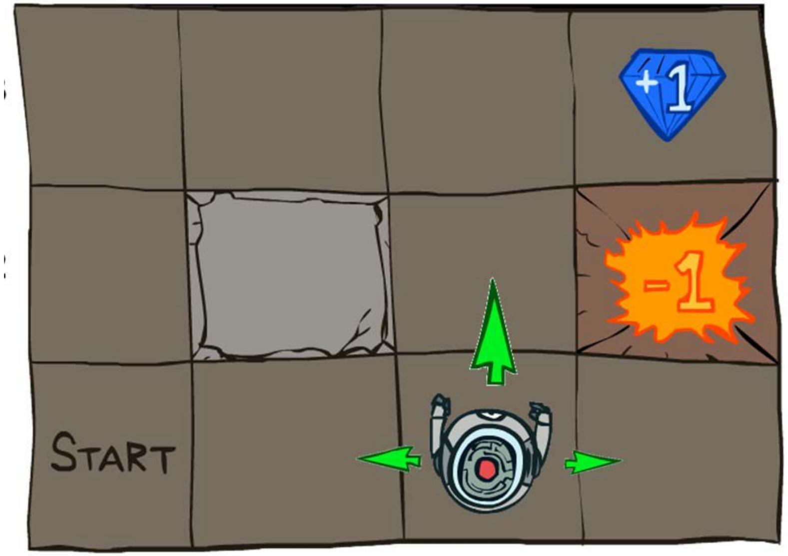
Markov Decision Process (MDP)

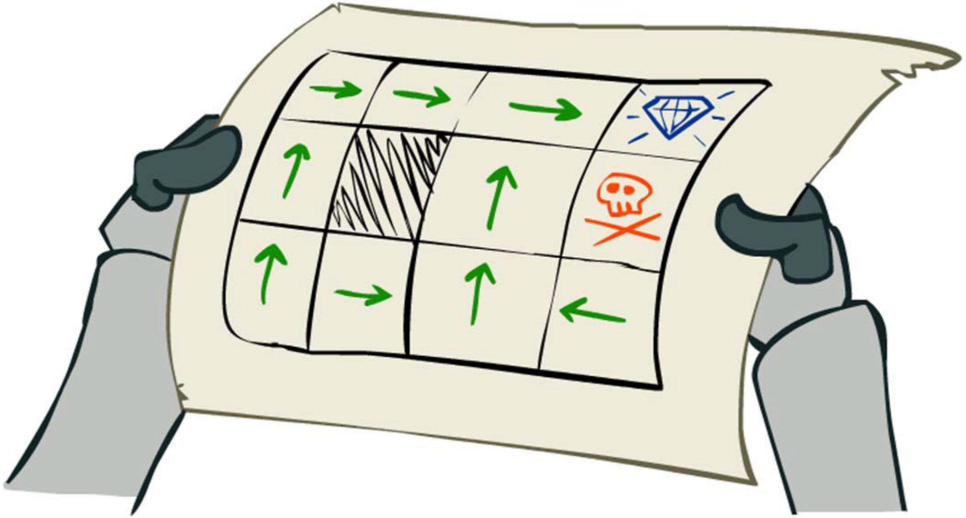
- The formal mathematical model $M = (S, A, P, r, \mu, \gamma)$:
- **State set** S. Initial state s_0. **Action set** A
- **Reward function:** $r(s_t, a_t)$
- **State transition model:** $P(s_{t+1} | s_t, a_t)$
  - Markov assumption: transition probability only depends on s_t and a_t, and not earlier history (older actions or states.
  - More generally: $P(r_t, s_{t+1} | s_t, a_t)$
- **Policy:** $\pi(s) : S \rightarrow A$ action to take at a particular state.

