CSC 580 Principles of Machine Learning

16 Reinforcement learning (RL)

Chicheng Zhang

Department of Computer Science

THE UNIVERSITY OF ARIZONA

*slides credit: built upon CSC 580 Fall 2021 lecture slides by Kwang-Sung Jun

Reinforcement learning references

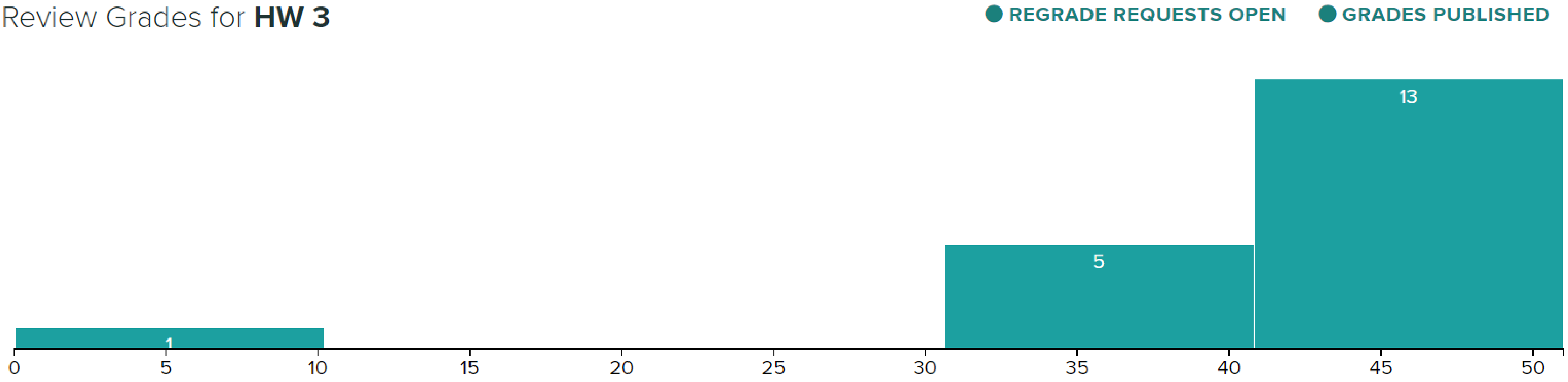
- ‘’Reinforcement learning’’ by Sutton & Barto (available online)
- RL course by David Silver: https://www.youtube.com/playlist?list=PLqYmG7hTraZBiG_XpjnPrSNw-1XQaM_gB

Reinforcement learning (RL)

- Task of an agent embedded in an environment
- repeat forever:
  - 1) sense world (=state)
  - 2) reason
  - 3) take an action (this changes the state)
  - 4) get feedback (usually a **reward** in $\mathbb{R}$),
  - 5) learn from the feedback


HW3 summary

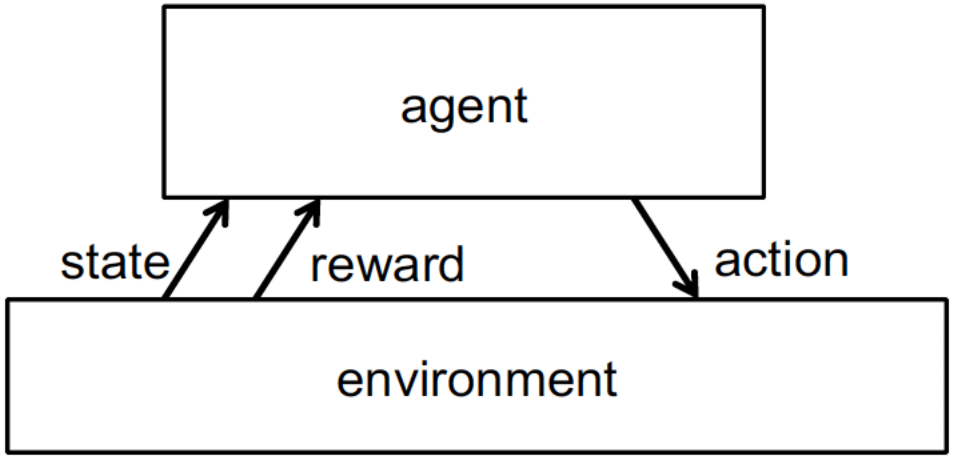
Review Grades for **HW 3**



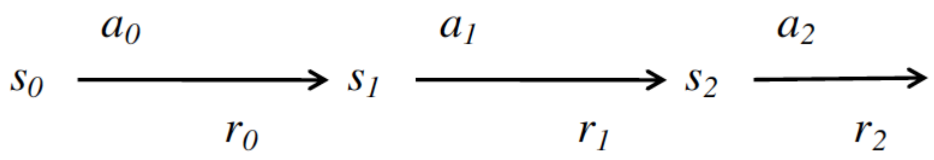
MINIMUM	MEDIAN	MAXIMUM	MEAN	STD DEV
10.0	**44.0**	**49.5**	**41.39**	**8.55**

- Solution guide is also up on D2L
- Final exam & project info up on Piazza

Reinforcement learning (RL)



- Markov decision process (MDP) environment $\mathcal{M}$
- set of states S
- set of actions A
- at each time t, agent observes state $s_t \in S$
  then chooses action $a_t \in A$
- then receives a reward r_t and moves to state s_{t+1}; repeat.



- **Markov assumption:**
  - $P(r_t | s_t, a_t, s_{t-1}, a_{t-1}, \dots) = P(r_t | s_t, a_t)$
  - $P(s_{t+1} | s_t, a_t, s_{t-1}, a_{t-1}, \dots) = P(s_{t+1} | s_t, a_t)$

These are unknown to the learner!

Goal: learn a <u>policy</u> $\pi : S \rightarrow A$ for choosing actions that maximizes **<u>expected cumulative (dicounted) reward</u>**

$$\mathbb{E}_\pi[r_0 + \gamma\, r_1 + \gamma^2 r_2 + \cdots \mid s_0] \text{ where } 0 \le \gamma < 1$$

for every possible starting state s_0

Unique challenges in RL I: temporal credit assignment

- Performance measure:
    - focuses on the quality of *a sequence of interdependent states / actions*
- Aim for maximization of *long-term rewards*

- E.g.
    - Daily exercise: short term – long term ++
    - Stay up all night playing video games: short term + long term --
    - Chess tactics: sacrifice pieces

- Different from supervised learning: correct classification on every individual examples
- Need to answer questions like: “what is the key step that caused me to lose this game?” – temporal credit assignment

Q: is Markov assumption too strong?

- We oftentimes define the state s_t to be the current observation o_t by the agent

- If so, how can we relax it?

Q: is Markov assumption too strong?

- We can always include the past observations into the state representation to incorporate past while keeping the Markovian assumption.
  - $s = (o_{-1}, o)$ *=> the state is now a tuple!*
  - be sure to disallow any transition that does not make sense
  - transition to $\left(o^{(1)}, o^{(2)}\right)$ to $\left(o^{(2)}, o^{(3)}\right)$ is valid
  - transition to $\left(o^{(1)}, o^{(2)}\right)$ to $\left(o^{(4)}, o^{(3)}\right)$ is invalid!

o_{-1}: the previous observation

The intention behind the RL formulation

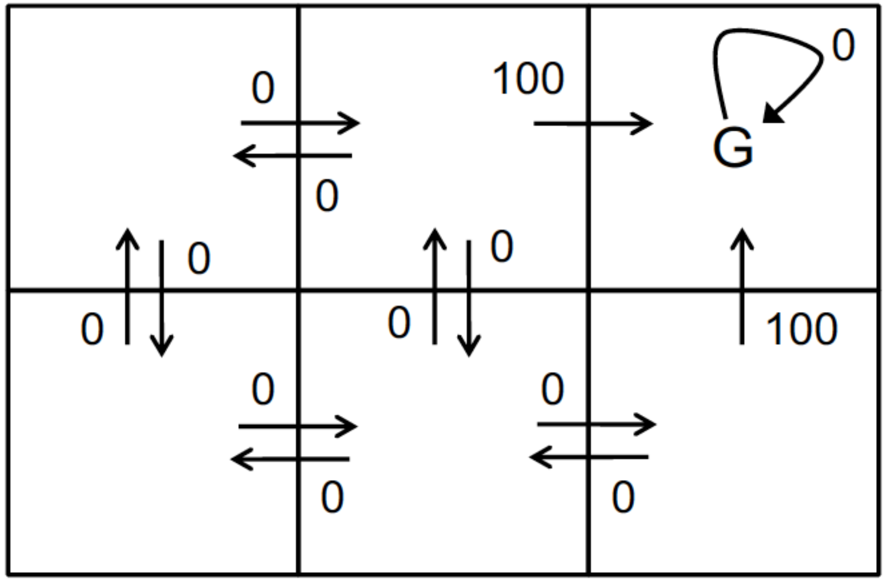
- Note that the formulation is **reward-driven.**
- Example: Robot learning: move a dish from one place to another
  - We can assign reward +10 when it accomplishes the task
  - We can also assign reward +1 when it picks up the dish successfully
- *Evaluative* feedback (cf. Instructive feedback – supervised learning)
- **The major premise:**
  All goals can be described by the maximization of expected cumulative reward.

