CSC 480/580 Principles of Machine Learning

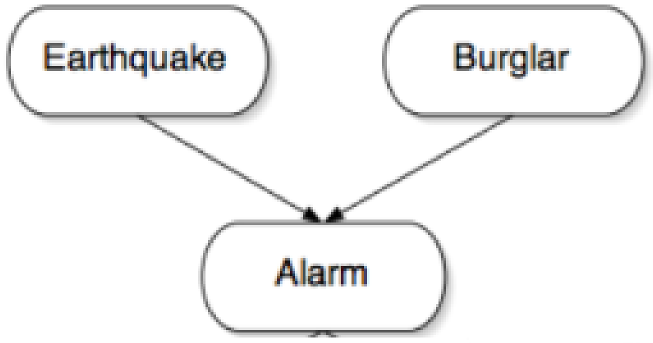
12 Reinforcement learning (RL)

Chicheng Zhang

Department of Computer Science

THE UNIVERSITY OF ARIZONA

HW3: a few comments



- What factorization of P(E, B, A) does this graph correspond to?
  - $P(E, B, A) = P(E)\, P(B)\, P(A|E, B)$
  - What does this equation mean, exactly?
  - For every $e, b, a \in \{0,1\}$,
    $$P(E = e, B = b, A = a) = P(E = e)\, P(B = b)\, P(A = a|E = e, B = b)$$
  (in total, 8 equalities)

  - Is $E \perp B \mid A$?
    - In fact, E and B are *negatively correlated* given $A = 1$

HW3: a few comments

- P3 (1) $x = (x_1, x_2)$ and $z = (z_1, z_2)$ are real vectors; let $K(x, z) = x_1 \cdot z_2$.

- A possible answer:
  - It is not a kernel, because we can find x, z such that $K(x, z) < 0$

- What is the problem with this answer?
  - Kernel functions do allow $K(x, z) < 0$!
  - Kernel functions don't allow $K(x, x) < 0$ though

HW3: a few comments

- P3 (2) x and z are integers between 0 and 100; let $K(x, z) = \min(x, z)$.

- A possible answer:
  - It is a kernel, because it satisfies positivity ($K(x,x) \geq 0$ for all x) and symmetry ($K(x,z) = K(z,x)$ for all x, z)

- What is the problem with this answer?
  - Positivity and symmetry are only necessary condition for a function to be a kernel, but not sufficient!
  - See a counterexample $K(x,z) = \max(x,z)$ in our lecture

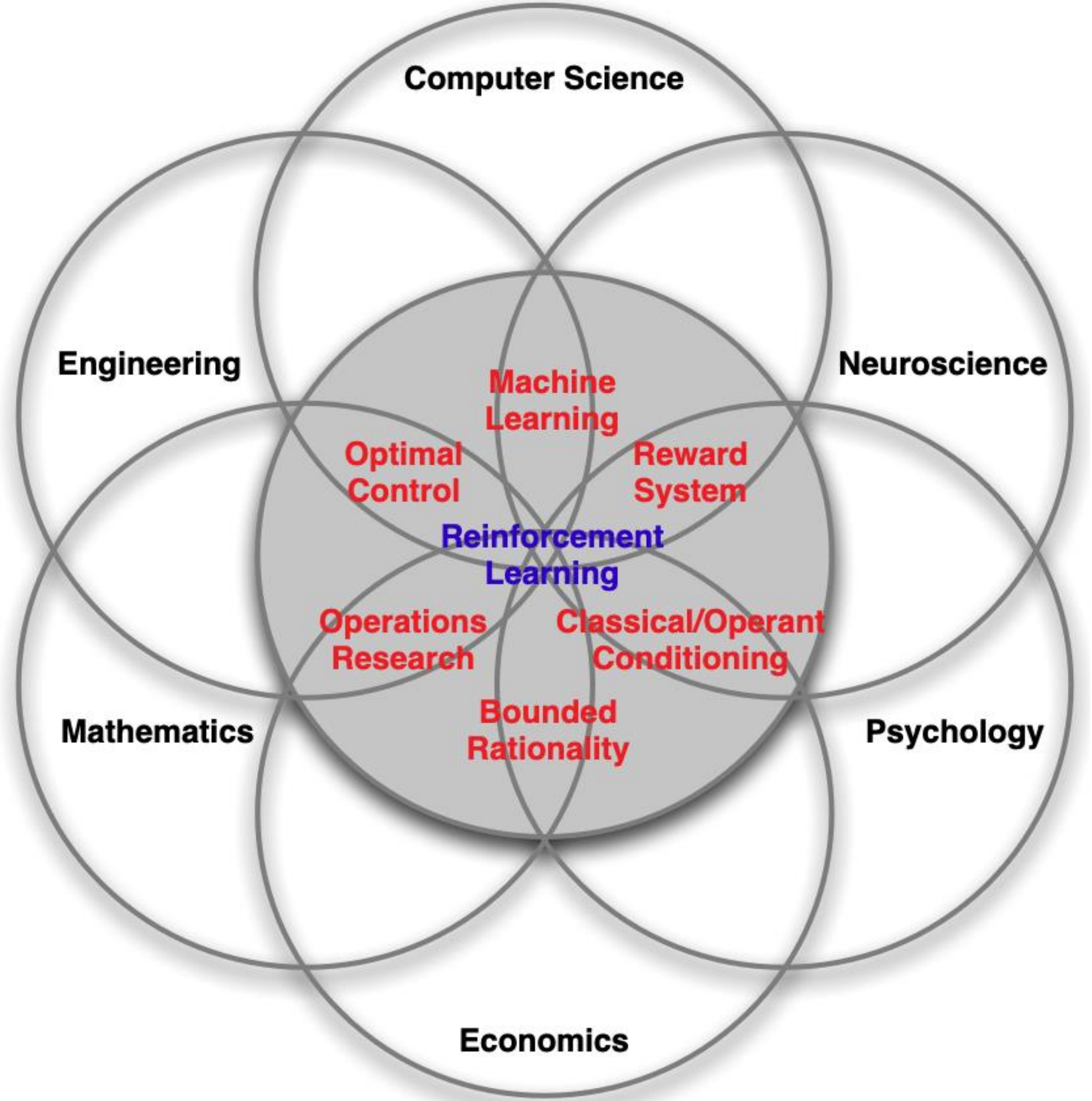
Reinforcement learning references

- ‘’Reinforcement learning’’ book by Sutton & Barto (available online)
- RL course by David Silver: https://www.youtube.com/watch?v=2pWv7GOvuf0&list=PLzuuYNsE1EZAXYR4FJ75jcJseBmo4KQ9-
- RL MOOC by Martha White and Adam White @UAlberta: https://www.coursera.org/specializations/reinforcement-learning

Outline

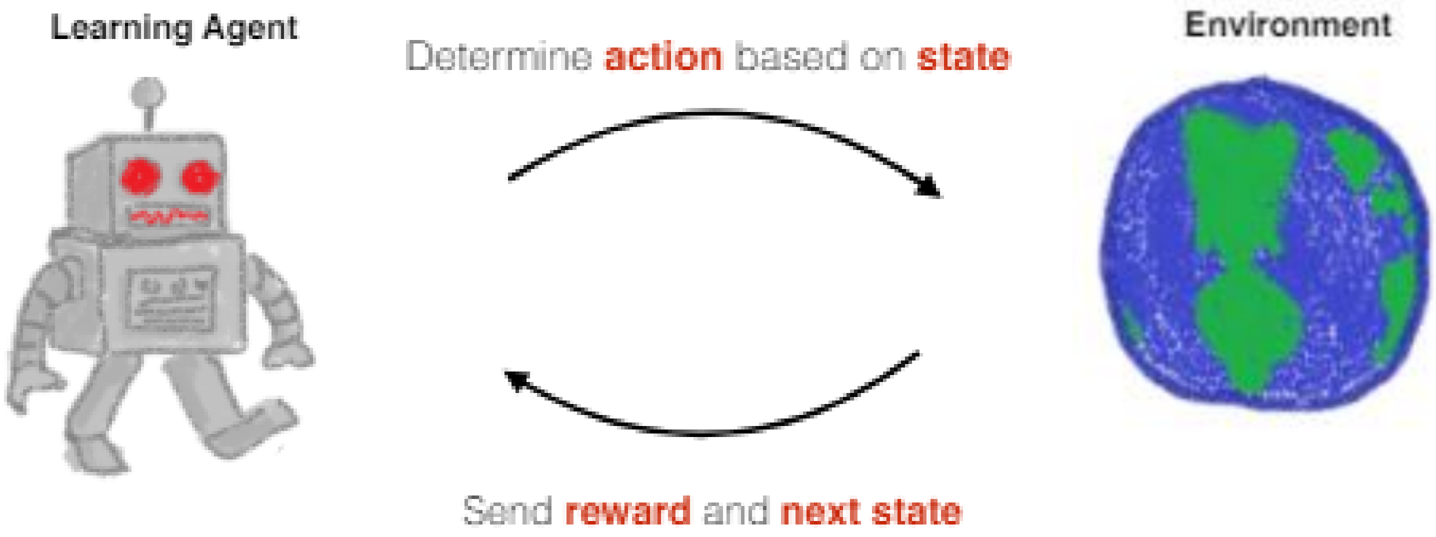
- Background / Markov Decision Processes (MDPs)
- Planning in MDPs
- Reinforcement Learning in MDPs

Background / Markov Decision Processes



Source: David Silver

Reinforcement Learning (RL)



- Applications:

Akshay Krishnamurthy & Wen Sun,
https://rltheorybook.github.io/colt21_part1.pdf

Characteristics of RL

How does RL differ from other ML frameworks?

- There is no supervisor, only a reward signal (evaluative vs. instructive feedback)
- Feedback is not instantaneous (delayed consequences)
- Data is not i.i.d. (it is sequential, time matters)
- The agent’s actions affect subsequent data it receives

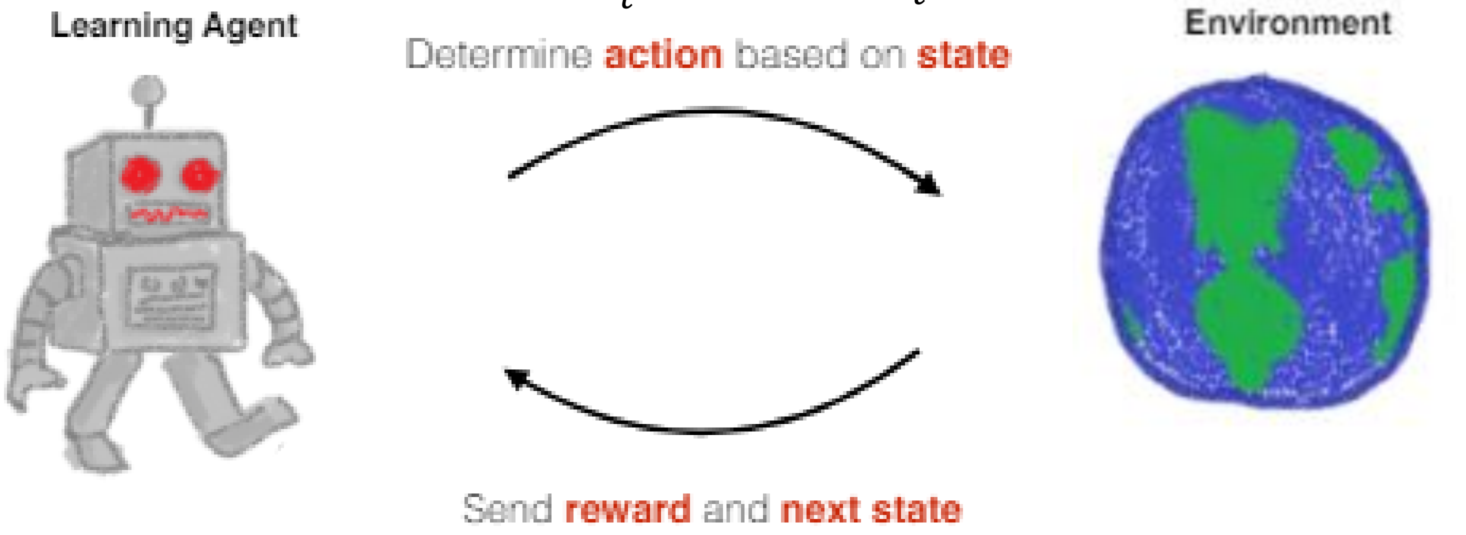
Source: David Silver

Examples of RL

- Fly stunt maneuvers in a helicopter (reward: not crashing)
- Manage an investment portfolio (reward: $)
- Play many different video games (reward: score)
- Make a humanoid robot walk (reward: distance traveled)
- Defeat world champion in Backgammon (reward: win/lose)
- Defeat world champion in Go! (reward: win/lose)

Markov Decision Process (MDP)

