DDA4220 · Fall 2026

Lecture 19

Deep Learning and Applications

Reinforcement learning: value learning and DQN

Zhen Liu

CUHK-Shenzhen

Lecture outline

01 Decisions and rewards

02 Returns and values

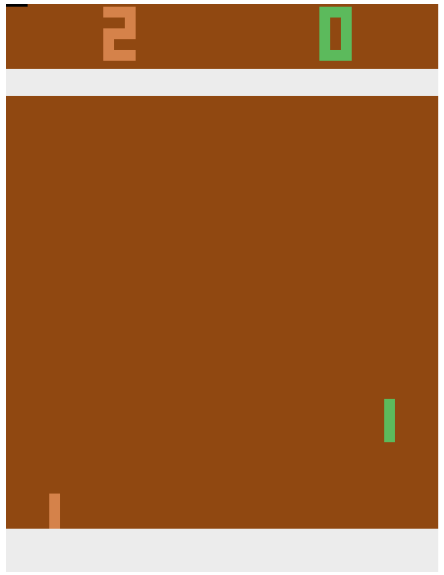
03 Learning from experience

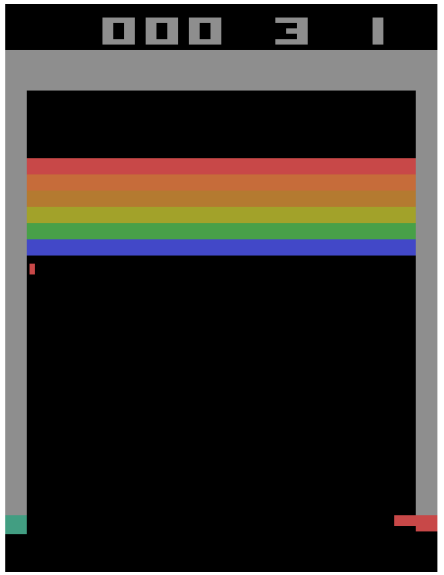
04 Deep Q-networks

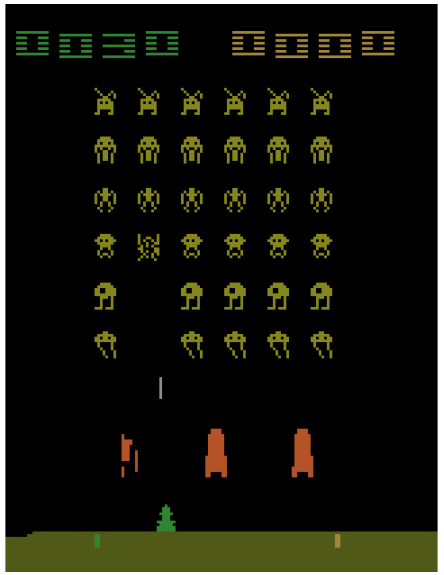
05 Capabilities and limits

* denotes optional slides

Learning to play from the screen

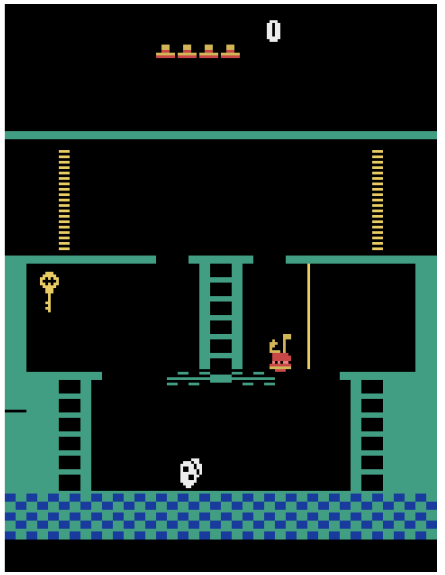

Pong


Breakout


Space Invaders



Seaquest


Montezuma's Revenge

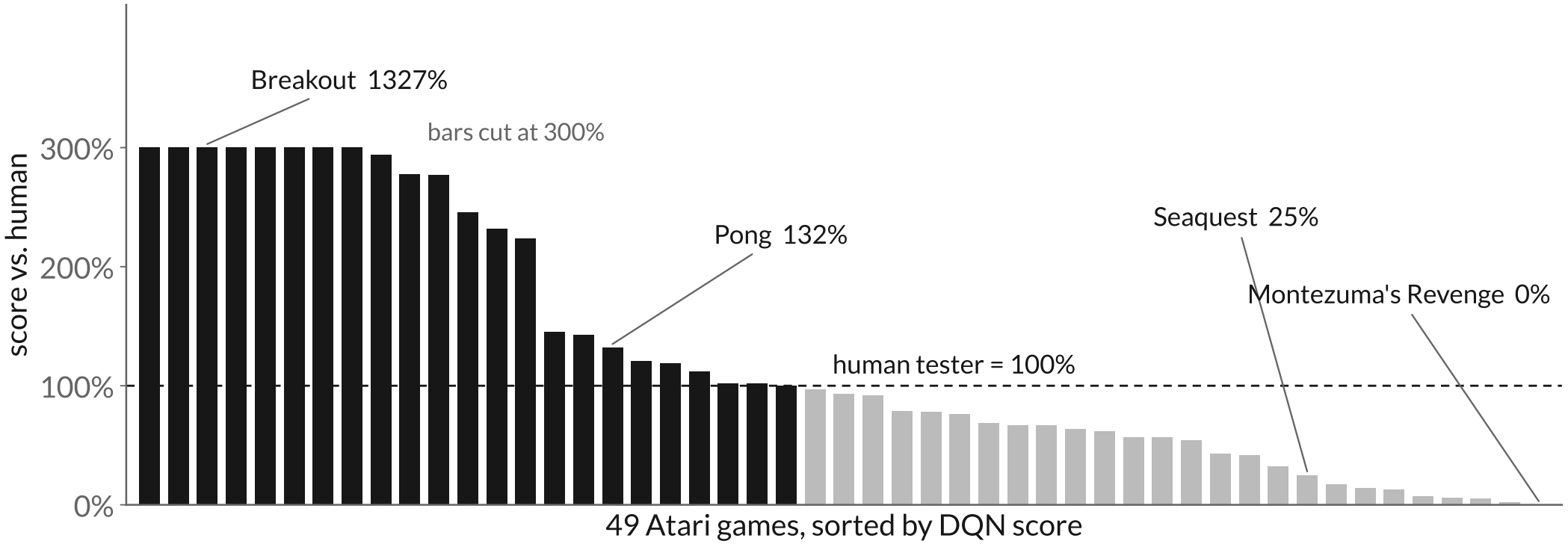
Input: the screen (210×160 pixels) and the score