(from David Silver's lecture)

Goal	Reward
Walk	Forward displacement
Escape maze	-1 if not out yet; 0 if out
Robots for recycling soda cans	+1 if a new can collected; -10 if run into things; 0 otherwise.
Win chess	0 if not finished; +1 if win; -1 if lose

The grid world: Learning to navigate

- The grid world



- State s: the location of the agent
- Each arrow represents an **action** a and the associated number represents **reward** $r(s, a)$ (assume that it is deterministic for now).

The structure of returns

- Define return at time step t:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$$

- The goal of RL: find a policy π that maximizes its return at the start:

$$\mathbb{E}_\pi[r_0 + \gamma\, r_1 + \gamma^2 r_2 + \cdots] = \mathbb{E}_\pi[G_0]$$

- G_t satisfies the following recurrence:

$$G_t = r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \cdots) = r_t + \gamma G_{t+1}$$

Current return Immediate reward Future return

Value function for a policy

- Given a policy $\pi: S \to A$, define its *value function* $V^\pi(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, \pi]$
- Important property (Bellman consistency equation):

$$
\begin{aligned}
V^\pi(s) &= \mathbb{E}[G_0 \mid s_0 = s, \pi] \\
&= \mathbb{E}[r_0 \mid s_0 = s, \pi] + \gamma \mathbb{E}[G_1 \mid s_0 = s, \pi] \\
&= R(s, \pi(s)) + \gamma \mathbb{E}_{s' \mid s, \pi(s)}[V^\pi(s')]
\end{aligned}
$$

where $R(s,a) = \mathbb{E}[r_t \mid s_t = s, a_t = a]$

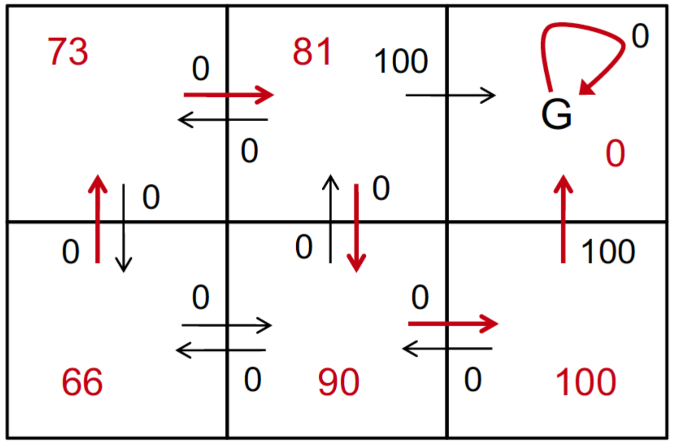
- Fact: there is a *single* policy π^* such that $\pi^* = \arg\max_\pi V^\pi(s)$ for all s
  - π^* is called the *optimal policy*
- $V^*(s)$:= the value function achieved by the optimal policy – optimal value function

* Note: We assume deterministic policies for simplicity; nondeterministic policy would assign probabilities to actions given state; i.e., p(a|s) =: $\pi(a|s)$ => $V^\pi(s) = \sum_{a \in A} \pi(a|s) \left(R(s,a) + \gamma \mathbb{E}_{s' \mid s, a}[V^\pi(s')] \right)$

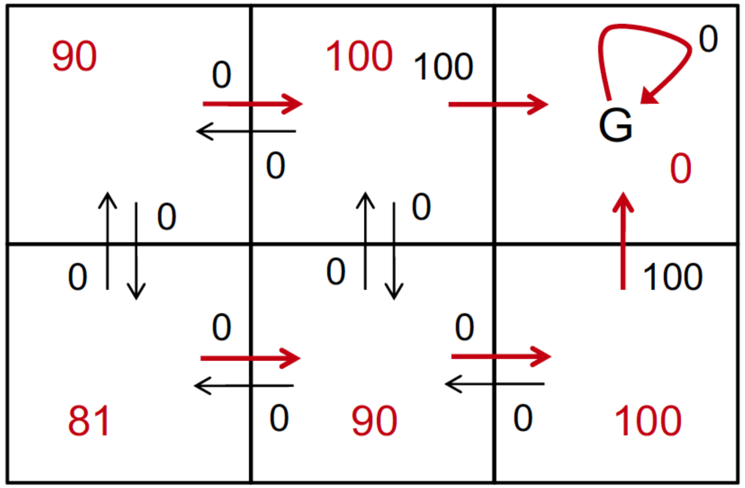
Value function for a policy π

- Suppose π is shown by red arrows, $\gamma = 0.9$

  $V^{\pi}(s)$ values are shown in red



optimal policy π^*



- The Bellman consistency equation:

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

* stochastic policy: $V^{\pi}(s) = \sum_a \pi(a|s) \left(R(s,a) + \gamma \cdot \sum_{s'} P(s'|s,a) \, V^{\pi}(s') \right)$

Policy evaluation

- How to compute V^π given MDP $\mathcal{M}$ and policy π?
- Recall Bellman consistency equation:

$$V^\pi(s) = \sum_a \pi(a|s)\left(R(s,a) + \gamma \cdot \sum_{s'} P(s'|s,a)\, V^\pi(s')\right)$$
$$= \underbrace{\sum_a \pi(a|s) R(s,a)}_{R^\pi(s)} + \gamma \cdot \sum_{s'} \underbrace{\left(\sum_a \pi(a|s)\, P(s'|s,a)\right)}_{M^\pi(s,s')} V^\pi(s')$$

- In matrix form (denote by $V^\pi = \left(V^\pi(s)\right)_{s\in S} \in \mathbb{R}^{|S|}$, etc):

$$V^\pi = R^\pi + \gamma M^\pi V^\pi$$

(recall the vector/matrix notation here)

- A linear system! How to solve it?
  - Gaussian elimination
- Is this efficient?
  - Time complexity: $O(|S|^3)$

Policy evaluation (cont'd)

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \left(R(s,a) + \gamma \cdot \sum_{s'} P(s'|s,a)\, V^{\pi}(s') \right)$$

<u>Fixed point iteration for policy evaluation</u>

Initialize: V^{π} arbitrarily (e.g., all zero).

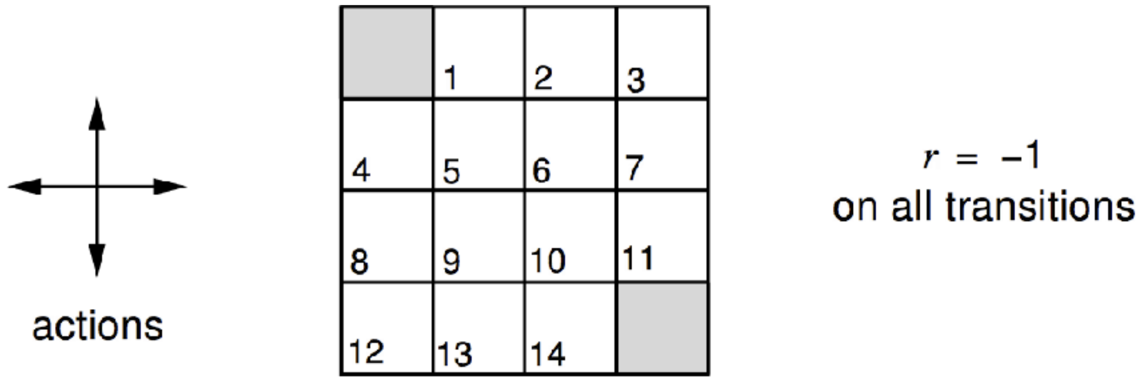
- While V^{π} does not change much from the previous iteration
  - $W^{\pi} \leftarrow V^{\pi}$
  - For each $s \in S$
    - $V^{\pi}(s) \leftarrow \sum_a \pi(a|s) \left(R(s,a) + \sum_{s'} P(s'|s,a) \cdot \gamma\, W^{\pi}(s')\right)$

- This is called synchronous update

- Asynchronous update: remove $W^{\pi} \leftarrow V^{\pi}$ and perform in-place updates for V^{π}
  - Preferred method.