$a_t \in A$ $s_t \in S$

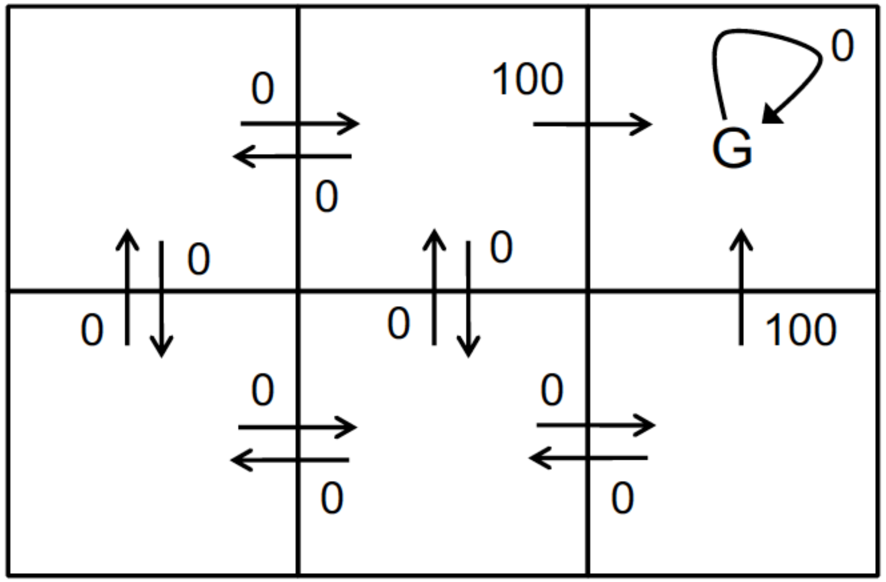


$r_t \in [0,1]$ $s_{t+1} \in S$

- Environment model $\mathcal{M}$
- Set of states S
- Set of actions A

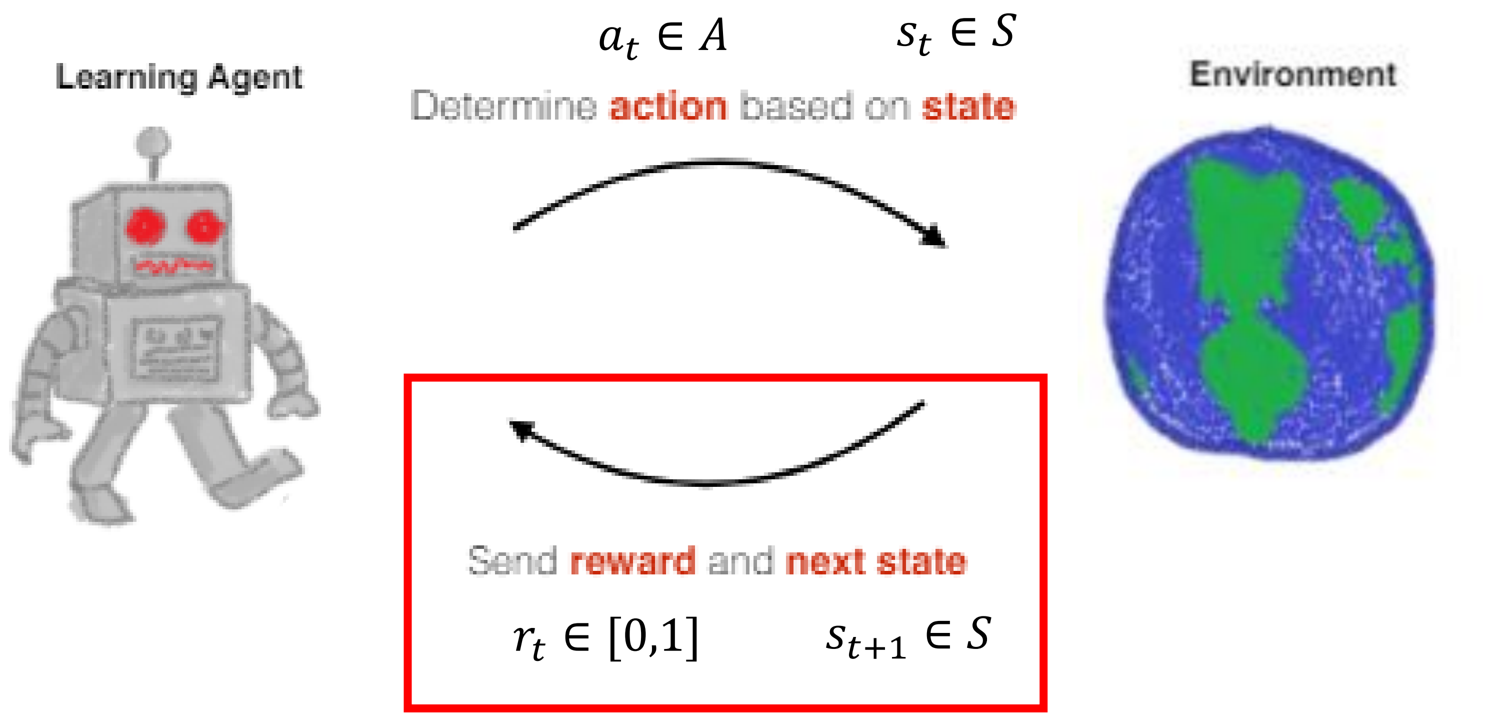
$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2}$$

Example: Learning to Navigate in the grid world



- State s: the location of the agent
- Each arrow represents an **action** a and the associated number represents deterministic **reward** $r(s, a)$
- How does the next state and current state relate to each other?

Markov Decision Process (MDP)



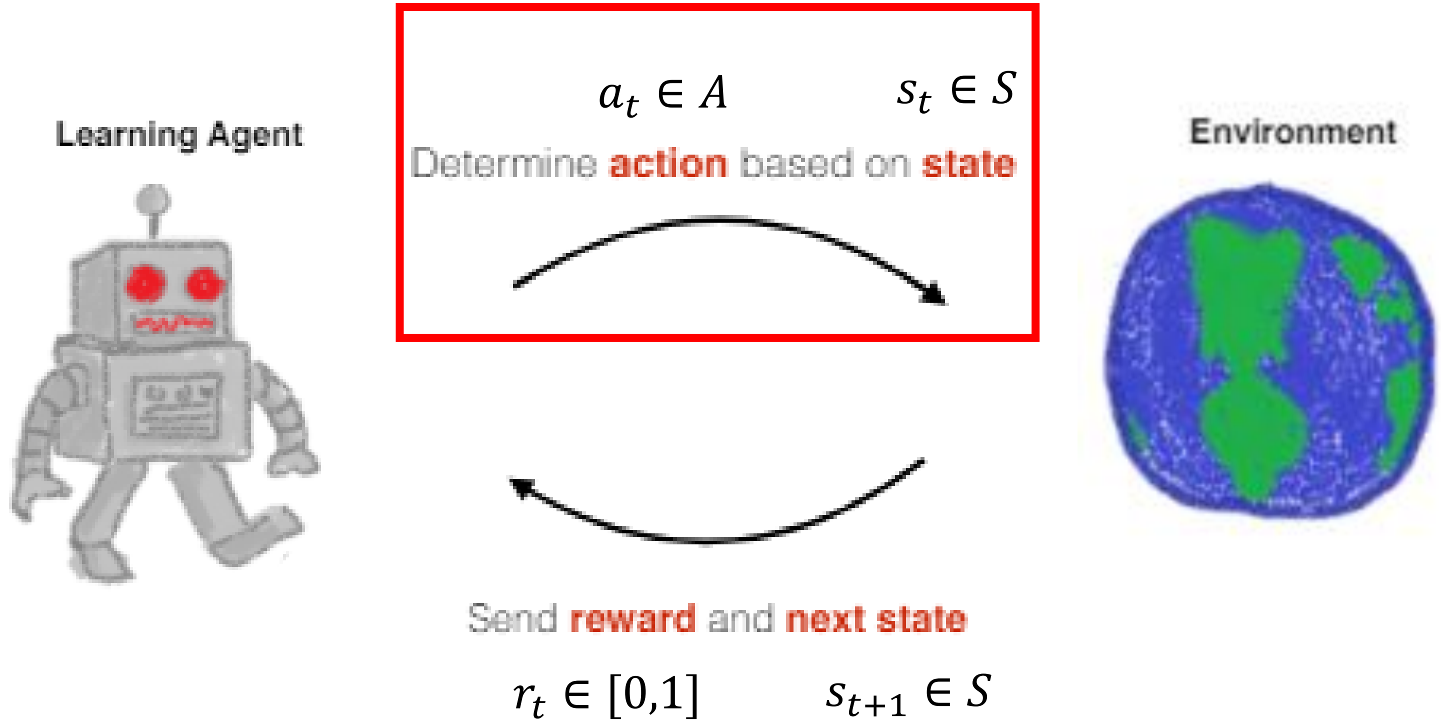
Markov assumption:

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

i.e. the future is independent of the past, given the present

These are unknown to the learner!

Markov Decision Process (MDP)



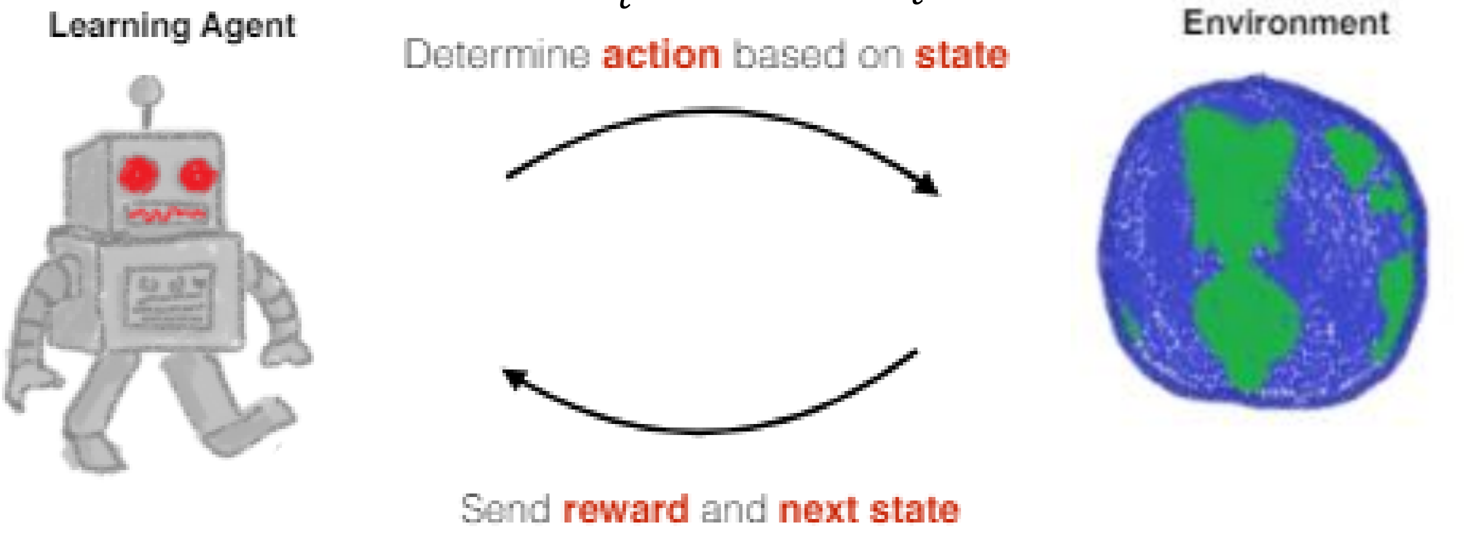
- A **policy** is the agent's behavior
- It is a mapping from state to action, e.g.
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a \mid s) = P(A_t = a \mid S_t = s)$

$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2}$$

- A policy, when interacting with MDP, generates a random *trajectory* $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$

Markov Decision Process (MDP)

$a_t \in A$ $\quad$ $s_t \in S$



$r_t \in [0,1]$ $\quad$ $s_{t+1} \in S$

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

$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2}$$

Summary: Specification of the environment

- Environment model MDP $\mathcal{M} = (S, A, R, P, \gamma)$
- R: a conditional probability table of current reward