Output: one of up to 18 joystick actions

No rules, no labelled moves; one setup for all 49 games

Source: Mnih et al., Nature 2015 · frames: Arcade Learning Environment

DQN against a human tester



0%: random play. 100%: a professional tester

At or above the tester in 23 of 49 games

Montezuma's Revenge: 0% (Section 05)

Source: Mnih et al., Nature 2015

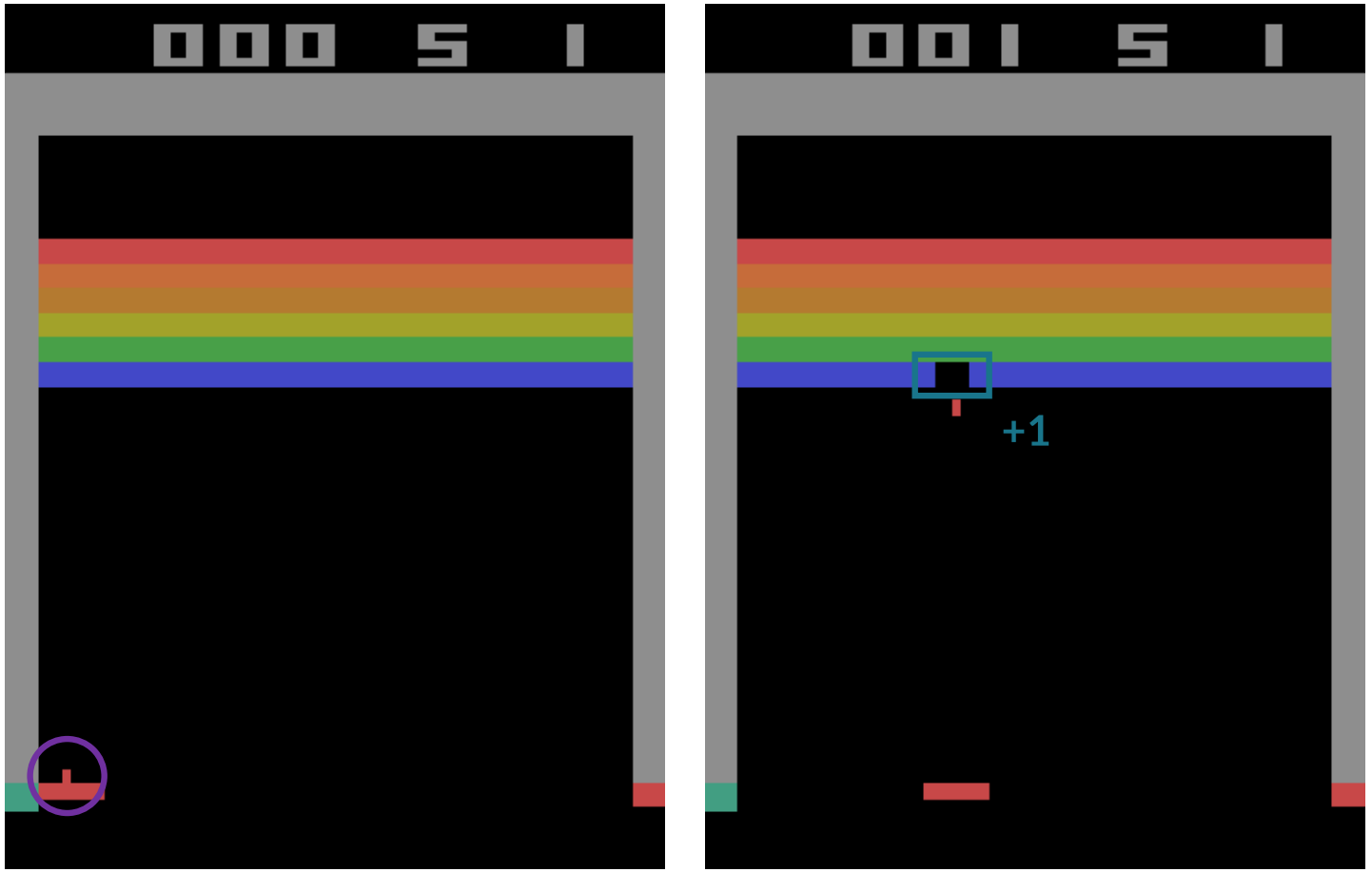
What supervised learning had

	Supervised (L1–L18)	Playing Breakout
Data	fixed (input, label) pairs	screens, actions, a score
Feedback	the correct output	a late number, never the right move
Who picks the data	collected beforehand	the agent's own actions

Learn which actions lead to reward, from the rewards alone

Reinforcement learning (RL)

Credit arrives late



Frame 63: the paddle hits the ball (ring)

Frame 126: a brick breaks, **+1**

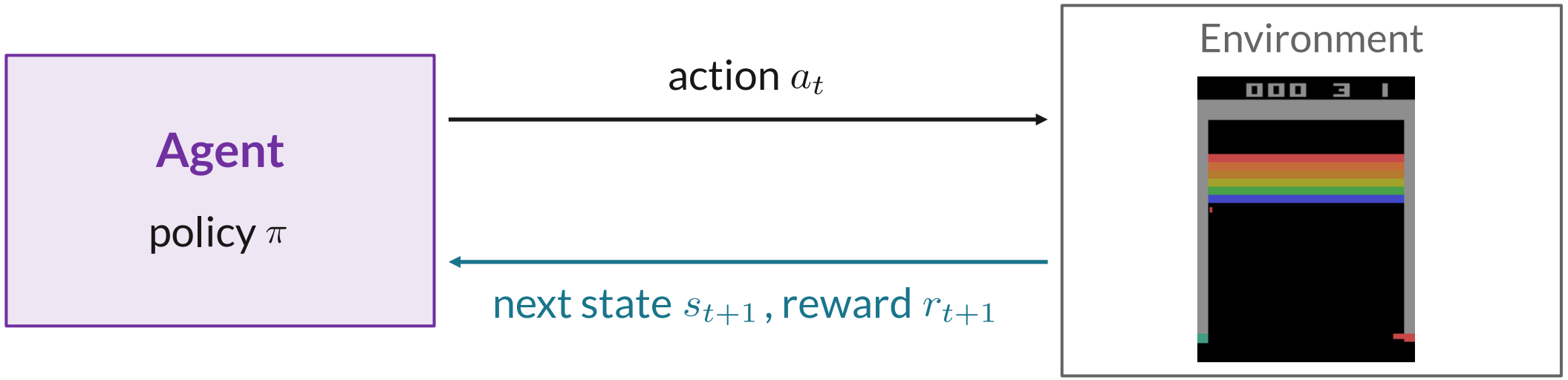
In between: 16 more decisions

Which decision earned the point?