Fixed point iteration: an illustration

- Episodic MDP (i.e., terminal states involved) with $\gamma = 1$
- Shaded squares are **terminal states**
- 4 actions
- Actions to the wall end up with the same state.
- Rewards are -1 until the terminal state is reached.
- The policy π: take an action uniformly at random.



Side Q: what’s the optimal policy under this reward setting?

Example

- Synchronous updates.
- Values are propagated!

v_k for the Random Policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

$$V^{\pi}(s) \leftarrow \sum_{a} \pi(a|s) \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a) \cdot W^{\pi}(s') \right)$$

Planning in MDPs

Planning in MDPs

- Given: full specification of $\mathcal{M}$, (specifically $\boldsymbol{R}(\boldsymbol{s}, \boldsymbol{a})$ **and** $\boldsymbol{P}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$ are known)
- Goal: find optimal policy π^* of $\mathcal{M}$

- Recall: $V^*(s)$ is the value function of the optimal policy.
- Claim: To find the optimal policy, it suffices to find $V^*(s)$ for every state s
- Why?

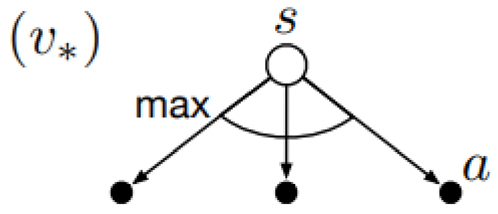
$$\pi^*(s_t) = \arg\max_{a \in A} R(s_t, a) + \gamma \sum_{s \in S} P(s_{t+1} = s | s_t, a) V^*(s)$$

- How to find $V^*(s)$?

Bellman optimality equation

- Fact: $V^*(s) = \max_\pi V^\pi(s)$ satisfies the following equation:

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



- This is known as the Bellman optimality equation
- Intuition:
  - $R(s,a) + \gamma \cdot \sum_{s'} P(s'|s,a)\, V^*(s')$ is the return achieved by: (1) taking action a; and (2) behave optimally afterwards
  - Optimal behavior = optimal action a + optimal behavior afterwards

- Issue: Bellman optimality equation has **no closed form solution**. (unlike computing V^π!)
- However, V^* can still be seen as a fixed point

Algorithm: Value iteration

Key idea: perform fixed point iteration on Bellman optimality equation

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

Initialize $V(s)$ arbitrarily

While $\{V(s)\}_{s \in S}$ is not much different from the previous iteration's $\{V(s)\}_{s \in S}$

- **For** each $s \in S$
  - $V(s) \leftarrow \max_a \; R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \cdot V(s')$
- **End For**

End While

Next lecture (11/30)

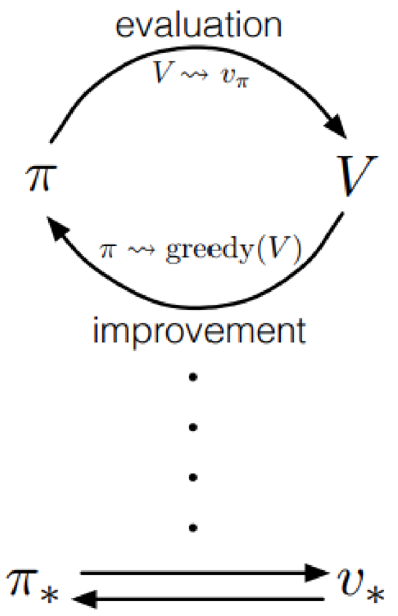
- RL cont’d: Q-learning
- Learning theory (if we have time)

- Assigned reading for this lecture and next lecture: Andrej Karpathy, Deep Reinforcement Learning: Pong from Pixels

Algorithm: Policy iteration

- The idea:

  estimate optimal value V^* and optimal policy π^* *simultaneously & iteratively*
- Observe:
  - π^* is greedy wrt V^*
  - V^* is the value function of π^*
- Can we obtain a pair (π, V) that exhibit the above properties?



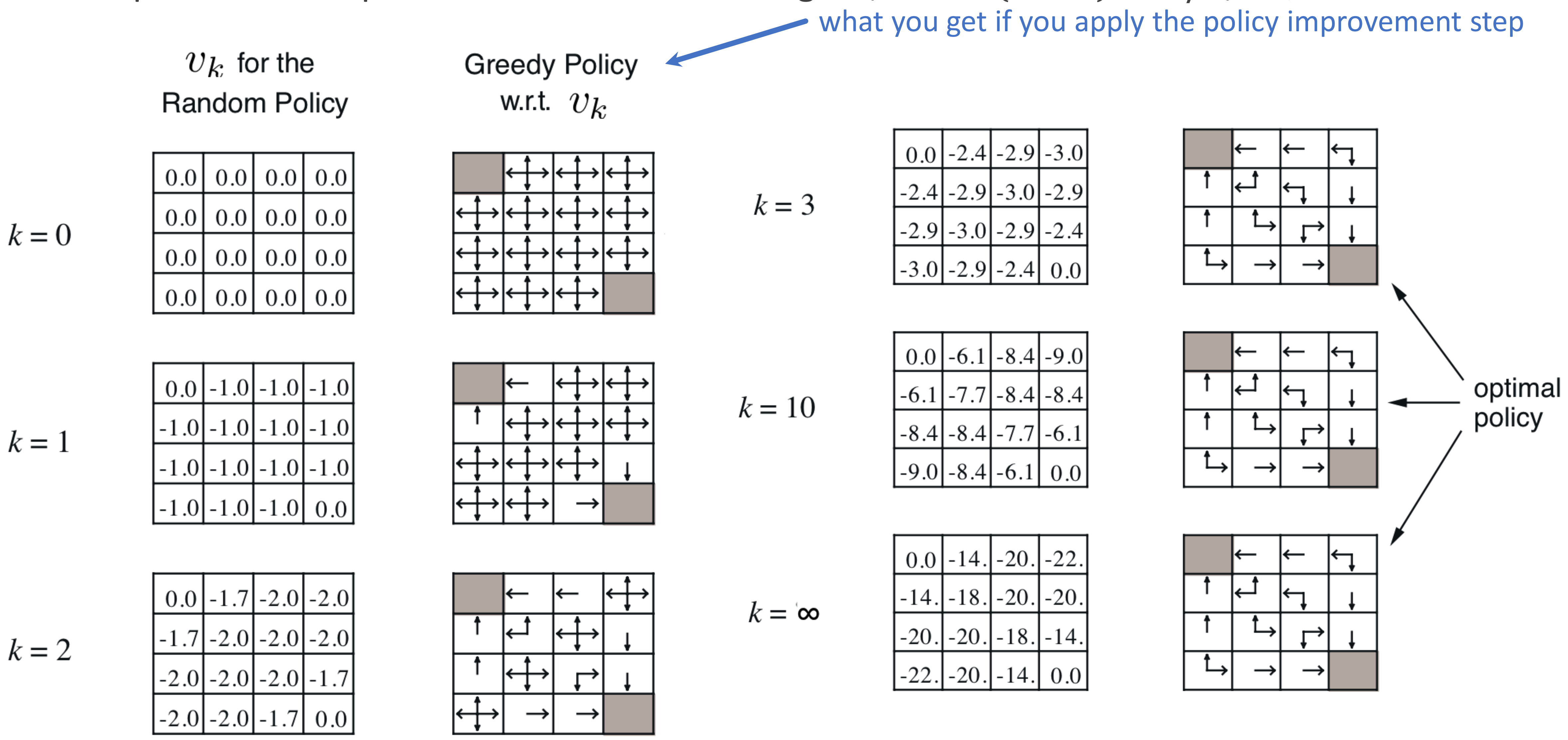
Algorithm:

- Start from an arbitrary policy π (e.g., assign actions randomly)
- Repeat the following:
  - **[Policy evaluation]** $V \leftarrow V^\pi$ (either solve the linear system or iterative method)
  - **[Policy improvement]** Update the policy: $\pi \leftarrow \text{greedy}(V)$

    For every $s \in S$, $\pi(s) \leftarrow \arg\max_a r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^\pi(s')$

Policy iteration with inexact policy evaluation

Suppose we perform fixed-point iteration for evaluating V^{π}, with $\pi(\, a \mid s\,) = 1/4, \forall s, a$



Algorithm: Modified policy iteration

- From previous slide: inexact value functions are still useful!