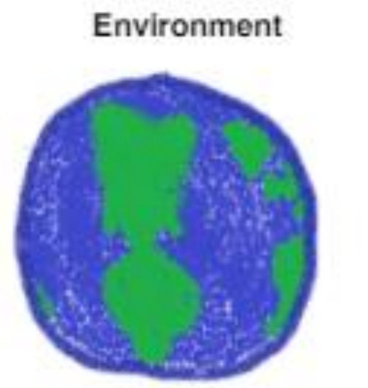
given current state & current action

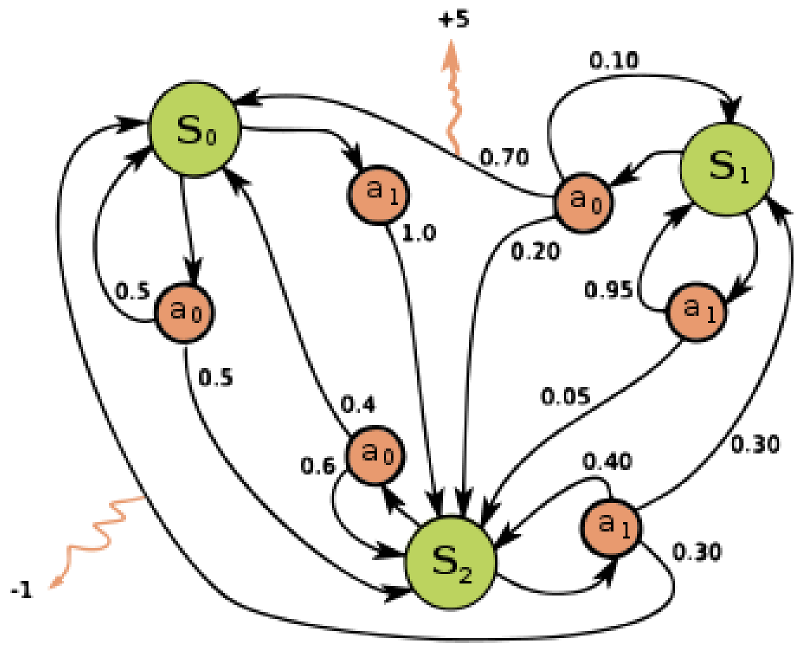
State s	Action a	Reward r	$R(r \mid s, a)$
S_1	a_0	$+5$	0.7
S_1	a_0	0	0.3
…			

- P: a conditional probability table of next state

given current state & current action

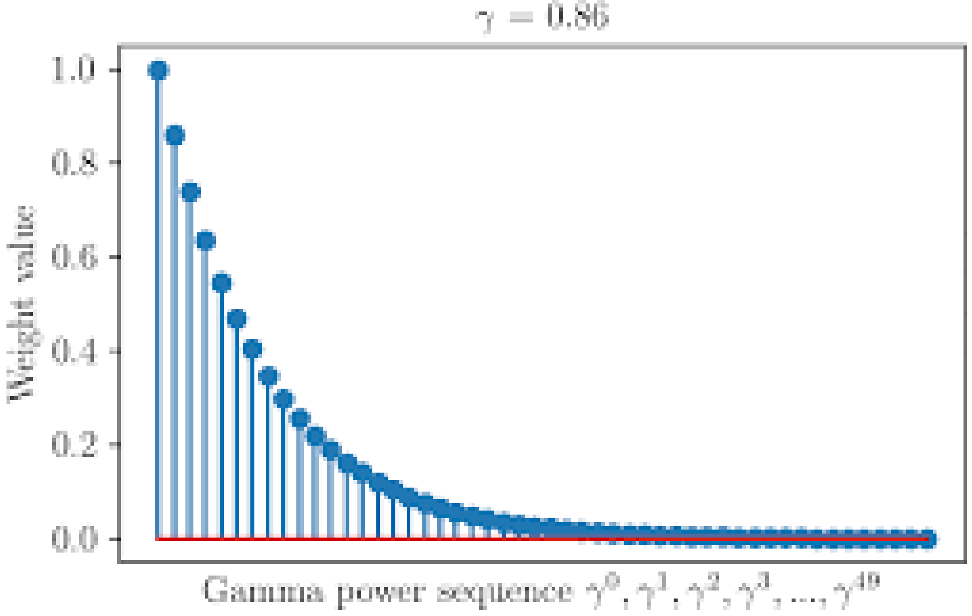
State s	Action a	Next state s'	$P(s' \mid s, a)$
S_1	a_0	S_0	0.7
S_1	a_0	S_2	0.2
…			





Discounted cumulative reward

- $R_0 = r_0 + \gamma\, r_1 + \gamma^2 r_2 + \cdots\,,\ 0 \le \gamma < 1$
- Discount: treating current reward as more worthy than future rewards
  - Larger $\gamma \Rightarrow$ focus more on longer term future



- Boundedness property:

$$0 \le R_0 \le 1 + \gamma + \gamma^2 + \cdots \le \frac{1}{1-\gamma}$$

- $\gamma = 1$: R_0 may diverge to $+\infty$
  - Maximize long-term average reward $\frac{1}{T}\sum_{t=1}^{T} r_t$

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

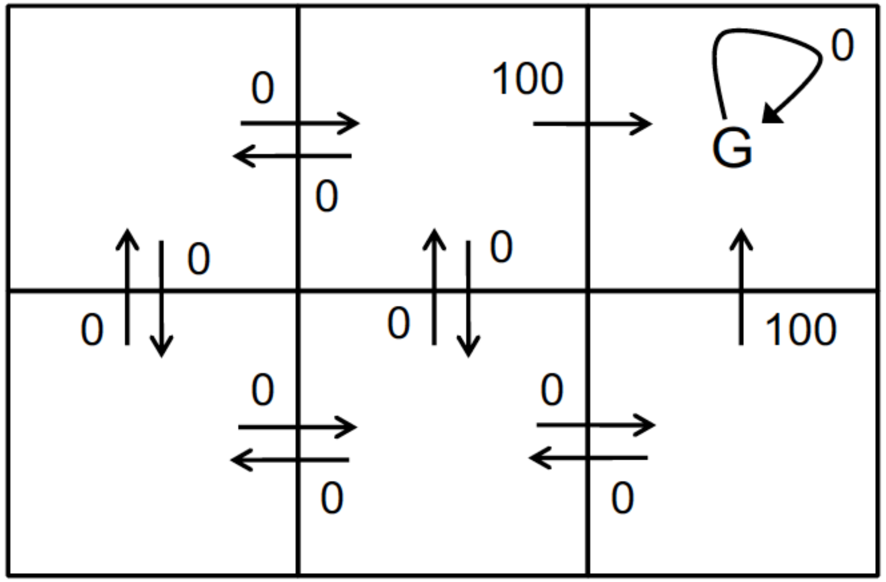
The Reward Hypothesis:
All goals can be described by the maximization of expected cumulative reward.

(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



- What do you think is the optimal behavior that maximizes reward?

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

- Prediction of future reward
- Used to evaluate goodness / badness of states given that the agent executes a policy π

$$V^{\pi}(s) = \mathbb{E}[\, r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \mid s_t = s, \pi \,]$$

- We explicitly notate that the value depends on the policy

Value function for a policy

- Important property (Bellman consistency equation):

$$V^\pi(s) = \underbrace{R(s,\pi(s))}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s'} P(s' \mid s,\pi(s))\, V^\pi(s')}_{\text{Expected Future reward}}, \forall s \in S$$

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

Justification:

$V^\pi(s) = \mathbb{E}[G_0 \mid s_0 = s, \pi]$ (definition)

$= \mathbb{E}[r_0 \mid s_0 = s, \pi] + \gamma \mathbb{E}[G_1 \mid s_0 = s, \pi]$ (return decomposition)

$= \mathbb{E}[r_0 \mid s_0 = s,\ a_0 = \pi(s)] + \gamma \mathbb{E}[V^\pi(s_1) \mid s_0 = s,\ a_0 = \pi(s)]$ (iterated expectation)

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

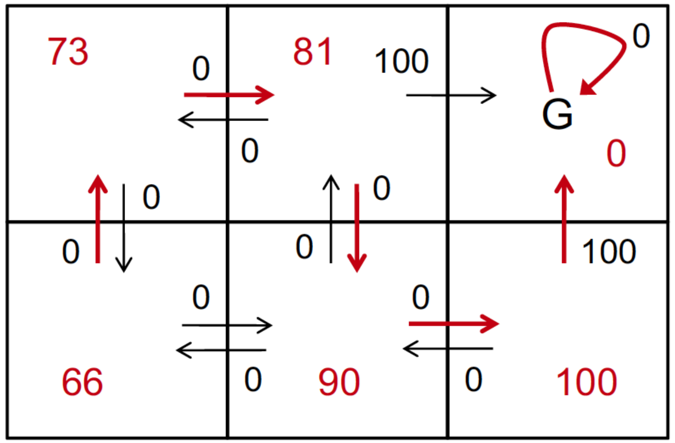
Optimal policy

- Fact: there is a 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

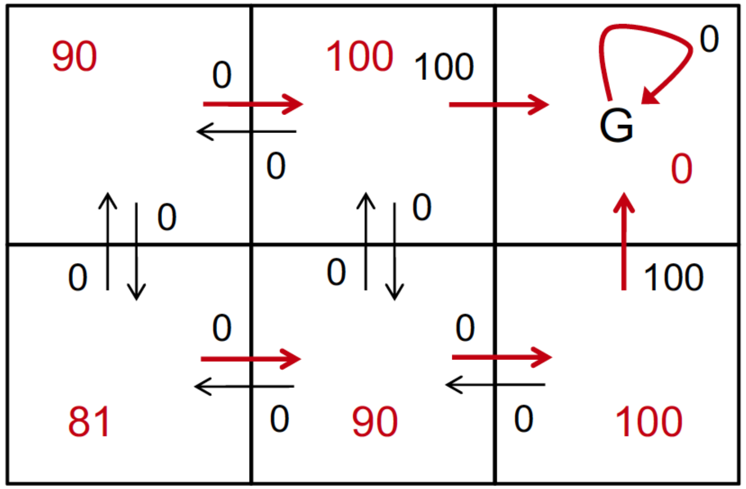
Value function for a policy π

- 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:

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

- How many equations and how many unknowns?
- 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?
  - E.g. Gaussian elimination
  - Alternatively, use *fixed-point iteration*: $V^{k+1} \leftarrow R^\pi + \gamma M^\pi V^k$

Planning in MDPs

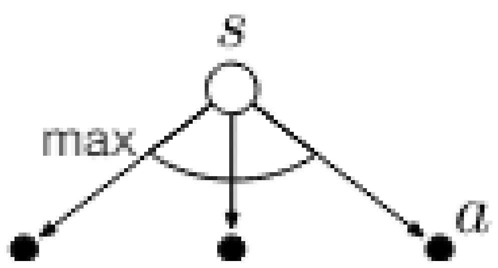
Planning in MDPs

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

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

$$\pi^*(s) = \arg\max_{a \in A} \; R(s,a) + \gamma \textstyle\sum_{s \in S} P(s'|s,a)\, V^*(s)$$

Expected reward if acting optimally subsequently

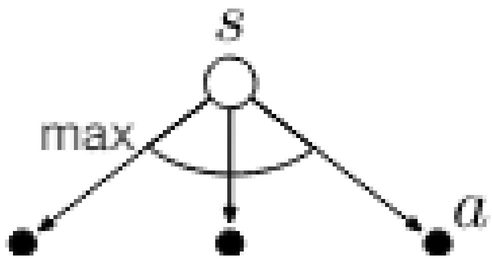


- How to find $V^*(s)$?

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:
  - Optimal behavior = optimal action a + optimal behavior afterwards

- Issue: Bellman optimality equation **is not a linear system**
- However, V^* can be seen as a *fixed point*

First Algorithm: Value iteration

Key idea: perform fixed point iteration on Bellman optimality equation

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

Initialize $V(s)$ arbitrarily

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

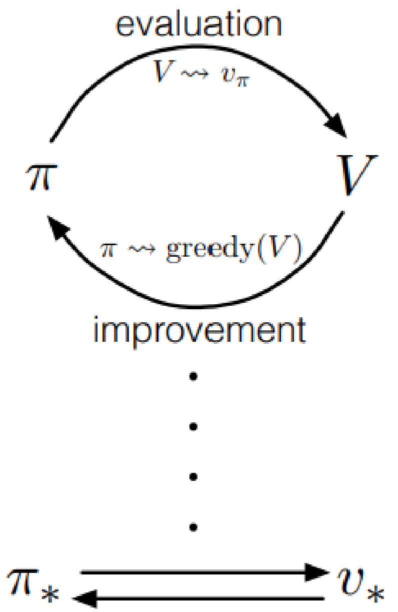
- **For** each $s \in S$:
  - $V(s) \leftarrow \max_a \; R(s,a) + \gamma \sum_{s'\in S} P(s'|s,a) \cdot V(s')$

- Fact: With about $O\left(\frac{1}{1-\gamma} \ln \frac{1}{\epsilon}\right)$ iterations, V becomes ϵ-close to V^*

Second Algorithm: Policy iteration

- The idea:

estimate optimal value V^* and optimal policy π^* *simultaneously & iteratively*



- Observe:
  - π^* is greedy wrt V^*, i.e.,

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

  - V^* is the value function of π^*

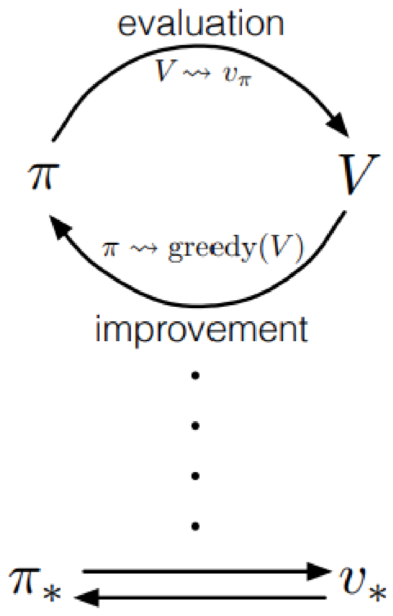
- Can we obtain a pair (π, V) that exhibit the above properties?