The credit assignment problem

Source: frames: Arcade Learning Environment

The agent–environment loop



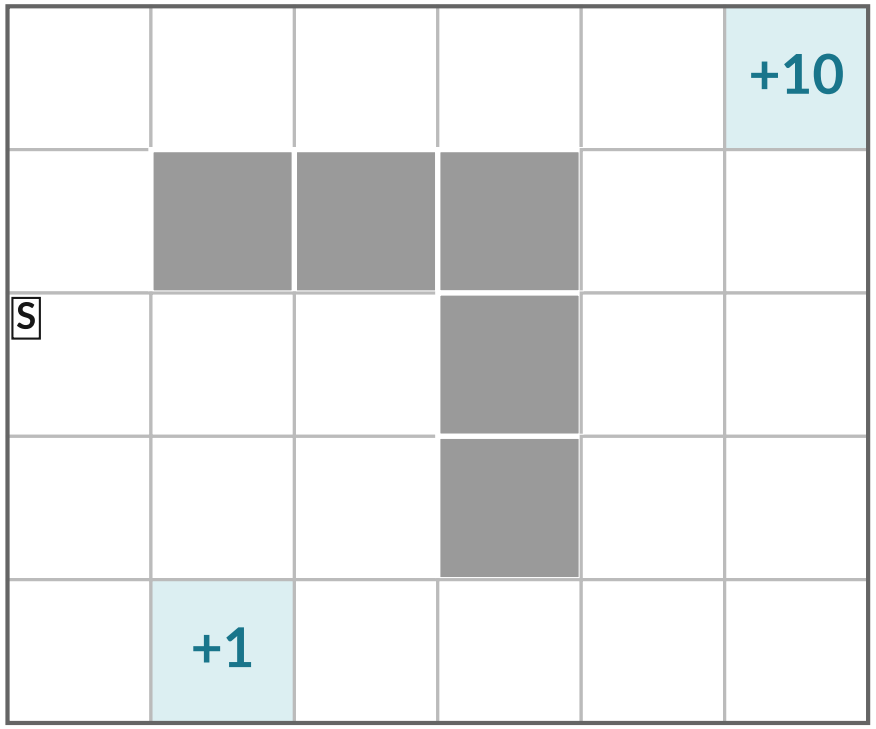
s_t: what the agent sees at step t; a_t: what it does

r_{t+1}: the reward it receives after acting

Goal: make the rewards add up to as much as possible

Source: frames: Arcade Learning Environment

A small world we can solve exactly



A robot starts at S; gray cells are walls

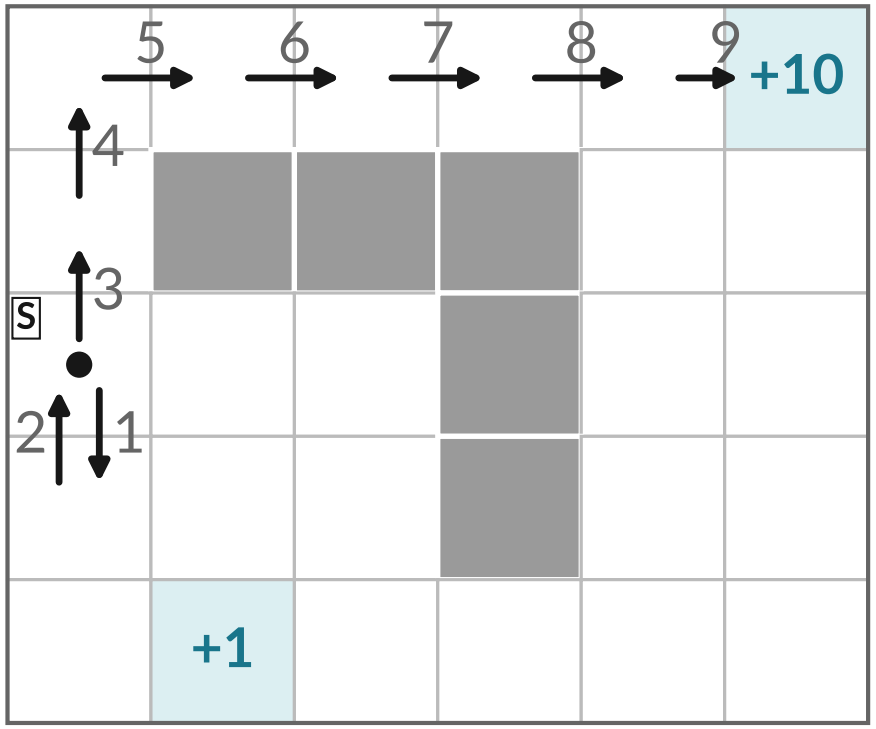
Moves: up, right, down, left; a wall means staying put

+1 parcel nearby, +10 delivery far away; either ends the episode

Every other move: reward 0

Small enough to solve exactly

One episode, step by step



An episode: states, actions, rewards

$$s_0,\ a_0,\ r_1,\ s_1,\ a_1,\ r_2,\ s_2,\ \ldots$$

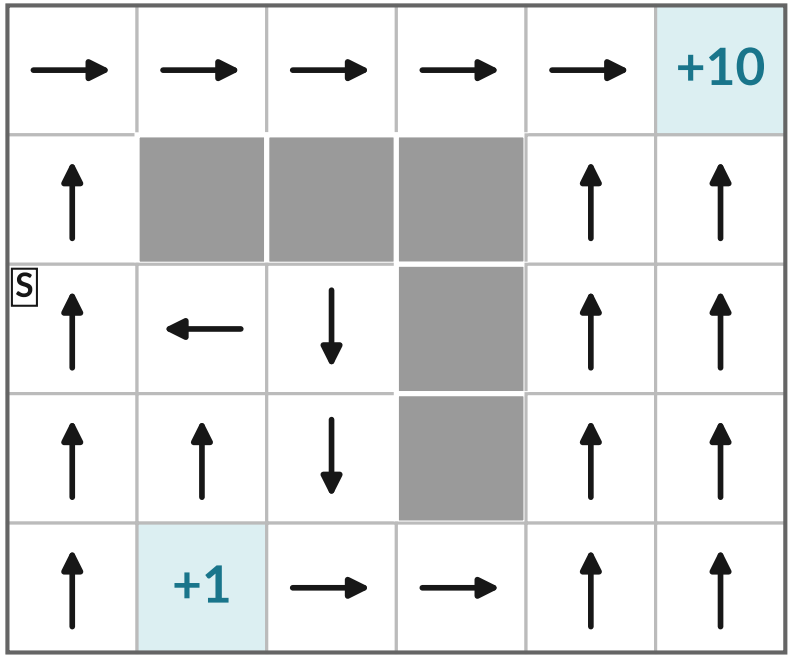
Here: 9 moves from S (numbered); moves 1–2 are a detour