- Start from an arbitrary policy π (e.g., assign actions randomly)

- **[(Inexact) Policy evaluation]** $V \leftarrow$ take k fixed-point iterations for computing V^{π} (so $V \approx V^{\pi}$)

This is **not a valid value function** anymore (no corresponding π that achieves this value in general)

- **[Policy improvement]** Update the policy:
  - For every $s \in S$, $\pi(s) = \arg\max_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V(s')$

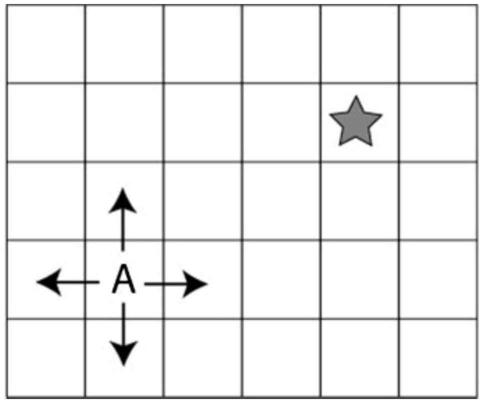
Summary

- Policy evaluation: just evaluates the value function for a given π
  - closed form / fixed-point iteration
- Planning:
  - Policy iteration: policy evaluation + policy improvement
  - Modified policy iteration: only k steps of policy evaluation
  - Value iteration: k=1

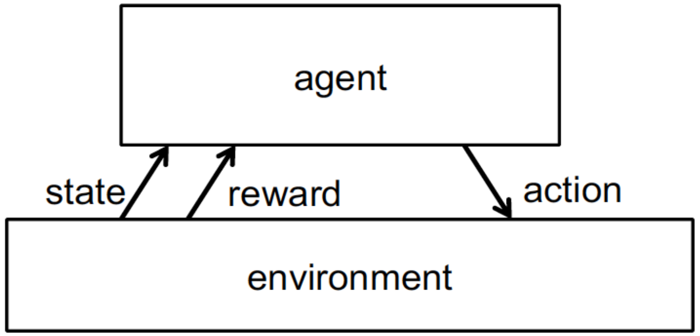
- Recall: so far, we are in the **planning** setting, where we are already given a **model** of the world: i.e. know $P(s'|s,a)$ and $P(r \mid s,a)$
- What if we don't? This is called the **"learning in MDPs"** problem

Learning in MDPs

Learning in MDPs: basic setup



- Given:
  - MDP $\mathcal{M}$ (unknown)
  - The ability to interact with $\mathcal{M}$ with T steps
    - Obtaining trajectory $s_0, a_0, r_0, \dots, s_T, a_T, r_T$

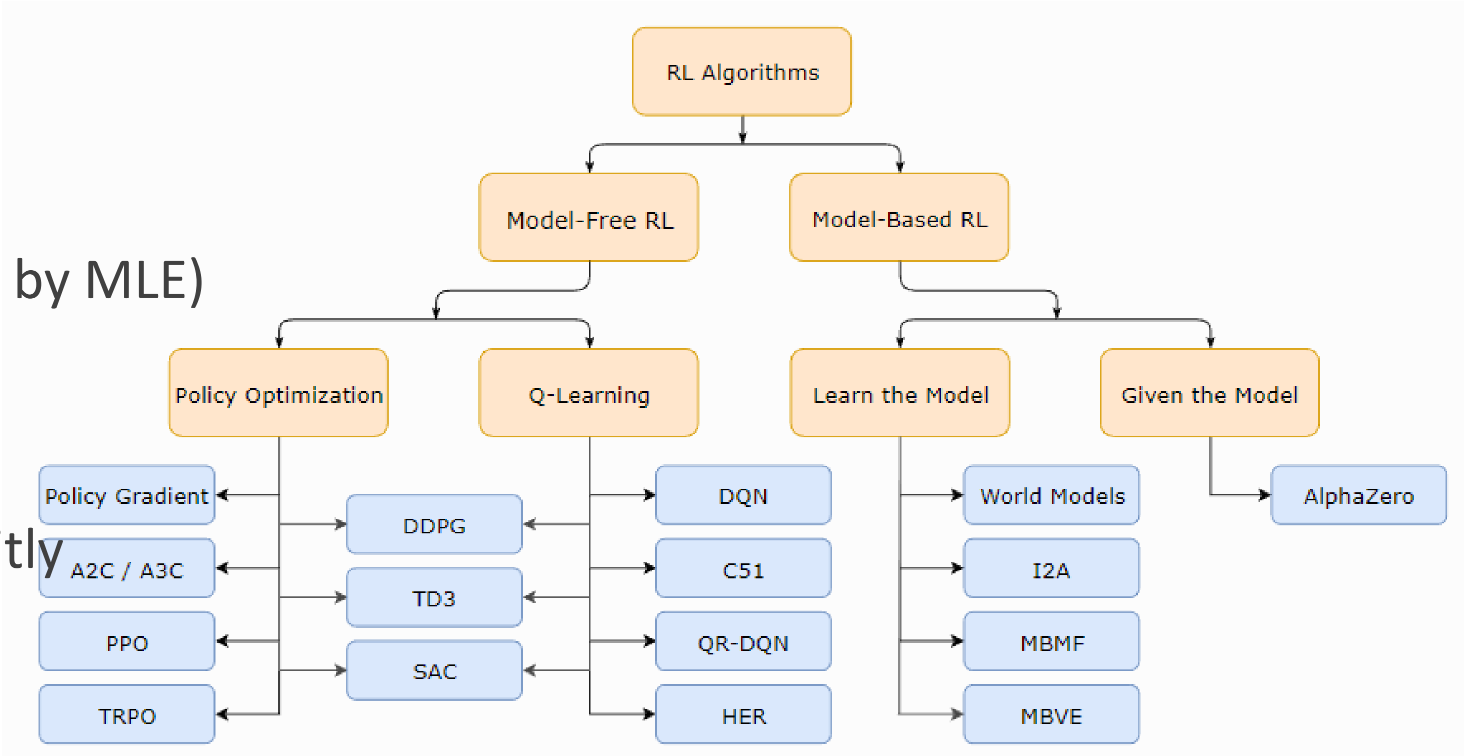


- Goal:
  - (Online learning) maximize cumulative reward $\mathbb{E}\left[\sum_{t=0}^{T} \gamma^t \, r_t\right]$
    - Useful in applications where every action taken have real-world consequences (e.g. medical treatment)
  - (Batch learning) output a policy $\hat{\pi}$ such that $V^{\hat{\pi}}$ is competitive with V^*
    - Useful in applications where experimentations are affordable (e.g. laboratory rats, simulators)

Learning in MDPs: a taxonomy of approaches

- Model-based RL:

  Repeat:
  - $\widehat{\mathcal{M}} \leftarrow$ Estimate $\mathcal{M}$ based on data (e.g. by MLE)
  - Plan according to $\widehat{\mathcal{M}}$

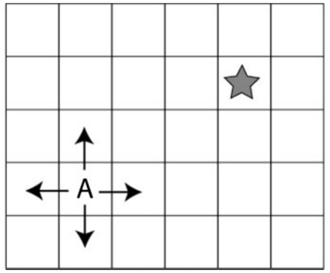


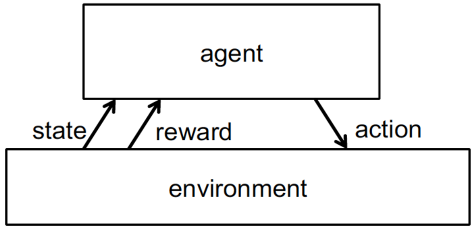
- Model-free RL: do not estimate $\widehat{\mathcal{M}}$ explicitly
  - Direct policy search
    - E.g. policy gradient (REINFORCE)
  - Value-based methods
    - E.g. Q-learning (this lecture)
  - Actor-critic: combination of the two ideas

https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

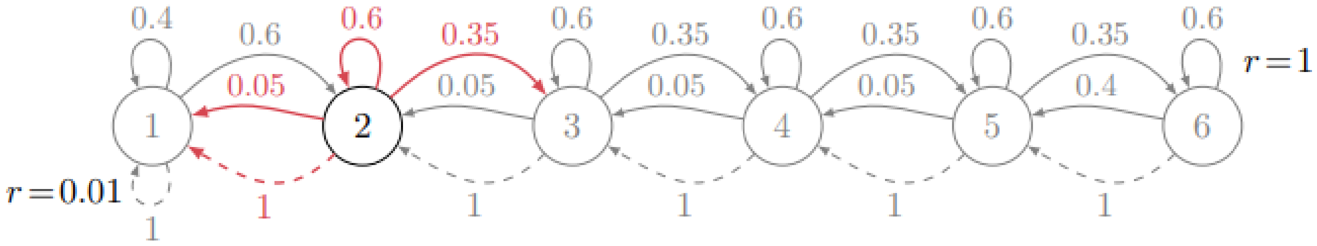
Unique challenges in RL II: exploration

- Learning agent's data is induced by its own actions
  - This is another key difference with supervised learning
- How to collect *useful* data?
  - The exploration challenge