Second Algorithm: Policy iteration

Algorithm:

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

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



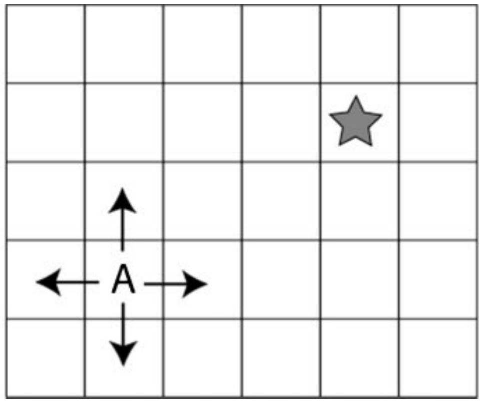
- Fact: With about $O\left(\frac{1}{1-\gamma} \ln \frac{1}{\epsilon}\right)$ iterations, V becomes ϵ-close to V^*

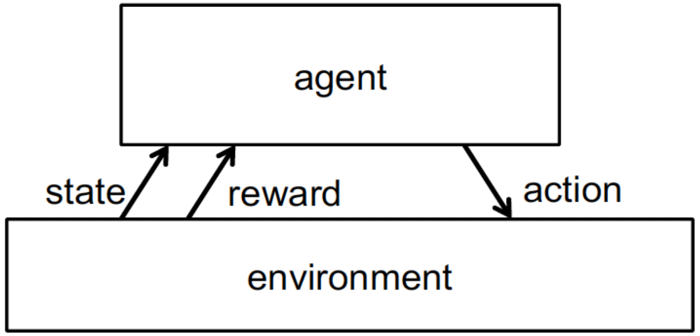
Summary

- 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



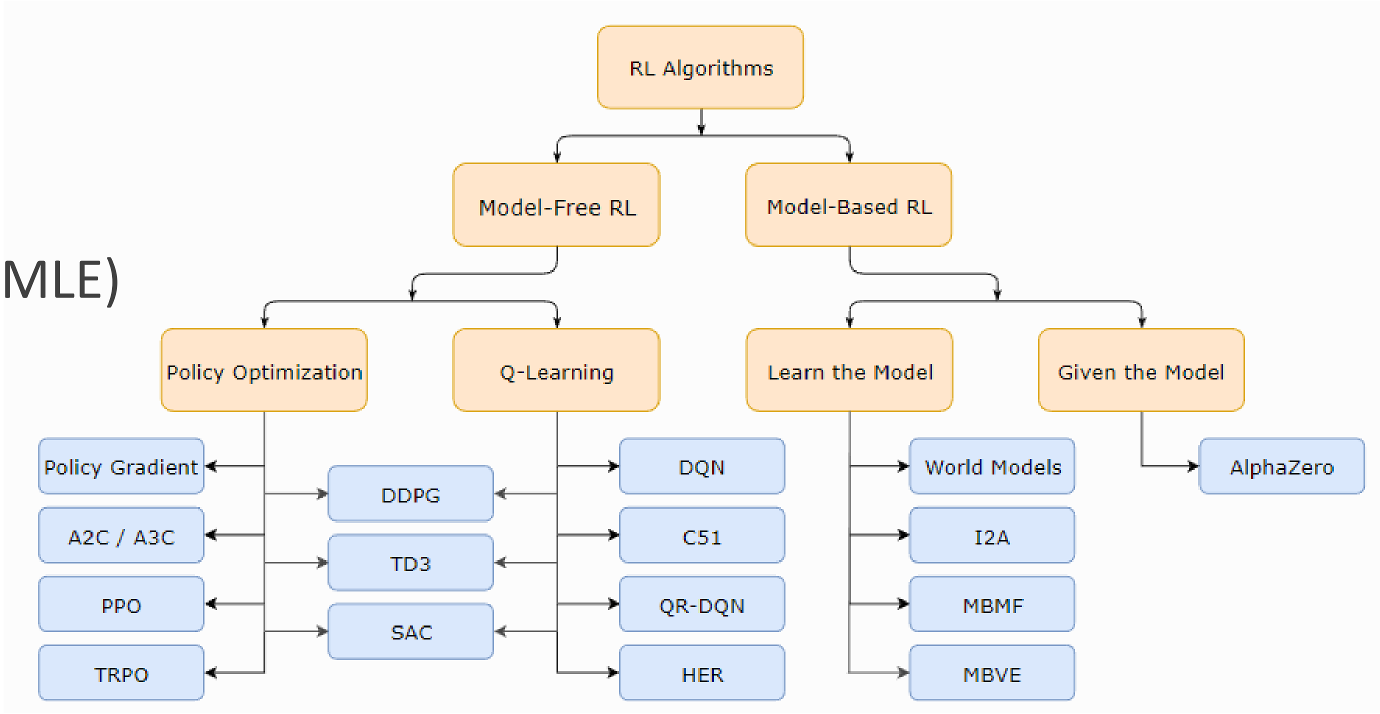


- Given:
  - MDP $\mathcal{M}$ (unknown)
  - The ability to interact with $\mathcal{M}$ for T steps
    - Obtaining trajectory $s_0, a_0, r_0, \ldots, s_T, a_T, r_T$
- Goal:
  - (Online learning) maximize cumulative reward $\mathbb{E}[\sum_{t=0}^{T} \gamma^t \, r_t]$
    - Useful in applications where every action taken has 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

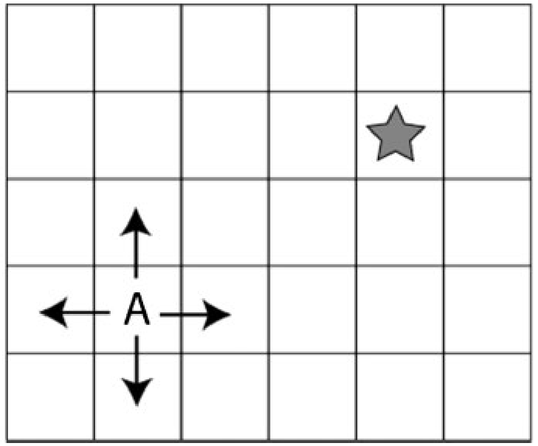
https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

Unique challenges in RL I: Temporal Credit Assignment

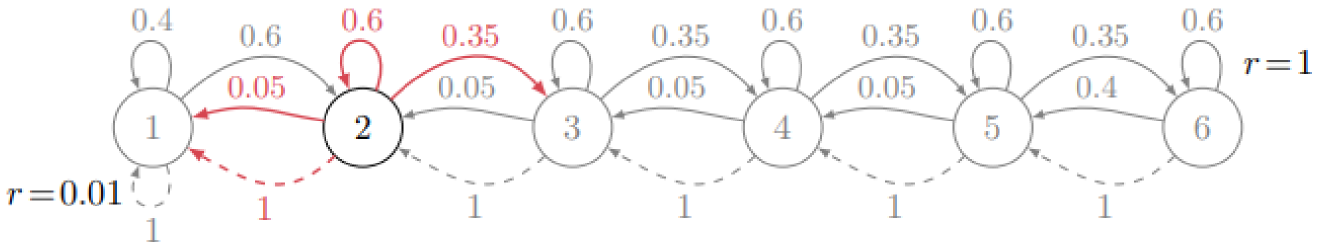
- Performance measure:
  - focuses on the quality of *a sequence of interdependent states / actions*
- Aim for maximization of *long-term rewards*

- E.g.
  - Daily exercise: short term – long term ++
  - Stay up all night playing video games: short term + long term --
  - Chess tactics: sacrifice pieces
- Need to answer questions like: “what is the key step that caused me to lose this game?” – temporal credit assignment

Unique challenges in RL II: Exploration



- Learning agent's data is induced by its own actions
- How to collect *useful* data?
    - The exploration challenge
- Rough intuition: collect data that "covers" all states and actions
    - ϵ-greedy exploration: w.p. ϵ, take actions uniformly at random
    - $\epsilon = 1$: uniform exploration
- 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

Learning Q-functions

- 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):

$$Q^{\pi}(s,a) = \mathbb{E}[\, G_0 \mid s_0 = s, a_0 = a,\ \pi \,] = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi}(s')$$

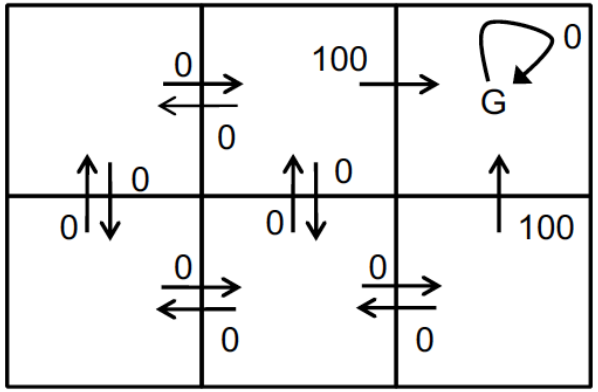
The optimal Q function

$$Q^{*}(s,a) = \mathbb{E}[\, G_0 \mid s_0 = s, a_0 = a,\ \pi^{*} \,] = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{*}(s')$$

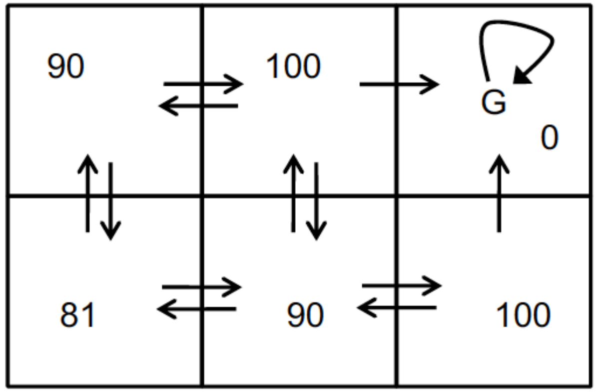
The optimal policy can be extracted from Q^*:

$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

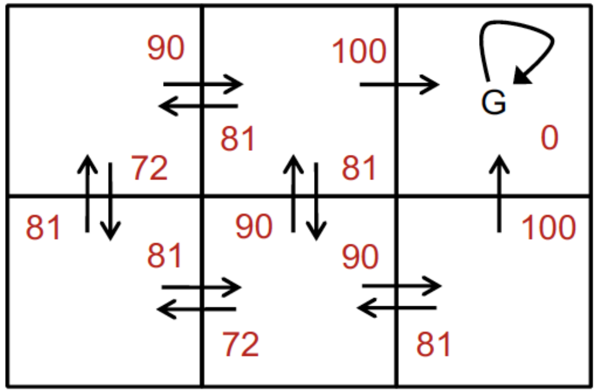
Q-values



$r(s, a)$ (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^*

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

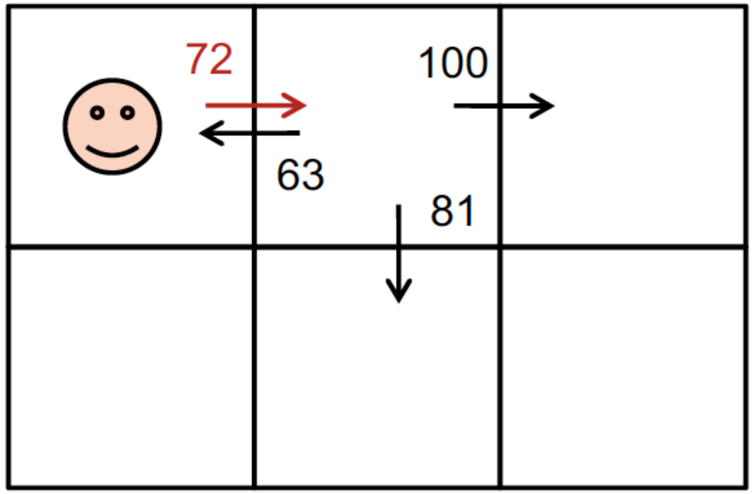
- We will use this to design our learning rule

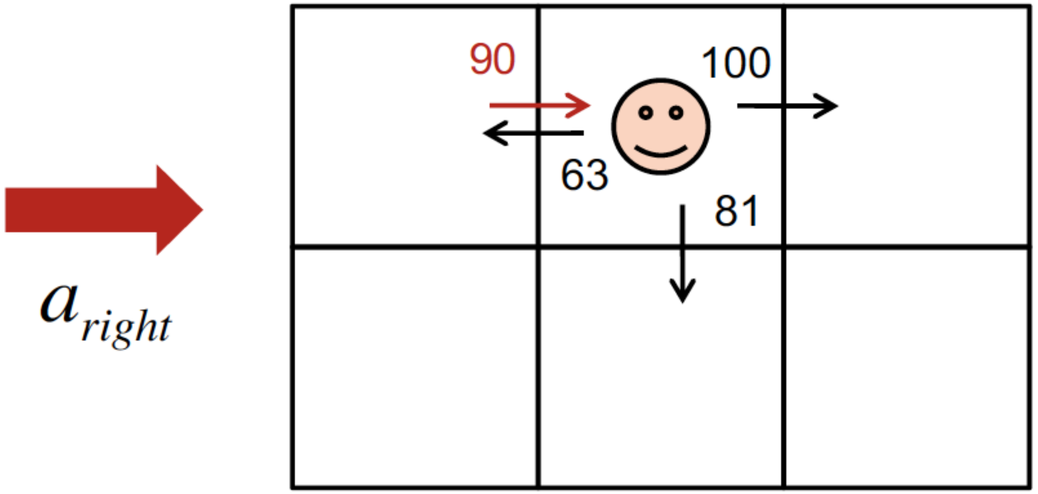
Algorithm: Q-learning (deterministic transitions/rewards)