Rewards: $0, 0, 0, 0, 0, 0, 0, 0,$ then **+10**

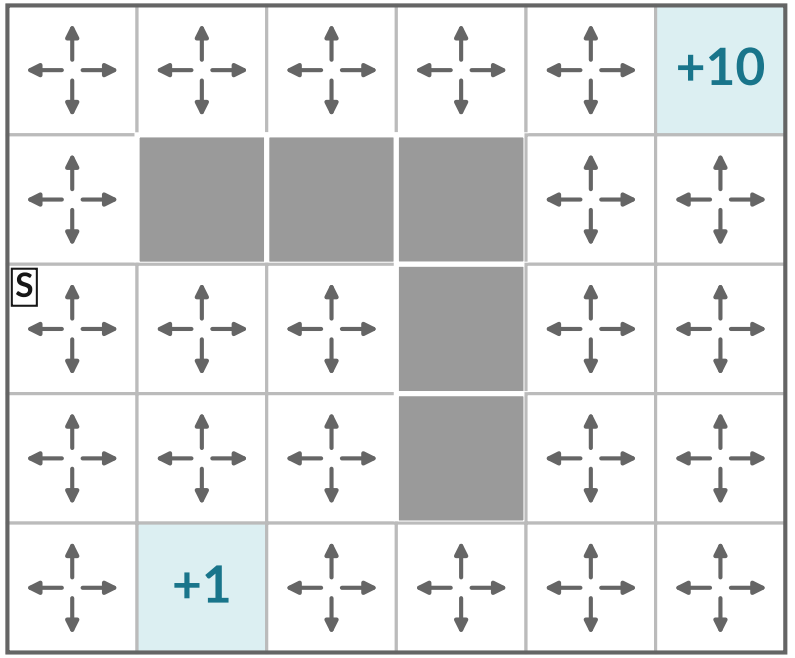
r_{t+1} is the reward for the move a_t

Only the last move is rewarded; the earlier ones made it possible

A policy chooses the actions



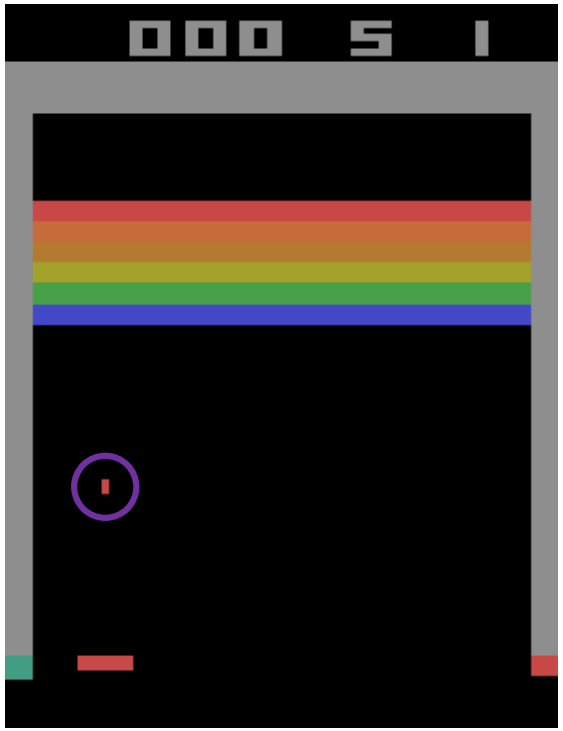
deterministic: one action per cell



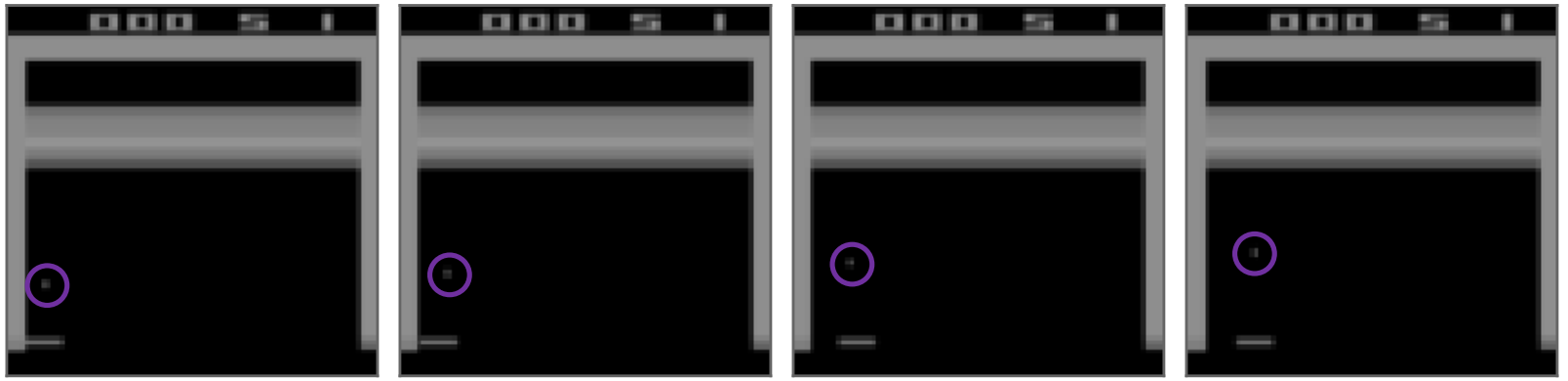
uniform random: each action with probability 1/4