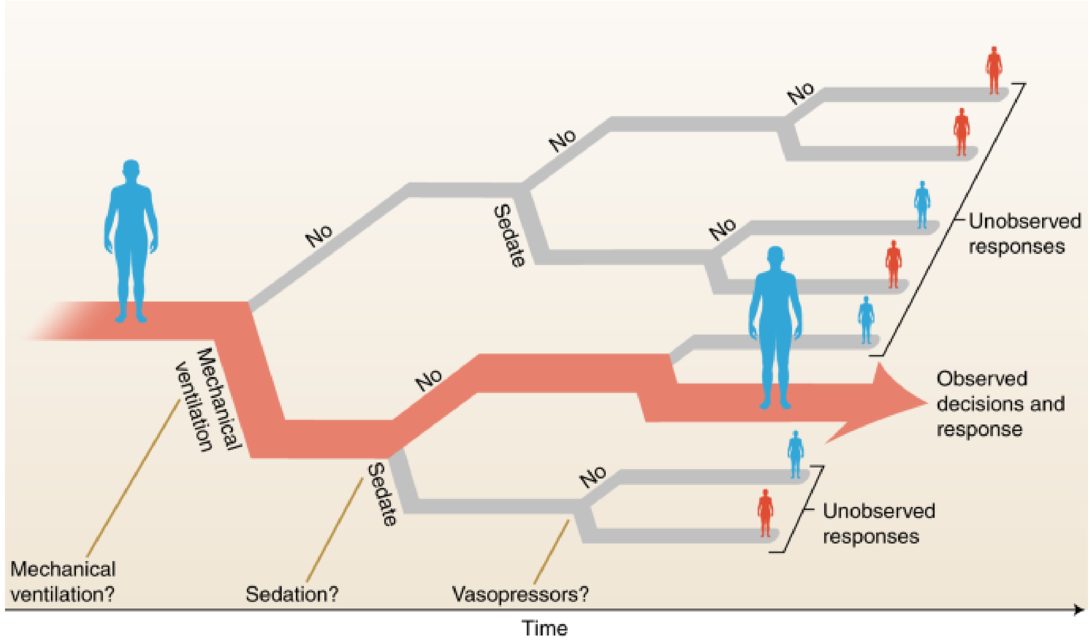
- Rough intuition: collect data that "covers" all states and actions
  - Uniform exploration: take actions uniformly at random
- Caveat: uniform exploration may fail because of some hard-to-reach states
  - E.g. RiverSwim [Strehl & Littman, 2008]



https://rlgammazero.github.io/docs/2020_AAAI_tut_part0.pdf

Unique challenges in RL II: exploration (cont’d)

- Extra challenge in the *online learning* setting
  - Need to take good actions that yield high rewards
  - Balance *exploration* vs. *exploitation*
  - Not an issue in the batch learning setting



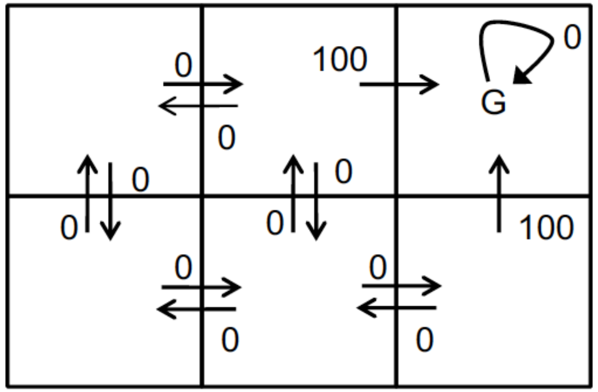
- Popular idea:
  - ϵ-greedy: w.p. $1 - \epsilon$, choose action that is believed to be optimal based on the information collected so far; otherwise, choose actions uniformly at random.
  - Again, ϵ-greedy may fail in some hard MDP environments

https://www.nature.com/articles/s41591-018-0310-5

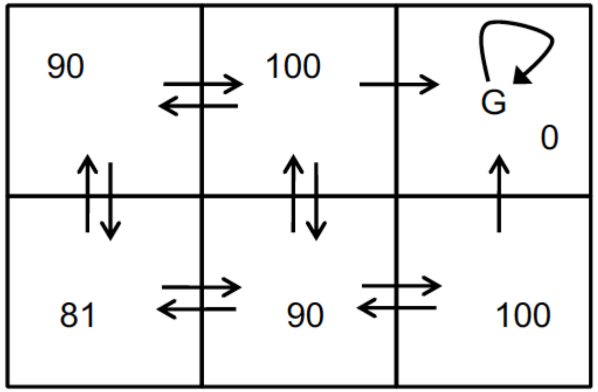
Q-functions: motivation

- Issue of V^π: only encodes the quality of states
  - But we need to learn what actions are good
- Is there a function that encodes the quality of actions as well?
- Action-value functions (Q-functions):
  - $Q^\pi(s,a) = \mathbb{E}[\, G_0 \mid s_0 = s, a_0 = a,\ \pi \,] = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^\pi(s')$
- The optimal Q function
  - $Q^*(s,a) = \mathbb{E}[\, G_0 \mid s_0 = s, a_0 = a,\ \pi^* \,] = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^*(s')$
- The optimal policy can be extracted from Q^*:
  - $\pi^*(s) = \arg\max_a Q^*(s,a)$

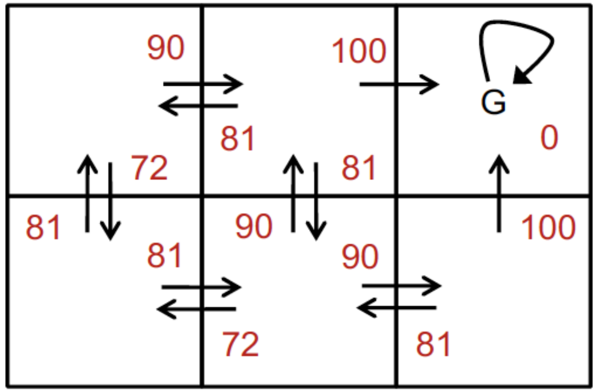
Q-values



$r(s, a)$ (immediate reward) values



$V^*(s)$ values



$Q^*(s, a)$ values

Q-learning: motivation

- We do not know the state transition nor the reward function.

- Instead of learning these model parameters, we directly attempt to estimate Q^*
  - Recall: "When solving a problem of interest, do not solve a more general problem as an intermediate step" – Vladimir Vapnik

- Similar to V^*, Q^* also satisfies a Bellman-optimality equation:

$$Q^*(s,a) = R(s,a) + \gamma \cdot \sum_{s'} P(\, s' \mid s, a \,) \max_{a'} Q^*(s', a')$$

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

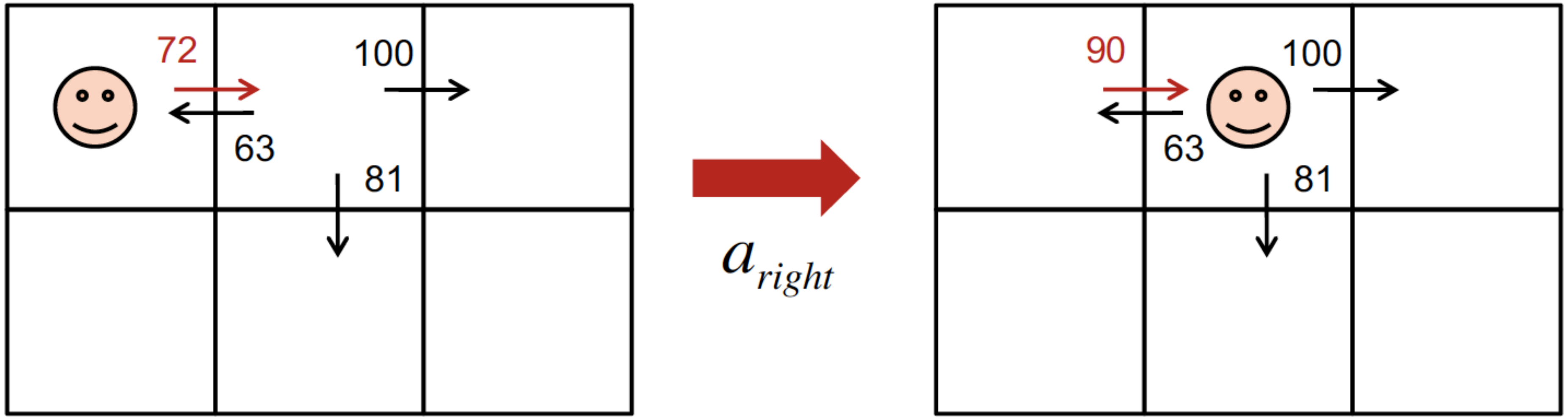
- We will use this to design our learning rule