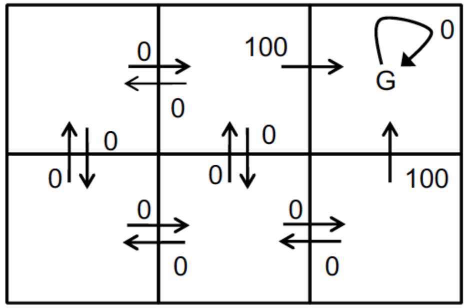
- Assume that we are in the tabular setting: S and A are both finite
- Initialize: $Q(s, a) = 0, \forall s, a$
- Observe the initial state s
- Repeat:
  - Select an action a and execute it (e.g., ϵ-greedy)
  - Receive a reward r
  - Observe a new state s'
  - Update: $Q(s, a) \leftarrow r + \gamma \max_{a'}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

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

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

$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2}$$

- How to fix this? Recall:

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

- 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 $X_1, \ldots, X_n \sim N(\mu, 1)$
- 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 $\hat{\mu}_{n-1} = \frac{1}{n-1}(X_1 + \cdots + X_{n-1})$
- This is equivalent to $\hat{\mu}_n = \underbrace{(1-\alpha_n)\hat{\mu}_{n-1}}_{\text{Old estimate (conservativenss)}} + \underbrace{\alpha_n X_n}_{\text{New data (correctivenss)}}$, where $\alpha_n = \frac{1}{n}$

Q-learning for Stochastic Transitions / Rewards

- Initialize: $Q(s,a) = 0, \forall s, a$
- Observe the initial state s
- Repeat
  - Take an action a
    - e.g., ϵ-greedy (taking $\mathrm{argmax}_a Q(s,a)$ w.p. $1-\epsilon$)
  - Receive the reward r
  - Observe the new state s'
  - Update: $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a')\right)$
  - $s \leftarrow s'$

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

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

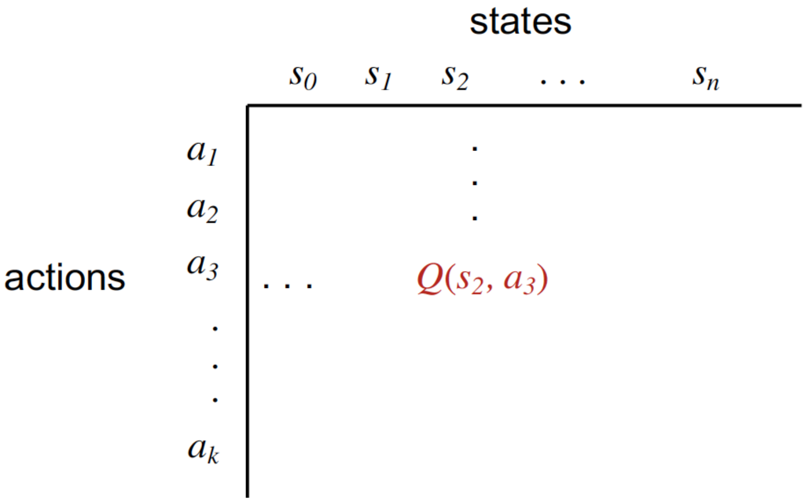
- Q-learning will converge to the optimal Q function (under certain niceness assumptions on the MDP, exploration policy, and step size scheme)

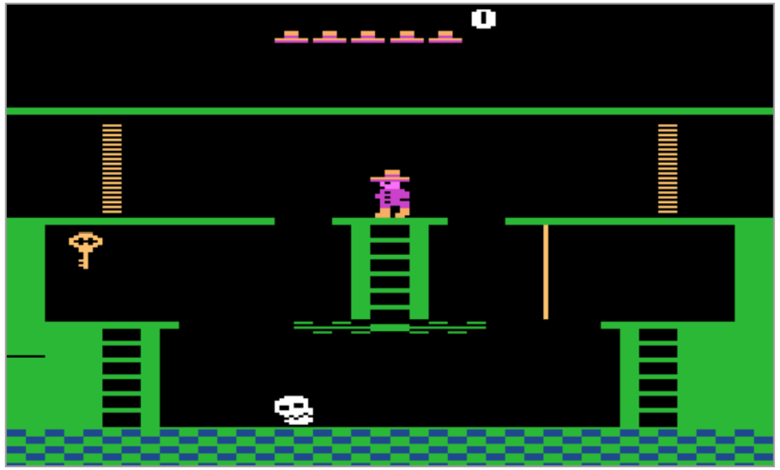
- In practice, it takes a lot of iterations!

- 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

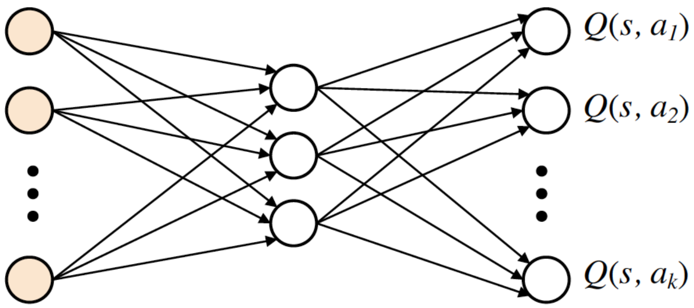
- Q-learning requires us to maintain a huge table, which is clearly infeasible with large state spaces





- How to design a Q-learning-style algorithm that can handle large state spaces?
- Idea: use a neural network to represent Q

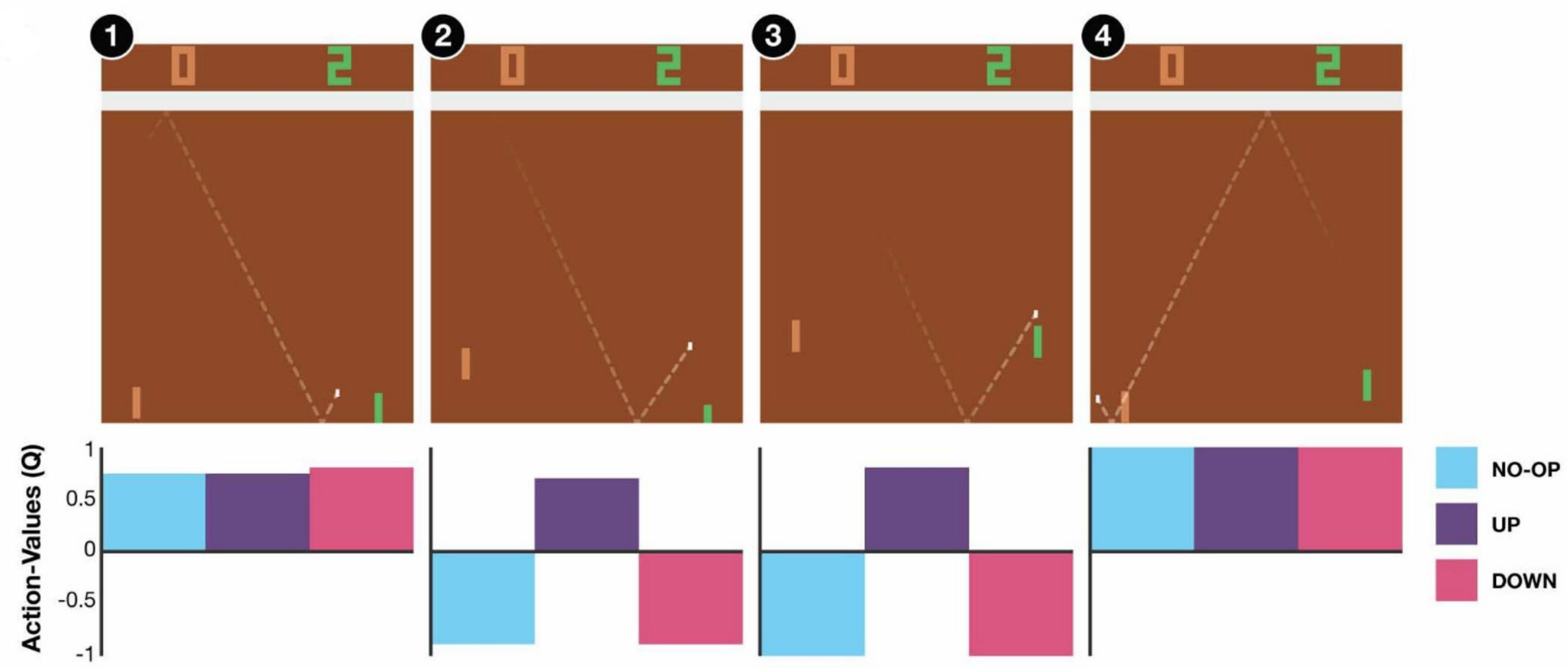
and learn the weights of the network (fitted-Q learning)