$\pi(a \mid s)$: the probability that the policy picks action a in state s

The state must carry what matters



one frame



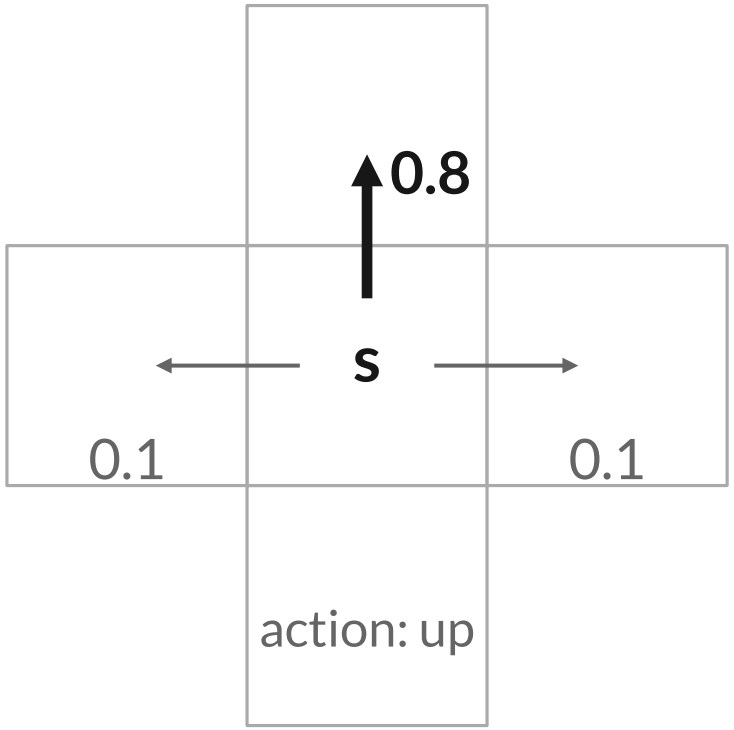
DQN's input: the last 4 frames, gray, 84 × 84

One frame: where the ball is, not where it is going

State: enough information to predict what happens next

Source: frames: Arcade Learning Environment · Mnih et al., Nature 2015

Moves can slip



Slippery grid: the chosen move with probability 0.8; sideways, 0.1 each

$P(s' \mid s, a)$: probability of landing in s' after action a in state s

s' (read "s prime"): the next state

Same action, different outcomes

Markov decision process

States	$\mathcal{S}$	the 25 open cells
Actions	$\mathcal{A}$	up, right, down, left
Transitions	$P(s' \mid s, a)$	where an action leads, and how likely
Rewards	r	+1 and +10 on arrival, else 0
Discount	γ	weight of later rewards (Section 02)

Markov property: only the current state and action matter for what comes next

$$P(s_{t+1} \mid s_t, a_t,\ s_{t-1}, a_{t-1}, \ldots, s_0) \;=\; P(s_{t+1} \mid s_t, a_t)$$

The data depend on how we act

2.5	1.7	1.3	0.8	0.4	+10
3.3				0.4	0.3
S 4.2	2.2	1.9		0.4	0.4
2.9	1.6	1.4		0.4	0.3
1.3	+1	0.7	0.6	0.5	0.3

uniform random policy

1.1	1.1	1.1	1.1	1.1	+10
1.2				0.1	0.0
S 1.2	0.1	0.0		0.0	0.0
0.1	0.0	0.0		0.0	0.0
0.0	+1	0.0	0.0	0.0	0.0

best route, with 10% random moves

Visits per episode to each cell

No fixed dataset: the policy decides which states we see

Lecture outline

01 Decisions and rewards

02 Returns and values

03 Learning from experience

04 Deep Q-networks

05 Capabilities and limits

Scoring a whole episode

First try: add up all rewards from step t on

$$G_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots$$

Route from S	sum of rewards
+1 parcel, 3 moves	**1**
+10 delivery, 7 moves	**10**
+10 delivery after 100 aimless moves	**10**

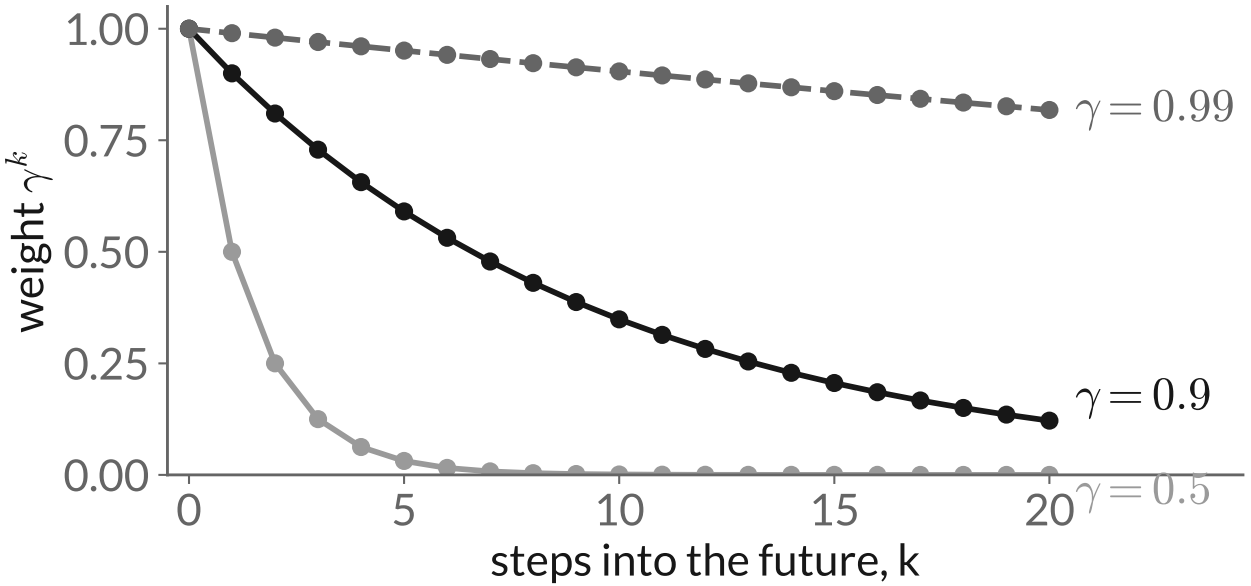
A quick and a slow delivery score the same

A task without an end (balancing a pole): infinite sum

Wanted: sooner rewards count more; the total stays finite

Discounting: later rewards count less

$$G_t \;=\; r_{t+1} + \gamma\, r_{t+2} + \gamma^2 r_{t+3} + \cdots \;=\; \sum_{k=0}^{\infty} \gamma^k\, r_{t+k+1}$$