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

Example of MDP: Grid World

Robot on a grid; goal: find the best policy

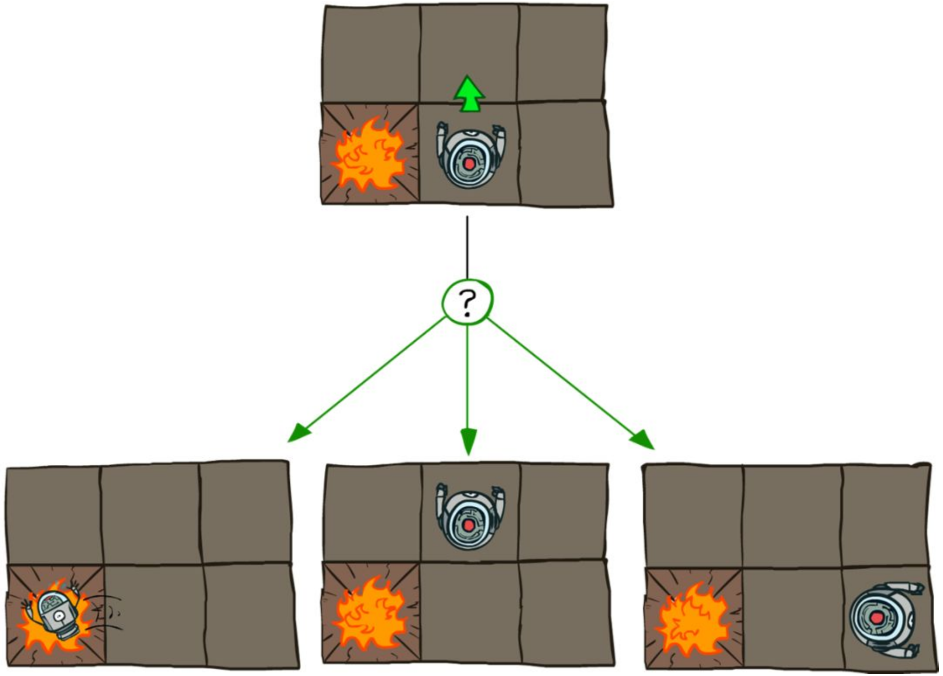


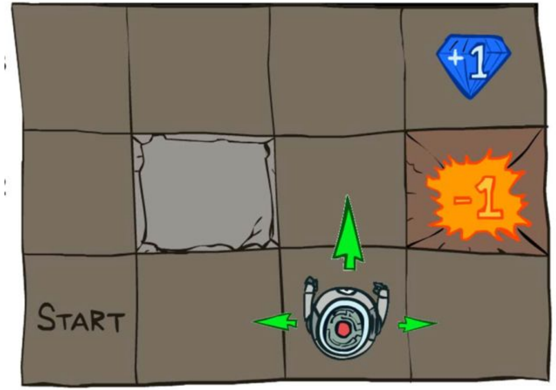


Source: P. Abbeel and D. Klein

Example of MDP: Grid World

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



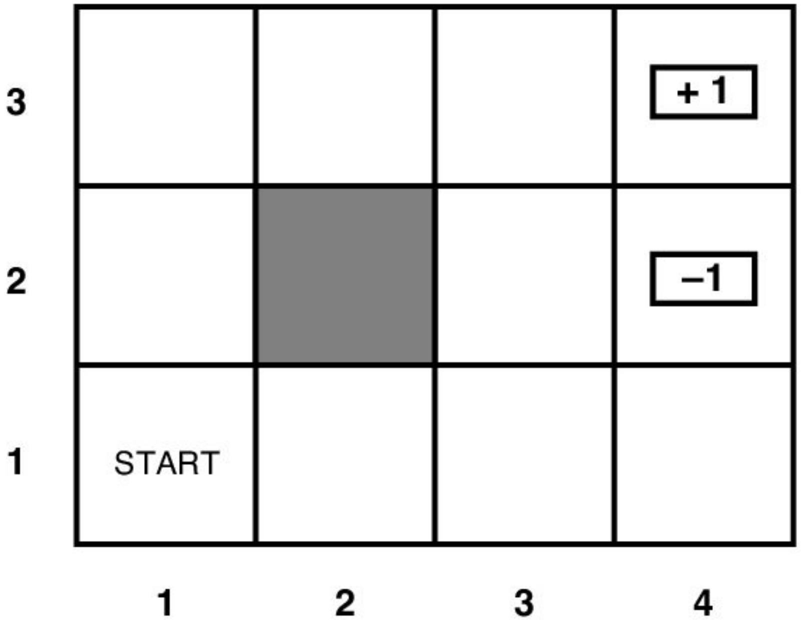


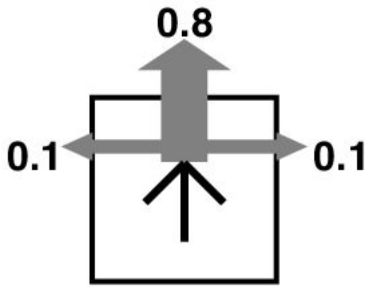
$r(s_{t+1}) = -0.04$ for every non-terminal state s_{t+1}