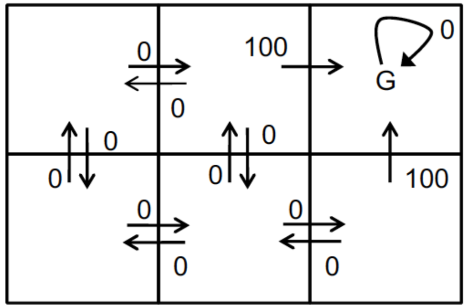
Algorithm: Q-learning (deterministic transitions/rewards)

- Assume that we are in the tabular setting: S and A are both finite
- Initialize: $Q(s,a) = 0, \forall s, a$
- Observe the initial state s
- Repeat:
    - Select an action a and execute it (e.g., ϵ-greedy)
    - Receive a reward r
    - Observe a new state s'
    - Update: $Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s', a')$ (similar to value iteration)
    - $\mathrm{s} \leftarrow s'$

$$Q^*(s,a) = R(s,a) + \gamma \cdot \sum_{s'} P(\, s' \mid s, a\,) \max_{a'} Q^*(s', a')$$

Q-learning: update example





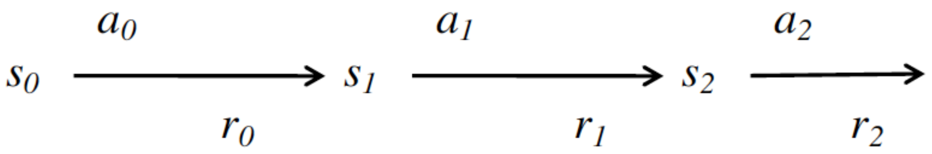
$r(s, a)$ (immediate reward) values

$$Q(s_1, a_{right}) \leftarrow r + \gamma \max_{a'} Q(s_2, a')$$
$$\leftarrow 0 + 0.9 \max\{63, 81, 100\}$$
$$\leftarrow 90$$

Q-learning for stochastic transitions/rewards

- Our update equation is problematic: $Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$

- For stochastic worlds:
  - Fix s, a, (next state, reward) s', r seen is stochastic
  - Even if $Q = Q^*$ in the previous iteration, $Q(s, a)$ will deviate from $Q^*(s, a)$ after the update
  - This results in $Q(s, a)$ not converging



- How to fix this? Recall:

$$Q^*(s, a) = R(s, a) + \gamma \cdot \sum_{s'} P(s' \mid s, a) \max_{a'} Q^*(s', a')$$

- We can use the idea of stochastic approximation (also called temporal difference learning in the RL context)

Stochastic approximation

- Given a *stream* of data points $X_1, \dots, X_n \sim N(\mu, 1)$
- How to estimate μ in an *anytime* manner?

- Idea 1: at time step n, output estimate $\hat{\mu}_n = X_n$

- Can we do better?

- Idea 2: at time step n, output estimate $\hat{\mu}_n = \frac{1}{n}(X_1 + \cdots + X_n)$
- This is equivalent to $\hat{\mu}_n = (1 - \alpha_n)\hat{\mu}_{n-1} + \alpha_n X_n$, where $\alpha_n = \frac{1}{n}$

Old estimate (conservativenss) — $(1 - \alpha_n)\hat{\mu}_{n-1}$

New data (correctivenss) — $\alpha_n X_n$

Q-learning for nondeterminstic transitions/rewards

$$Q^*(s,a) = R(s,a) + \gamma \cdot \sum_{s'} P(s' \mid s,a) \max_{a'} Q^*(s',a')$$

- Initialize: $Q(s,a) = 0, \forall s, a$
- Observe the initial state s
- Repeat
  - Take an action a
    - e.g., ϵ-greedy (taking $\text{argmax}_a Q(s,a)$ w.p. $1-\epsilon$)
    - or Boltzmann exploration: $a \sim P(a|s) = \frac{\exp(Q(s,a)/\tau)}{\sum_{a'} \exp(Q(s,a')/\tau)}$
  - Receive the reward r
  - Observe the new state s'
  - Update: $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a')\right)$
  - $s \leftarrow s'$

α is a hyperparameter! (next slide)

The choice of α

- $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a')\right)$
- For example, $\alpha = \frac{1}{1 + \#\text{times}(s,a)}$.
- Q: Why is this a reasonable choice?

Discussion

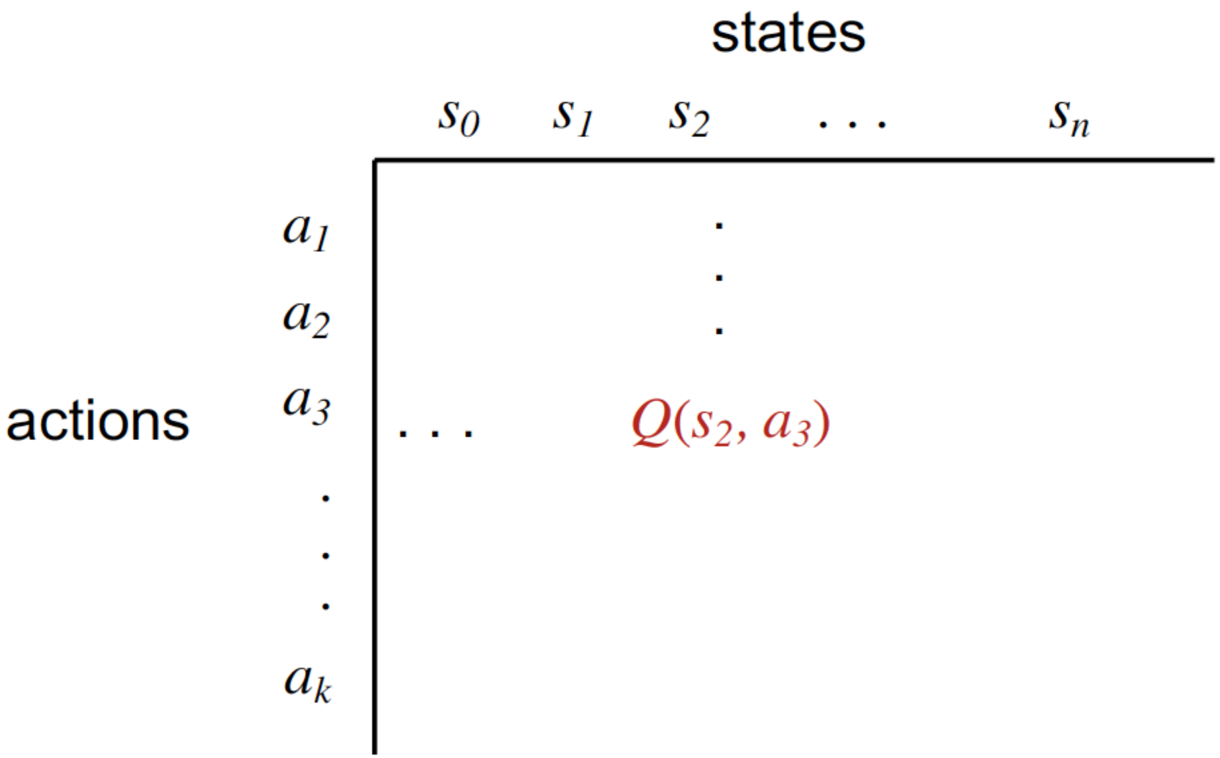
- Q-learning will converge to the optimal Q function (under certain niceness assumptions on the MDP, exploration policy, and step size scheme)