γ (gamma): the discount, between 0 and 1

A reward k steps away is multiplied by γ^k

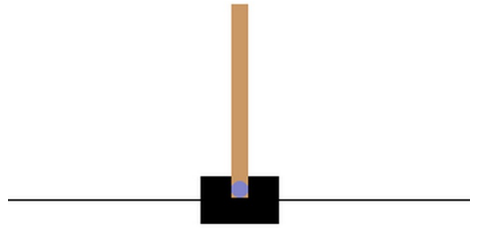
After 10 steps: weight 0.001 for $\gamma = 0.5$, 0.35 for 0.9, 0.90 for 0.99

G_t: the return, the quantity the agent tries to make large

Discounting keeps returns finite

Every reward at most $r_{\max}$: a geometric series

$$G_t \;\le\; r_{\max}\,(1+\gamma+\gamma^2+\cdots) \;=\; \frac{r_{\max}}{1-\gamma}$$



$\gamma = 0.9$: at most $10\,r_{\max}$; $\gamma = 0.99$: at most $100\,r_{\max}$

CartPole (right): push the cart left or right to keep the pole up

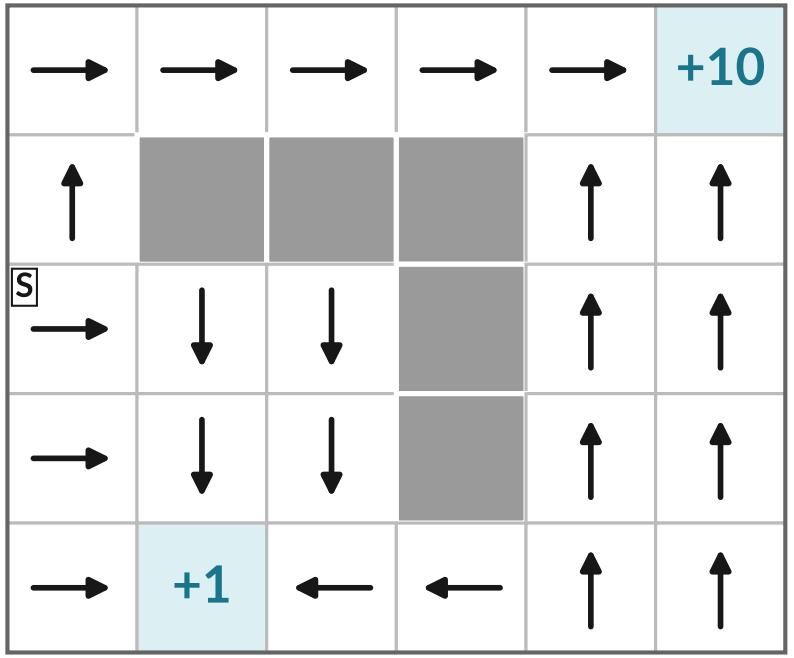
Reward 1 per step while the pole is up; at most 500 steps

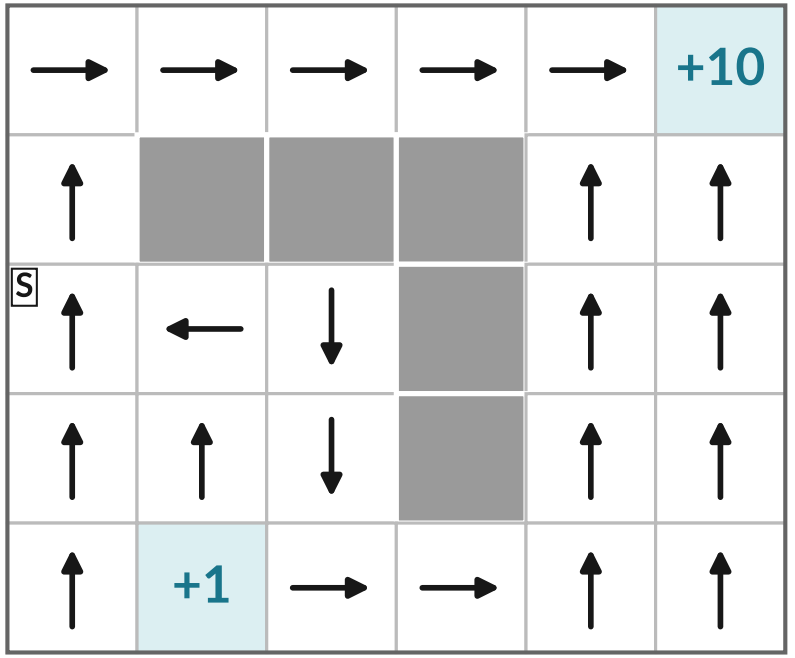
$$G_0 \;\le\; 1+0.99+\cdots+0.99^{499} \;=\; \frac{1-0.99^{500}}{1-0.99} \;\approx\; 99.3$$