Shorthand for perfectly correlated $p(r_t, s_{t+1} | s_t, a_t)$

Grid World Abstraction

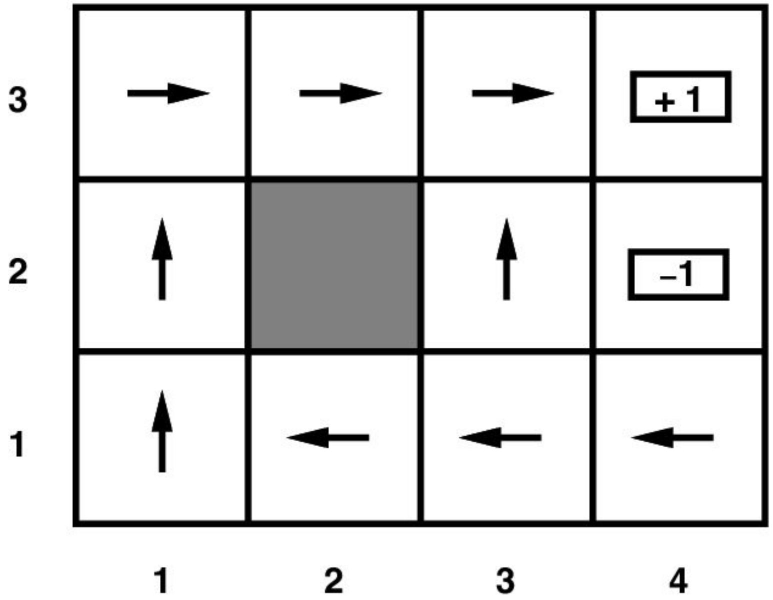
Note: (i) Robot is unreliable (ii) Reach target fast

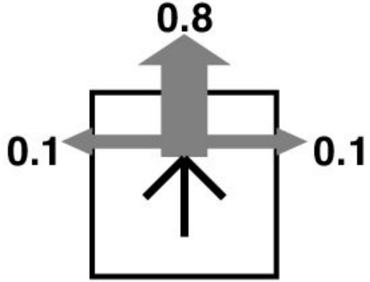




Grid World Optimal Policy

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





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:** $r(s_t, a_t)$
- **Policy**: $\pi(s) : S \rightarrow A$ action to take at a particular state.

How do we find the best policy?

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

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

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

Break & Quiz

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

- A. The reward function must output a scalar value (**True: need to be able to compare**)
- B. The policy maps states to actions (**True: a policy tells you what action to take for each state**).
- **C. The probability of next state can depend on current and previous states** (**False: Markov assumption**).
- D. The solution of MDP is to find a policy that maximizes the cumulative rewards (**True: want to maximize rewards overall**).

Defining the Optimal Policy

- For policy π, **the value** starting from s_0 produced by following that policy:

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

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

Discounted Rewards

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

- Solution

$$U(sequence) = r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots = \sum_{t \geq 0} \gamma^t r_t$$

- Discount factor $\gamma \in (0,1)$
  - 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?

- Optimal policy $\pi^* \in argmax_{\pi} V^{\pi}(s_0)$
- At any state s, we should take action $a = \pi^*(s)$
- Define $V^*(s) = V^{\pi^*}(s)$
- If we know V^*, we can extract π^*:

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

Bellman Equation

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

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

immediate 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
- Assume we know: reward $r(.)$, transition probability $P(s'|s,a)$
  - Knowing r and P is the "planning" problem
  - In reality r and P must be estimated from interactions with the MDP environment: "reinforcement learning"
- Also know $V^*(s)$ satisfies Bellman equation (recursion above)

A: fixed point iteration

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

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

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. $1/(1-\gamma)$
- **C. $1/(1-\gamma^2)$** (States: A,B,A,B,… rewards 1,0, γ^2,0, γ^4,0)
- D. 1

Q-Learning

Our first reinforcement learning algorithm

- Don't know the whole r and P. But can see interaction trajectory (s_t, a_t, r_t, s_{t+1})
- **Q-learning**: get an action-utility 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^*!