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

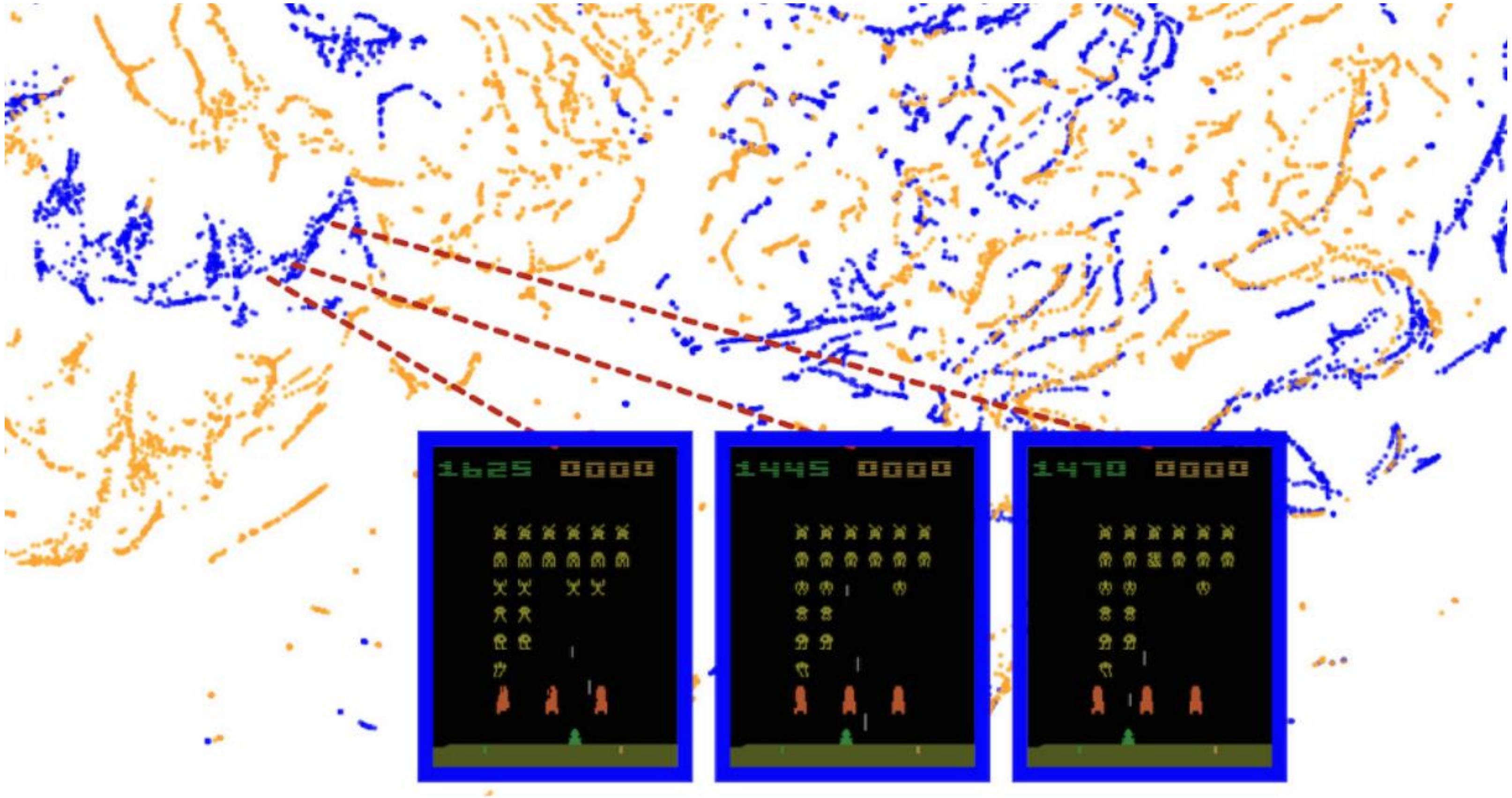
Fitted Q-learning example: Atari games

- The learned Q functions are sensible



Fitted Q-learning example: Atari games

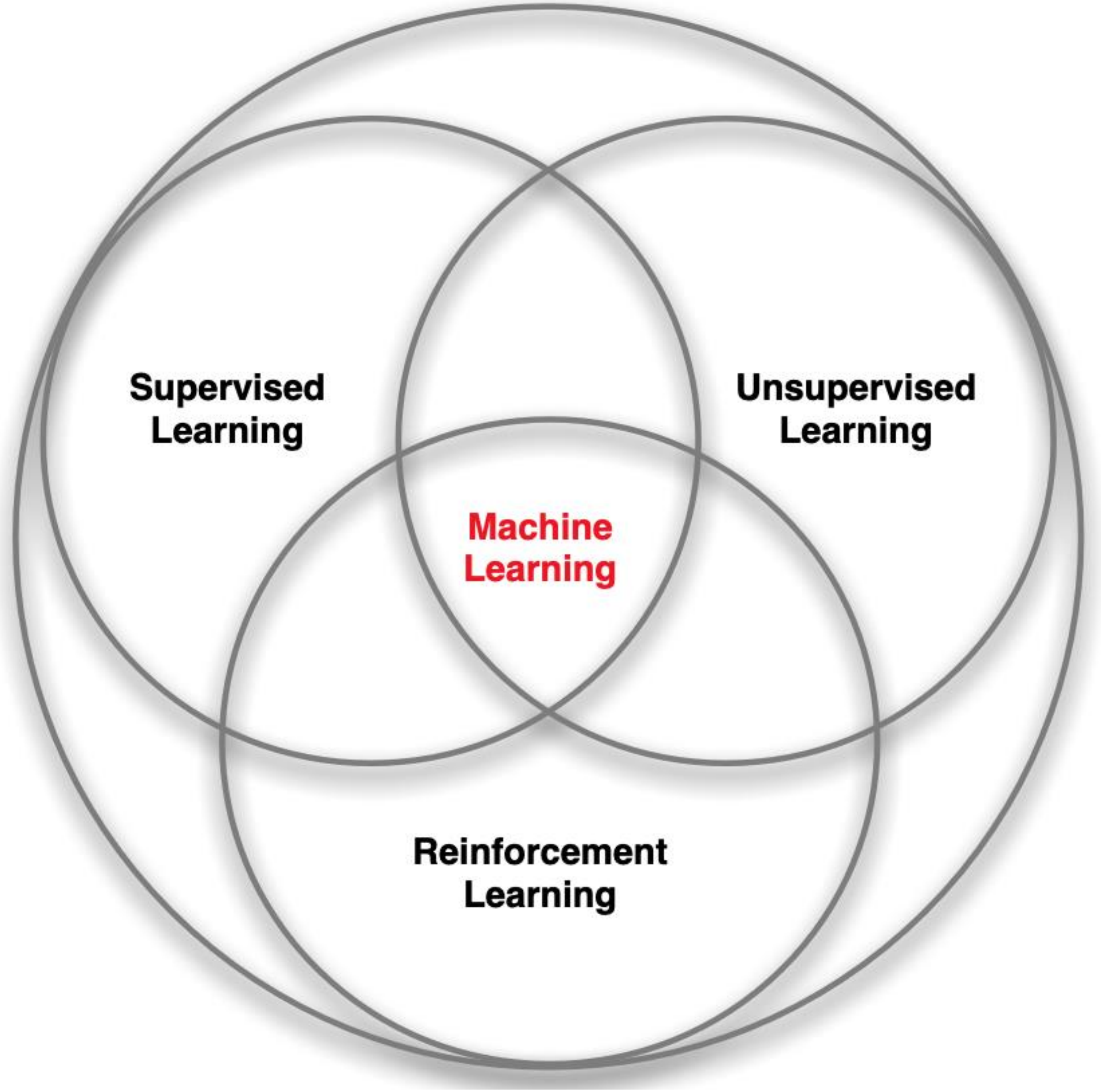
- Q-network's last hidden layer extracts useful representations
- Consequently Q-network provides Q-value estimates that generalize across states



Summary

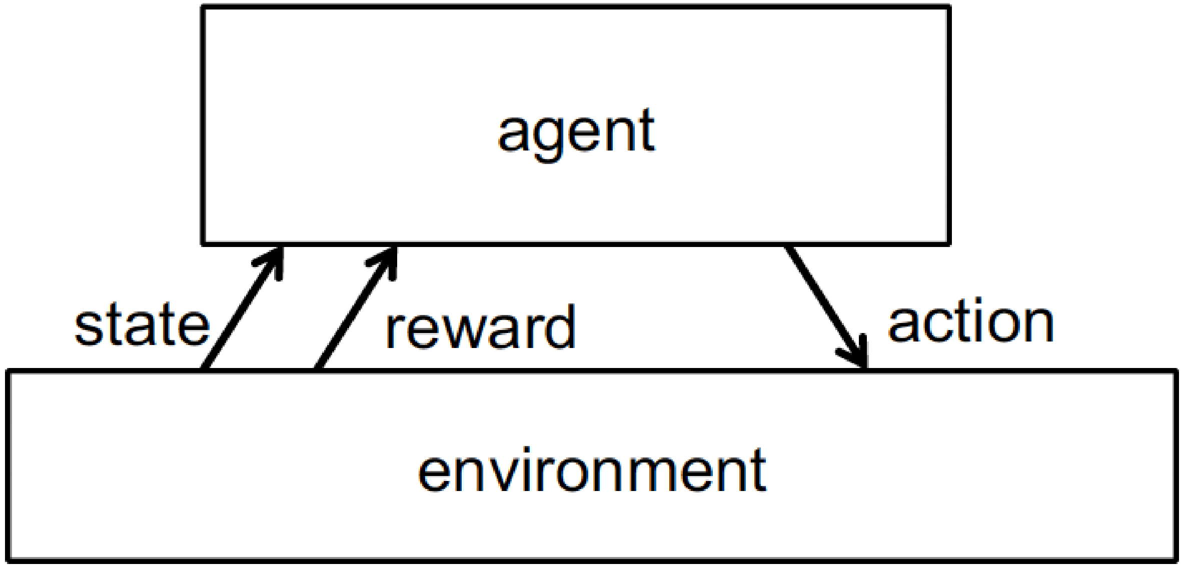
- MDPs: Reward driven philosophy
- Policy evaluation: Bellman consistency equations; fixed point iteration
- Planning in MDPs: value iteration; policy iteration
- Learning in MDPs: Monte Carlo learning, Q-learning; function approximation

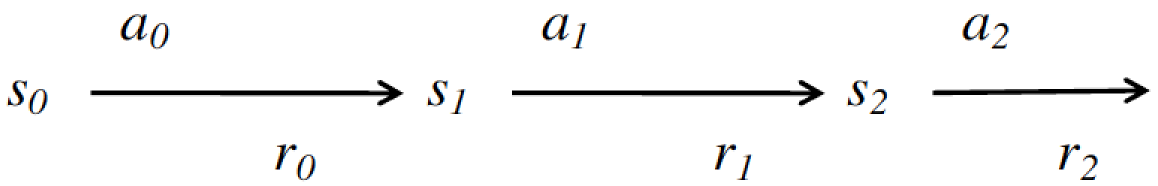
Backup



Source: David Silver

Markov Decision Process (MDP)



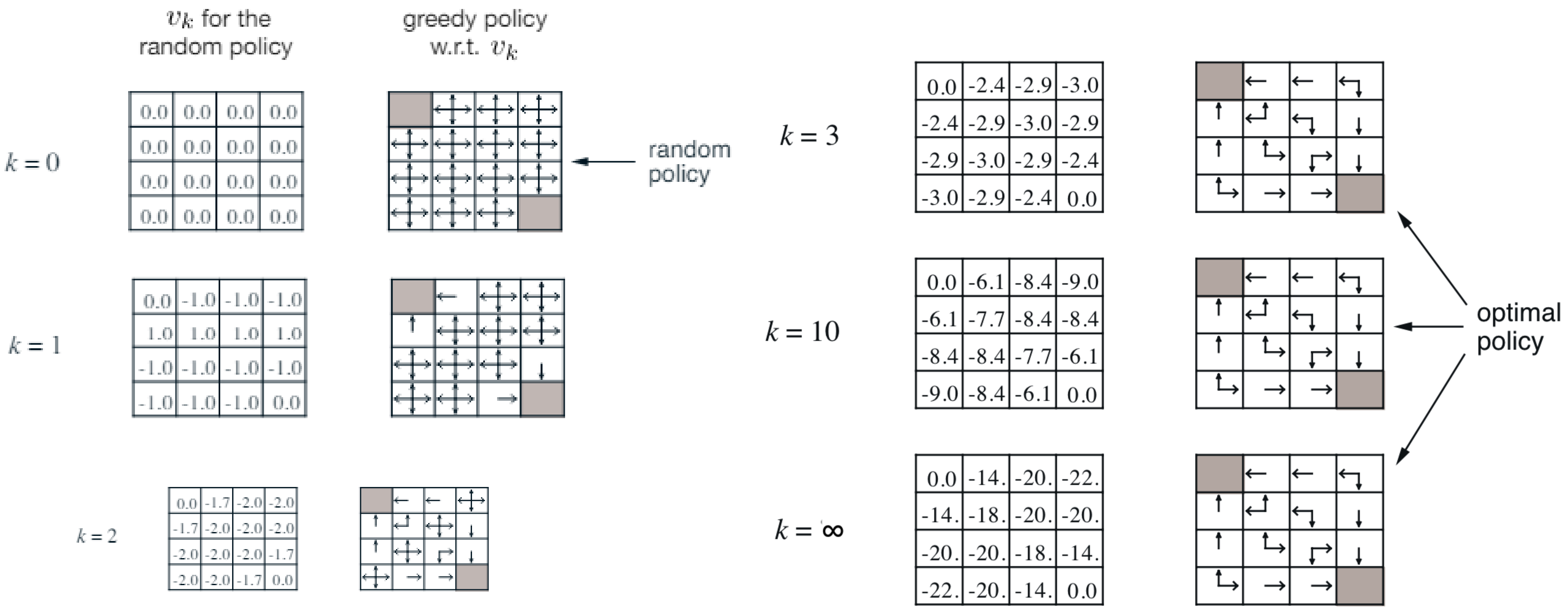


- Environment model $\mathcal{M}$
- Set of states S
- Set of actions A
- at each time t, agent observes state $s_t \in S$, then chooses action $a_t \in A$
- then receives a reward r_t and moves to state s_{t+1}; repeat.

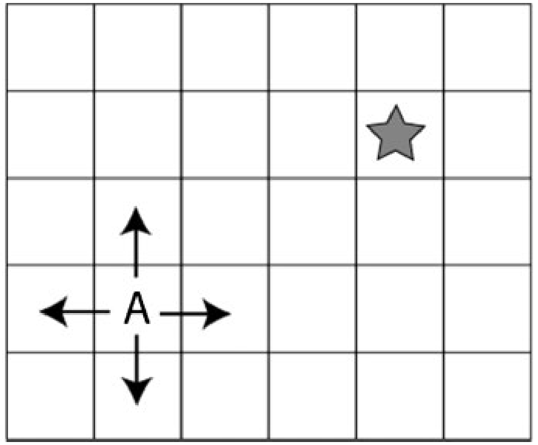
Policy iteration: an interesting observation