99.3 returns in Section 04 as a check on learned values

Source: CartPole-v1, Gymnasium

The discount changes the best route





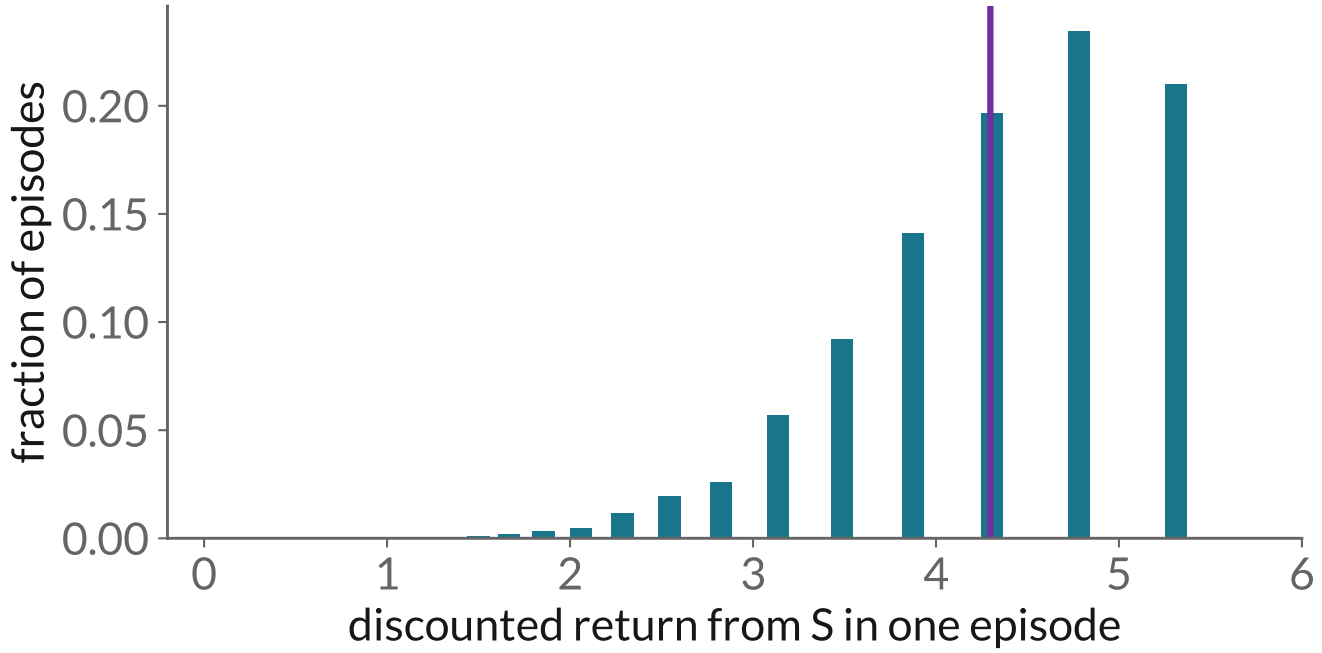
$\gamma = 0.5$: go for the $+1$

$\gamma = 0.9$: go for the $+10$

From S: $+1$ (3 moves) is worth $\gamma^2 \cdot 1$; $+10$ (7 moves) is worth $\gamma^6 \cdot 10$

$\gamma = 0.5$: 0.25 against 0.16. $\gamma = 0.9$: 0.81 against 5.31

Slips make the return random



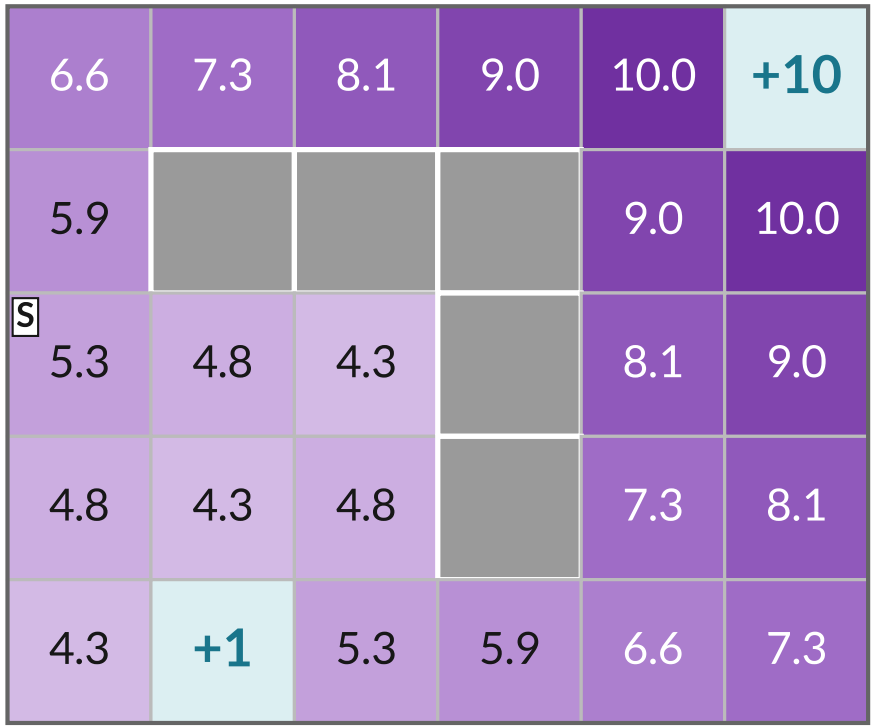
Slippery grid, one fixed policy, 5000 episodes from S

Returns from 0 to 5.3: slips add detours

Purple line: their average, 4.30

A single return is noisy; we want its average

State value: the average return



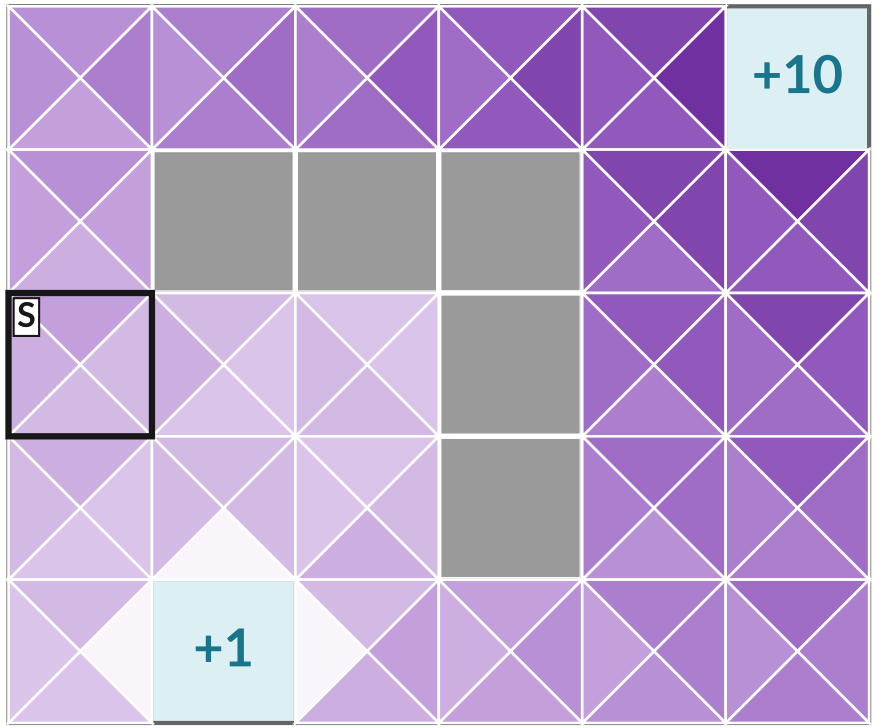
$$V^{\pi}(s) \;=\; \mathbb{E}_{\pi}[\, G_t \mid s_t = s \,]$$