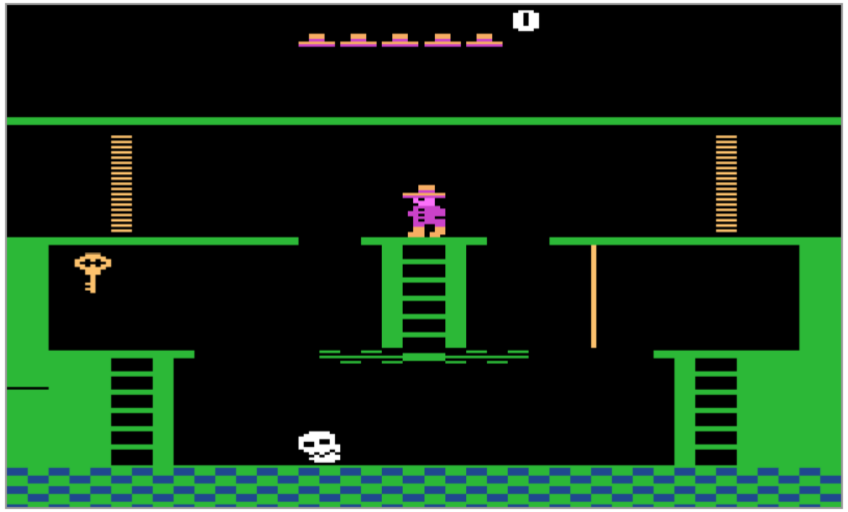
- In practice, it takes a lot of iterations!

- Comparison: Model-based learning vs. Q-learning when choosing actions
  - Model-based
    - need to look ahead using some estimates of rewards and transition probabilities (Model Predictive Control)
  - Q-learning (model-free)
    - just choose the action with the largest Q value

Challenge of Q-learning: large state spaces

- Q-learning requires us to maintain a huge table, which is clearly infeasible with large state spaces





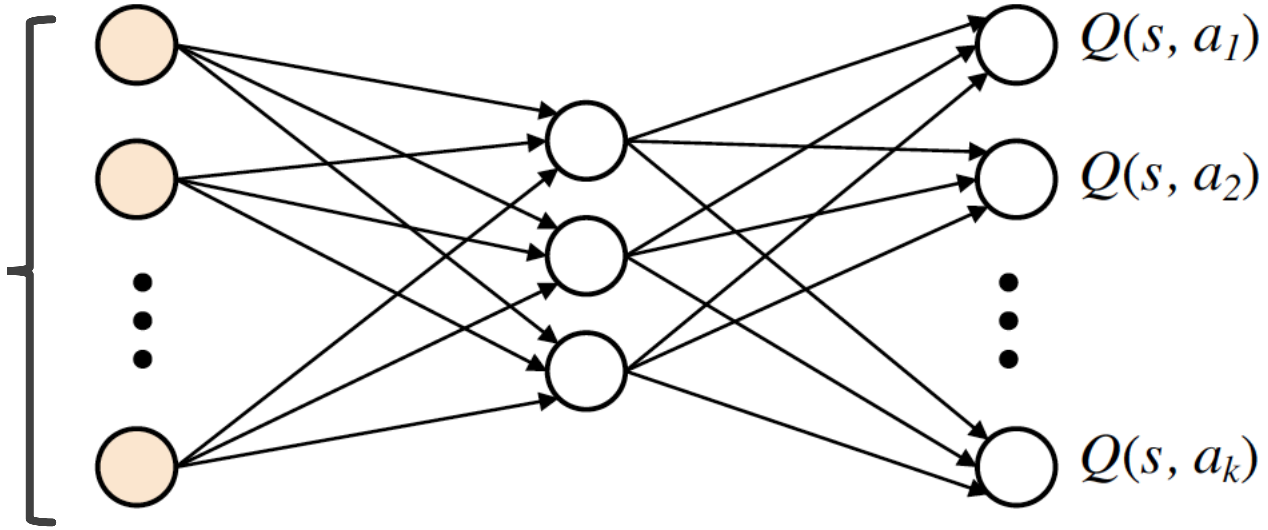
- How to design a Q-learning-style algorithm that can handle large state spaces?

https://www.microsoft.com/en-us/research/uploads/prod/2018/09/Reinforcement-Learning-with-Rich-Observations-SLIDES.pdf

Q function approximation

- We can use some other function representation (e.g. a neural net) to compactly encode a substitute for the big table.
- We've been thinking states as discrete (the set S), but in fact, they can be a feature vector!

encoding of the state



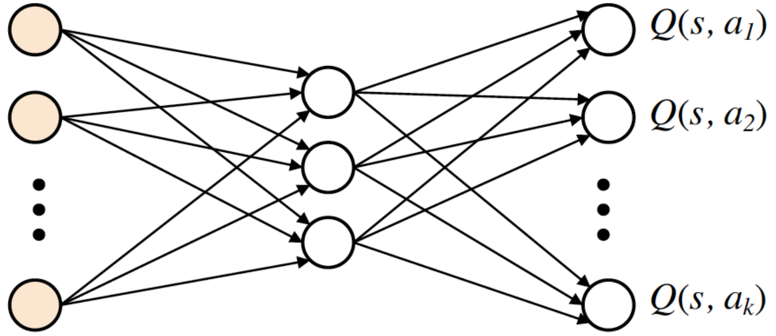
each input unit can be a sensor value
(or more generally, a feature)

Q: why is this a good idea?

Why Q function approximation?

- 1. memory issue
- 2. is able to *generalize across states*! may speed up the convergence.

- Example: 100 binary features for states. 10 possible actions.
- Q table size = 10 x 2^{100} entries
- NN with 100 hidden units:
  - 100 x 100 + 100 x 10 = 11k weights (not counting bias for simplicity)



Algorithm: fitted Q-learning

Repeat

- observe the state s
- compute $Q(s,a)$ for each action a (forward pass on the NN)
- select action a (use ϵ-greedy or Boltzmann exploration) and execute it
- observe the new state s’ and the reward r
- compute $Q(s',a')$ for each action a’ (forward pass on the NN)
- update the NN with the instance
  - $x \leftarrow s$
  - $y \leftarrow (1-\alpha)Q(s,a) + \alpha\left(r + \gamma \cdot \max_{a'} Q(s',a')\right)$ (label for Q(s,a))

Calculate Q value you would have put into the Q-table and use it as the training label.
Use the squared loss and perform backpropagation!

Fitted Q-learning example: Atari games

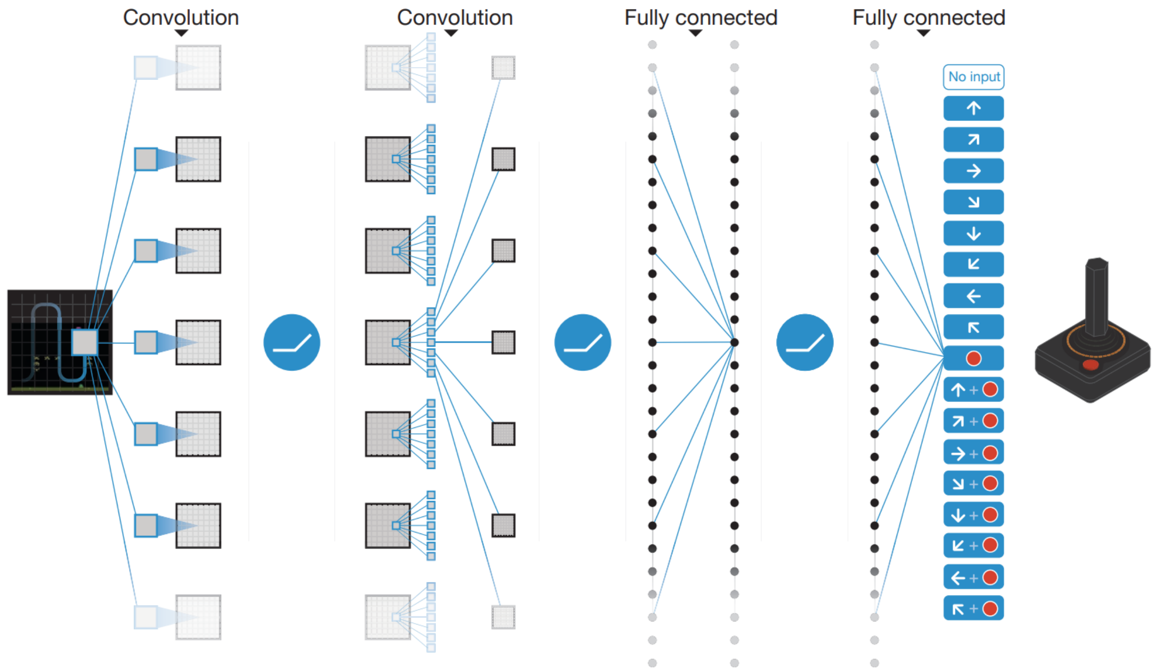
- Human-level control through deep reinforcement learning (Mnih et al, 2013, 2015)
- Tested Fitted Q-learning on 49 Atari games