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

what you get if you apply the policy improvement step

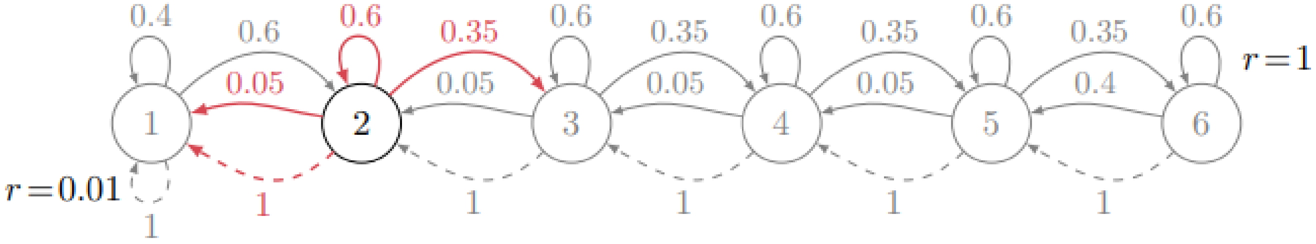


Unique challenges in RL II: Exploration



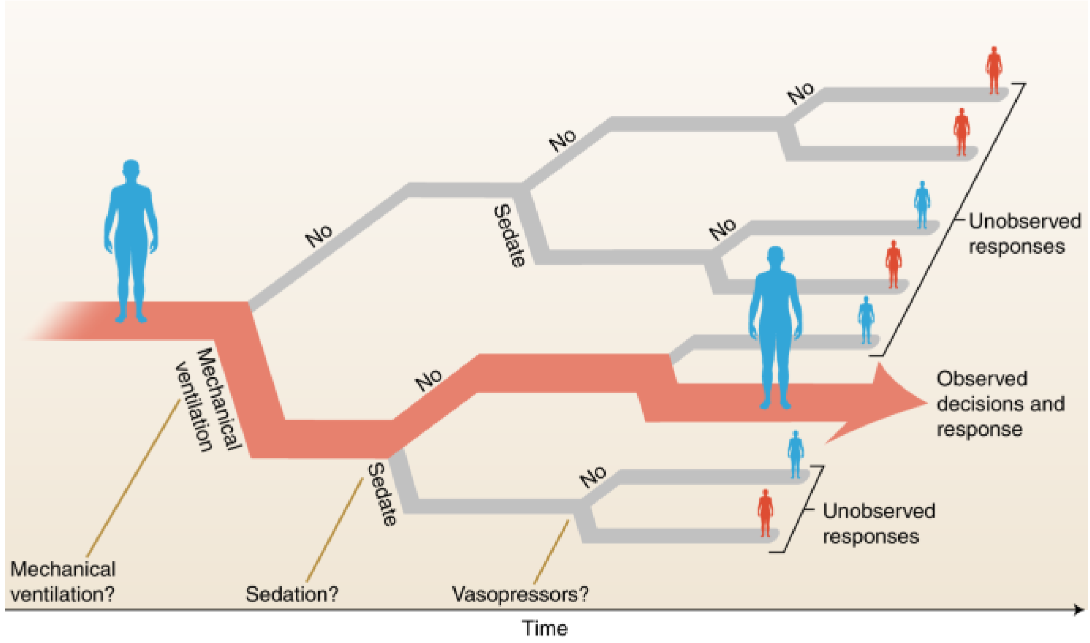
- Learning agent's data is induced by its own actions
- How to collect *useful* data?
  - The exploration challenge

- 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]



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

Monte Carlo Reinforcement Learning

- MC methods learn directly from episodes of experience
- MC is *model-free*: no knowledge of MDP transitions / rewards
- MC learns from complete episodes (no bootstrapping)
- MC uses the simplest idea: value = mean return
- Caveat: Can only apply MC to episodic MDPs (must terminate)

Credit: David Silver

Monte Carlo Reinforcement Learning

Goal: learn V^π from episodes of experience under policy π:

$$S_1, A_1, R_2, \ldots, S_k \sim \pi$$

Recall that *return* is total discounted reward:

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots$$

And recall that the *value function* is expected return:

$$V^\pi(s) = E_\pi[\, G_t \mid S_t = s \,]$$

MC policy evaluation uses *empirical mean* return instead of *expected return*

Credit: David Silver

First-Visit MC Policy Evaluation

- To evaluate *s*
- The **first** time-step t that s is visited in an episode
- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Estimate value by mean return $V(s) \leftarrow S(s)/N(s)$
- By the law of large numbers $V(s) \rightarrow V^{\pi}$ as $N(s) \rightarrow \infty$

Credit: David Silver

Every-Visit MC Policy Evaluation

- To evaluate s
- **Every** time-step t that s is visited in an episode
- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Estimate value by mean return $V(s) \leftarrow S(s)/N(s)$
- Again, $V(s) \rightarrow V^{\pi}$ as $N(s) \rightarrow \infty$

Credit: David Silver

Example: Blackjack

Objective: Have your card sum be greater than the dealer's without going over 21

States (200 of them)

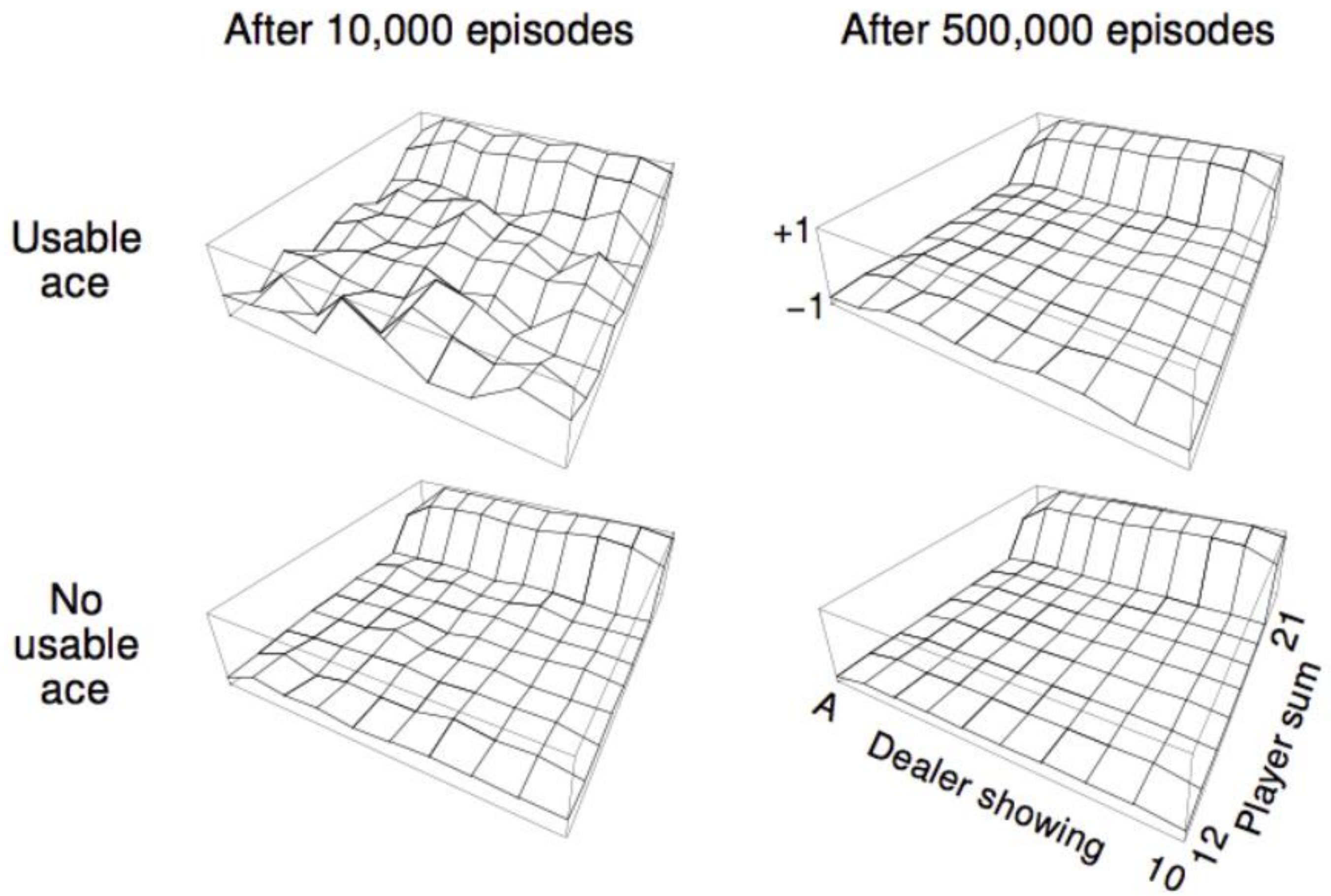
- Current sum (12-21)
- Dealer's showing card (Ace-10)
- Do I have a useable ace?

Reward +1 for winning, 0 for draw, -1 for losing

Actions Hold (stop receiving cards), Hit (receive another card)

Credit: David Silver

Example: Blackjack



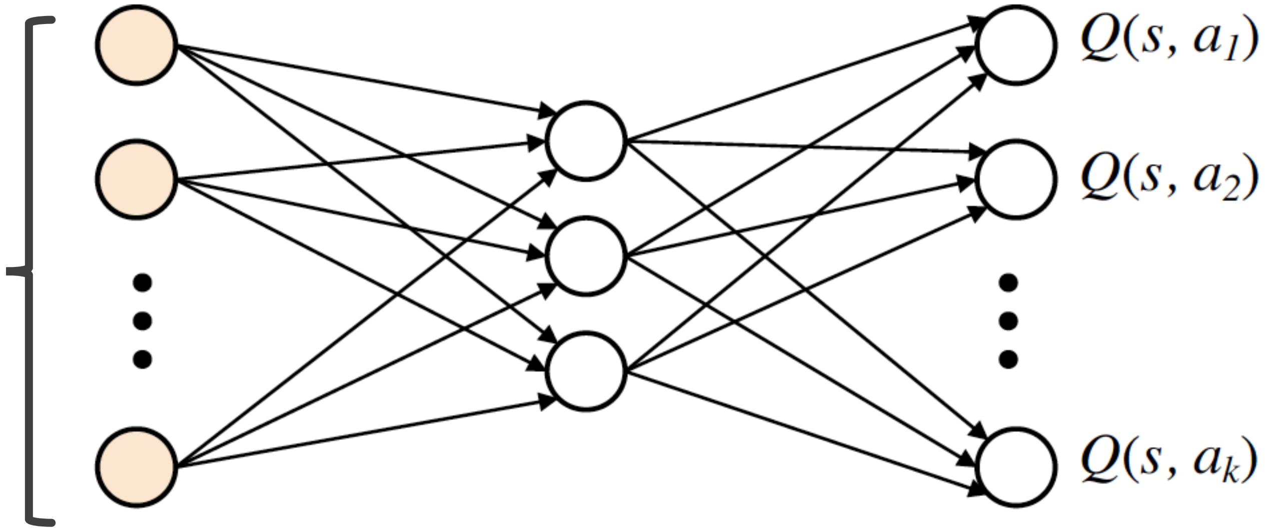
Policy Hold if sum at least 20, otherwise hit

Credit: David Silver

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



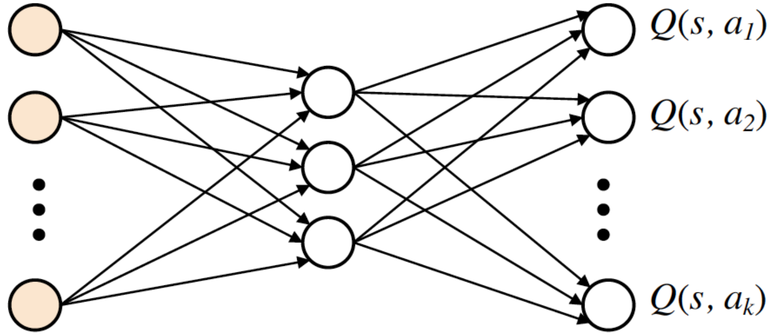
each input unit can be a sensor value
(or more generally, a feature)

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

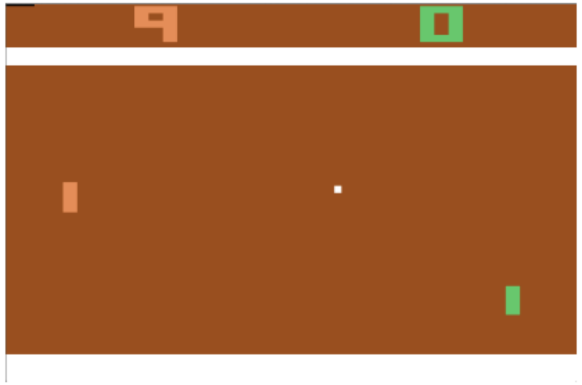
Repeat

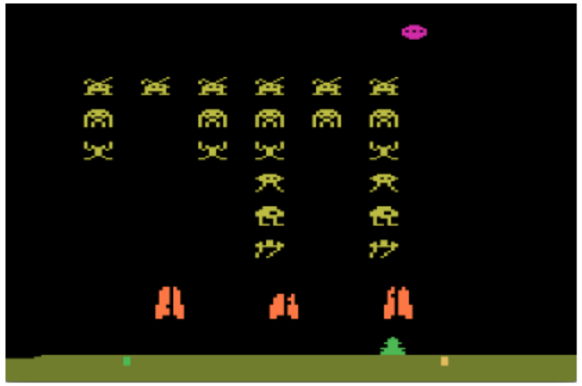
- observe the state s
- compute $Q(s, a)$ for each action a (forward pass on the NN)
- select action a (e.g. use ϵ-greedy) and execute it
- observe the new state s' and the reward r
- compute $Q(s', a')$ for each action a' (forward pass on the NN)
- update the NN with the instance
  - $x \leftarrow s$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