The Q*(s,a) function

- Starting from state s, perform (perhaps suboptimal) action a. THEN follow the optimal policy

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

- Equivalent to

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_b Q^*(s',b)$$

Q-Learning

- Estimate Q*(*s*,*a*) from data $\{(s_t, a_t, r_t, s_{t+1})\}$:

1. Initialize Q(.,.) arbitrarily (eg all zeros)
    1. Except terminal states $Q(s_{terminal},.)=0$
2. Iterate over data until Q(.,.) converges:

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_b Q(s_{t+1}, b))$$

Learning rate

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 prevent you from discovering the true optimal strategy

Q-Learning: ε-Greedy Behavior Policy

Getting data with both **exploration and exploitation**

- With probability ε, take a random action; else the action with the highest (current) Q(*s*,*a*) value.

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

The Q-learning algorithm

Input: step size α, greedy parameter ϵ

1. Q(.,.)=0
2. for each episode
3. draw initial state $s \sim \mu$
4. while (s not terminal)
5. perform $a = \epsilon$-greedy(Q), receive r, s'
6. $Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_b Q(s',b))$
7. $s \leftarrow s'$
8. endwhile
9. endfor

Note: step 5 can use any other behavior policies

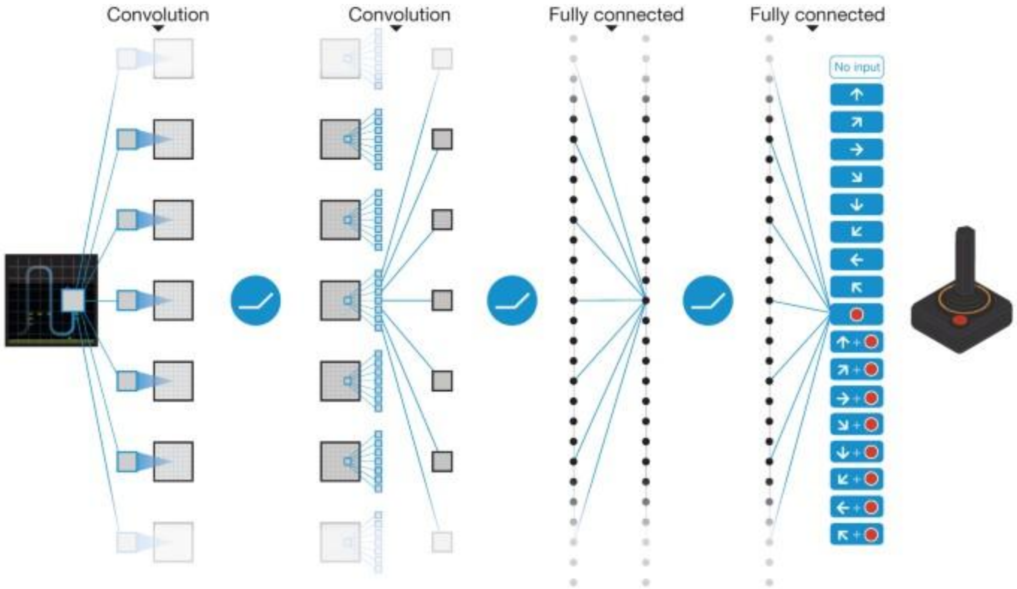
The Q-learning algorithm

- Step 5 can use any other behavior policies to choose action a, as long as all actions are chosen frequently enough
- The cumulative rewards during Q-learning may not be the highest
- But after Q-learning converges, can extract an optimal policy:

$$\pi^*(s) \in \mathrm{argmax_a Q(s, a)}$$
$$V^*(s) = \max_a Q^*(s, a)$$

Deep Q-Learning

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



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

Summary

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