- Achieves >=75% of human professional players’ scores on 29 games
- Can significantly outperform human players in many games

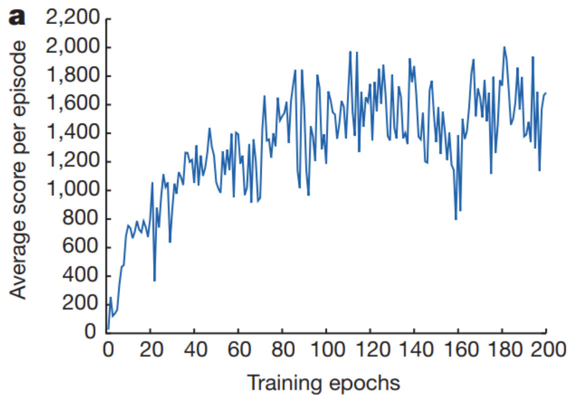
https://arxiv.org/pdf/1312.5602.pdf
https://www.nature.com/articles/nature14236

Fitted Q-learning example: Atari games (cont'd)

- The neural network for fitting Q values
  - Convolutional architecture to handle states as images

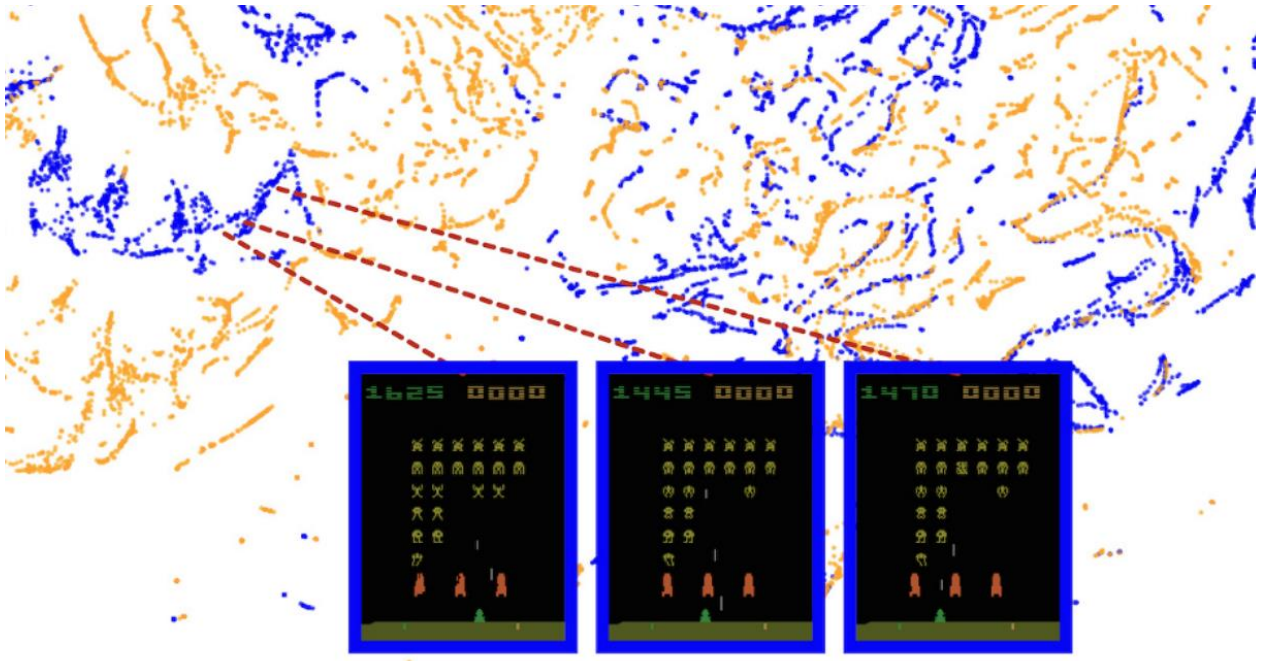


- Learning curve: (Space Invaders, ϵ-greedy with $\epsilon = 0.05$)



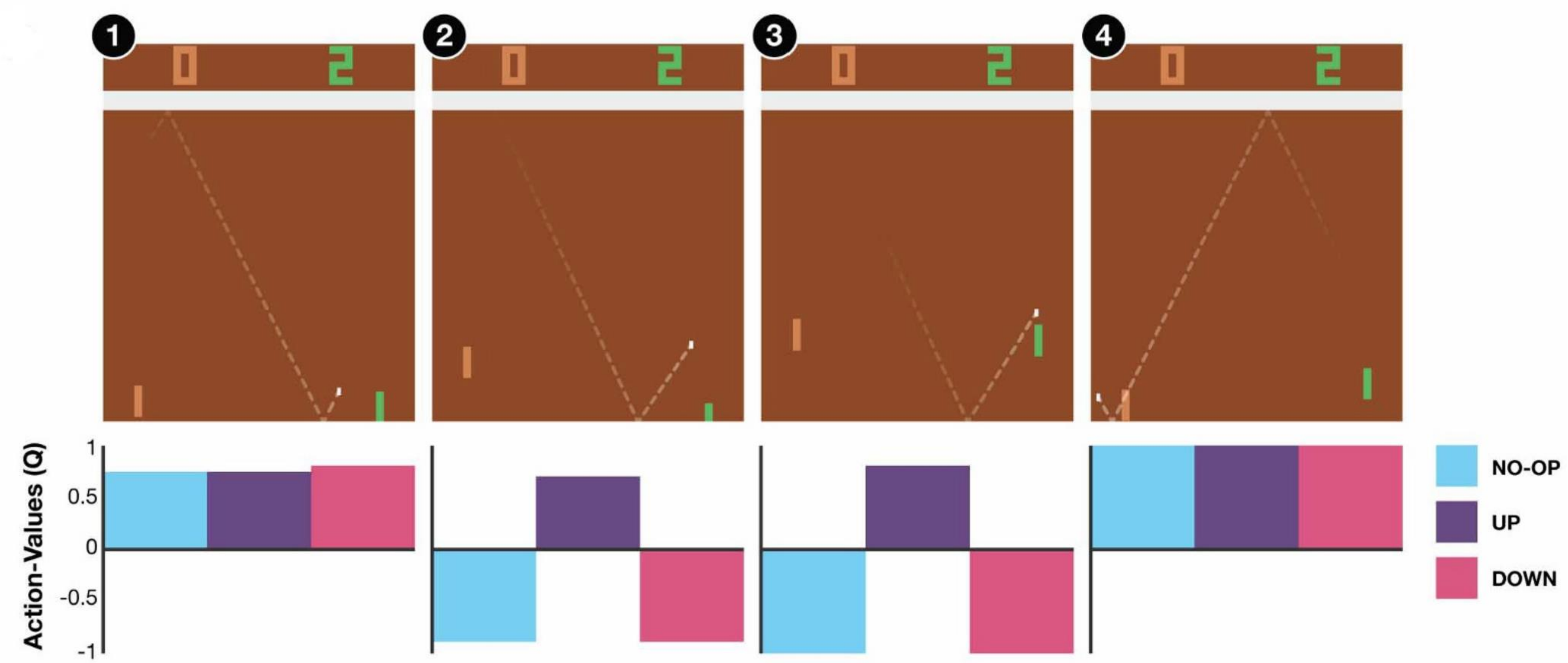
Fitted Q-learning example: Atari games (cont'd)

- Q-network's last hidden layer extracts useful representations
- Consequently Q-network provides Q-value estimates that generalize across states



Fitted Q-learning example: Atari games (cont'd)

- The learned Q functions are sensible



Summary

- MDPs: Reward driven philosophy
- Policy evaluation: Bellman consistency equations; fixed point iteration
- Planning in MDPs: value iteration; policy iteration
- Learning in MDPs: Q-learning; function approximation

Next lecture (12/5)

- Learning theory: fundamental limits of ML algorithms

- Assigned reading: Andrej Karpathy, Deep Reinforcement Learning: Pong from Pixels (We will have a reading quiz on this next time)