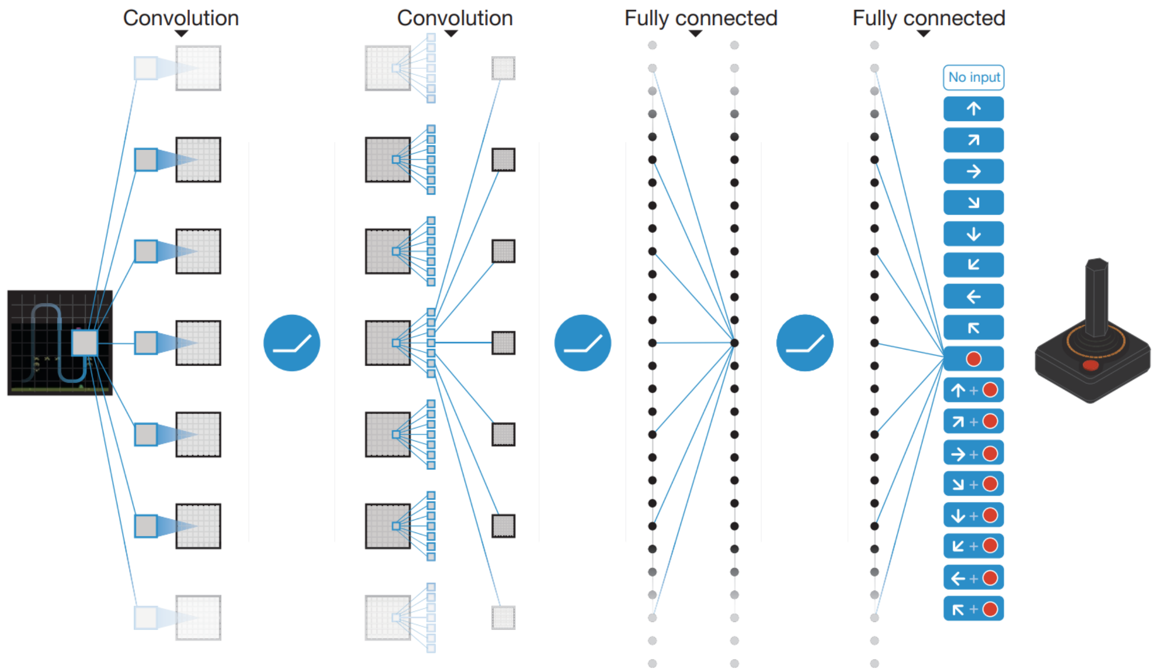


- Achieves >=75% of human professional players' scores on 29 games
- Can significantly outperform human players in many games

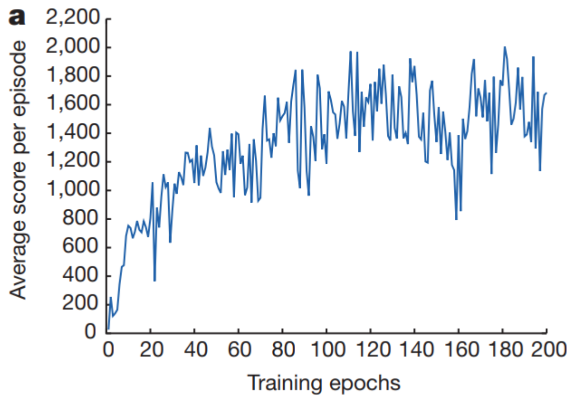
https://arxiv.org/pdf/1312.5602.pdf
https://www.nature.com/articles/nature14236

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

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

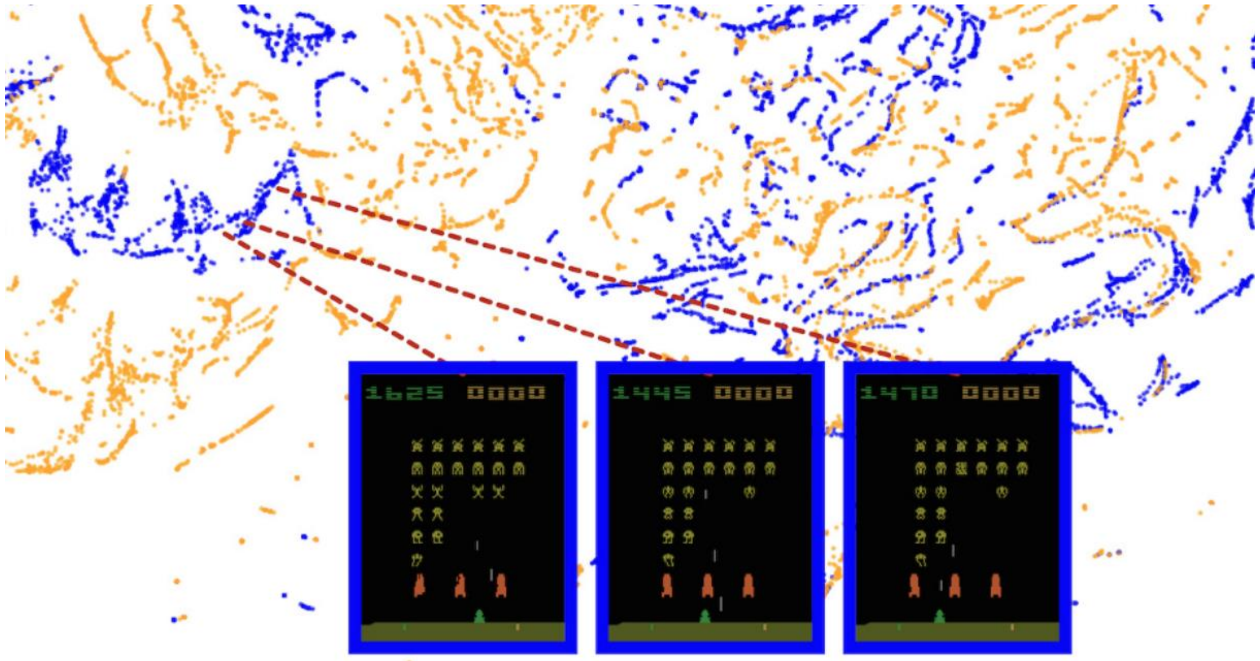


- Learning curve: (Space Invaders, ϵ-greedy with $\epsilon = 0.05$)



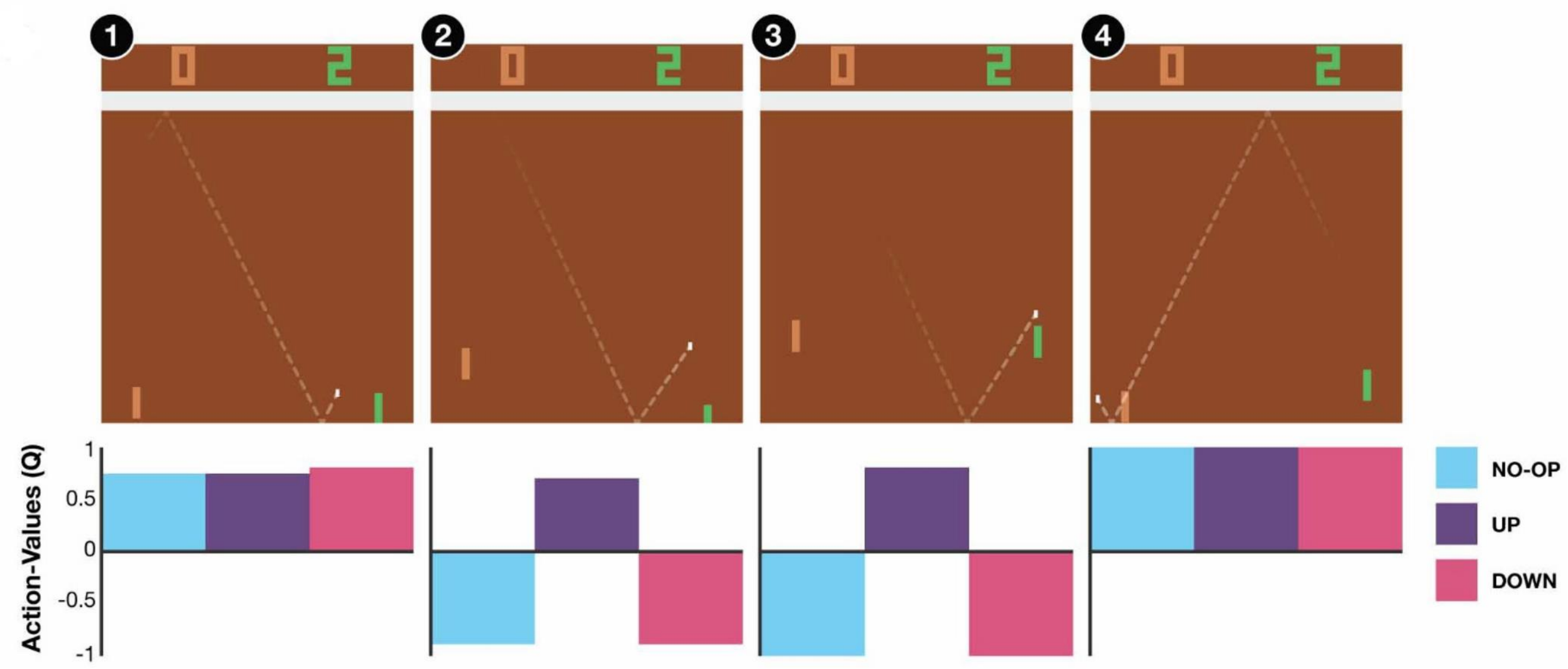
Fitted Q-learning example: Atari games (cont'd)

- 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

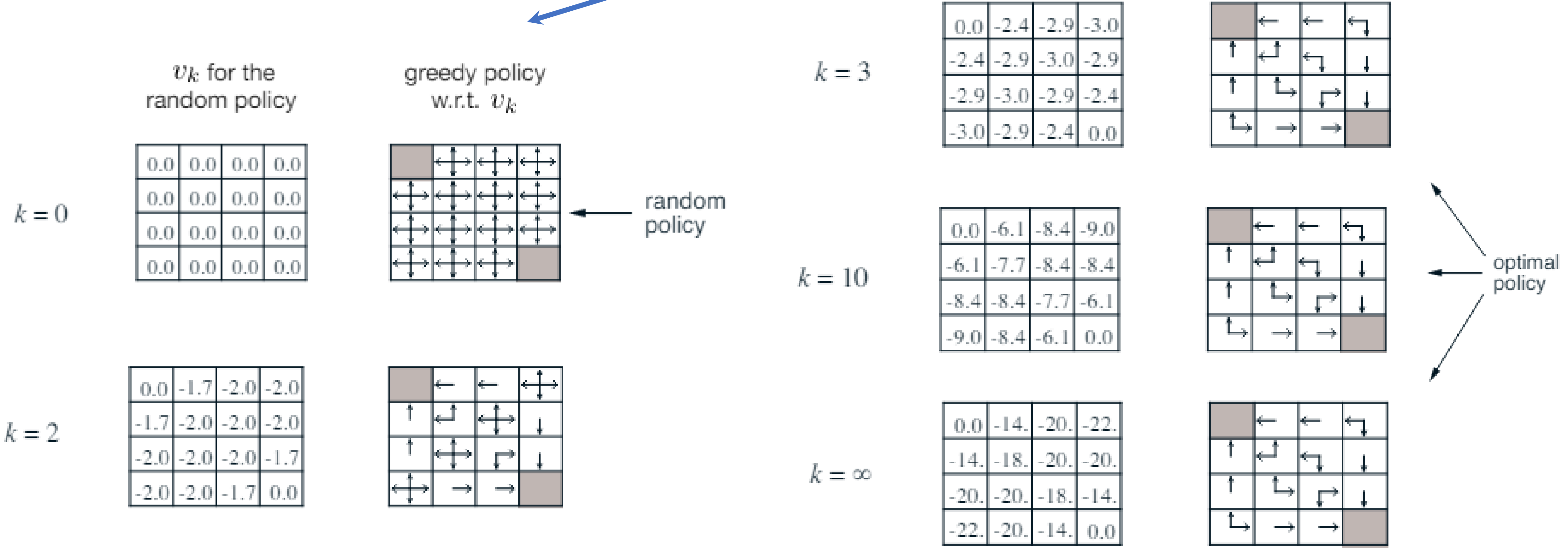


Policy iteration: an interesting observation

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

$$V^{k+1} \leftarrow R^{\pi} + \gamma M^{\pi} V^{k}$$

what you get if you apply the policy improvement step



Even though V^{k} may be far from V^{π}, the greedy policy of V^{k} is close to that of V^{π}

Algorithm: <u>Modified</u> policy iteration

- From previous slide: inexact value functions are still useful!
- Start from an arbitrary policy π (e.g., assign actions randomly)

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

- Repeat the following (until V converges):
  - **[(Inexact) Policy evaluation]** $V \leftarrow$ take k fixed-point iterations for computing V^{π} (so $V \approx V^{\pi}$)
  - **[Policy improvement]** Update the policy:

    For every s $\in S$, $\quad \pi(s) = \arg\max_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V(s')$

- Policy evaluation: just evaluates the value function for a given π
  - closed form / fixed-point iteration

- Planning:
  - Value iteration
  - Policy iteration: policy evaluation + policy improvement