$\mathbb{E}_{\pi}$: average over the policy's choices and the slips

Map: best policy, $\gamma = 0.9$, no slips

V **: how good a state is, not what to do there**

Action value: first a, then follow π



$$Q^{\pi}(s,a) \;=\; \mathbb{E}_{\pi}[\, G_t \mid s_t = s,\; a_t = a \,]$$

Take a now, then follow π

Four triangles per cell: one value per action

At S (outlined)

up	**5.31**
left (wall: stay)	4.78
right	4.30
down	4.30

Comparing Q values says what to do

The return is recursive

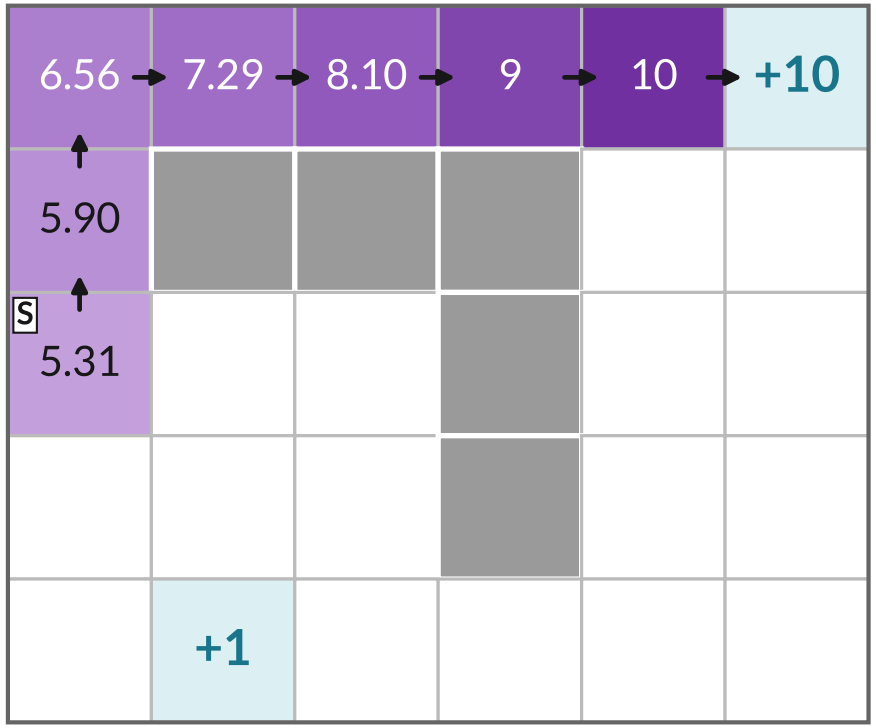
$$
\begin{aligned}
G_t &= r_{t+1} + \gamma\, r_{t+2} + \gamma^2 r_{t+3} + \cdots \\
&= r_{t+1} + \gamma \left(r_{t+2} + \gamma\, r_{t+3} + \cdots \right) \\
&= r_{t+1} + \gamma\, G_{t+1}
\end{aligned}
$$

Return now = next reward + $\gamma \times$ return from the next step

Map: returns along the delivery route, $\gamma = 0.9$; no reward until the end, so each is $0.9 \times$ the next

The same numbers as the value map (slide 21)

Next: average this to get an equation for V



Bellman equation for a policy

Average $G_t = r_{t+1} + \gamma\, G_{t+1}$ over everything that can happen from s:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi\big[\, r_{t+1} + \gamma\, G_{t+1} \mid s_t = s \,\big] \\ &= \mathbb{E}_\pi\big[\, r_{t+1} + \gamma\, V^\pi(s_{t+1}) \mid s_t = s \,\big] \end{aligned}$$

from s_{t+1} on, the average return is $V^\pi(s_{t+1})$

Written out, with the averages over actions and slips:

$$V^\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} P(s' \mid s, a) \Big[\, r + \gamma\, V^\pi(s') \,\Big]$$

average over actions

average over outcomes

reward

next state's value

One equation per state: 25 equations, 25 unknowns

A state's value is tied to its neighbours' values

Checking the equation at S



Top: probability of landing there
Bottom: exact value

Slippery grid, $\gamma = 0.9$; at S the best policy chooses up

No reward on these moves

$$V(S) = 0.8 \cdot 0.9 \cdot 4.965 + 0.1 \cdot 0.9 \cdot 3.734 + 0.1 \cdot 0.9 \cdot 4.297 = 3.575 + 0.336 + 0.387 = 4.297$$

Exact values satisfy the equation at every cell

Acting from Q needs no model

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

$\max_a$: the largest value; $\arg\max_a$: the action that attains it

At S: $\max = 5.31$, $\arg\max =$ up

With only V , choosing needs a look-ahead through P (the bracket of the Bellman equation):

$$\pi(s) \;=\; \arg\max_a \sum_{s'} P(s' \mid s,a)\,\big[r + \gamma V(s')\big]$$

Model: the rules P and r of the environment

Without a model: learn Q and read off the action

The best values: Bellman optimality

$$Q^*(s,a) = \sum_{s'} P(s' \mid s,a) \left[r + \gamma \max_{a'} Q^*(s',a') \right]$$

average over outcomes

reward

then act as well as possible

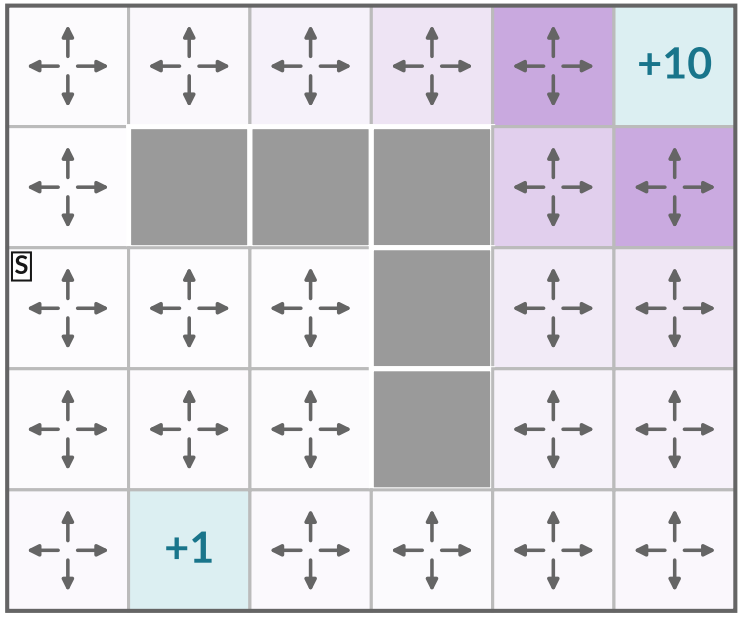
Q^* : action values of the best policy; $V^*(s) = \max_a Q^*(s,a)$

The max over a' replaces the policy's average: from s' on, act as well as possible

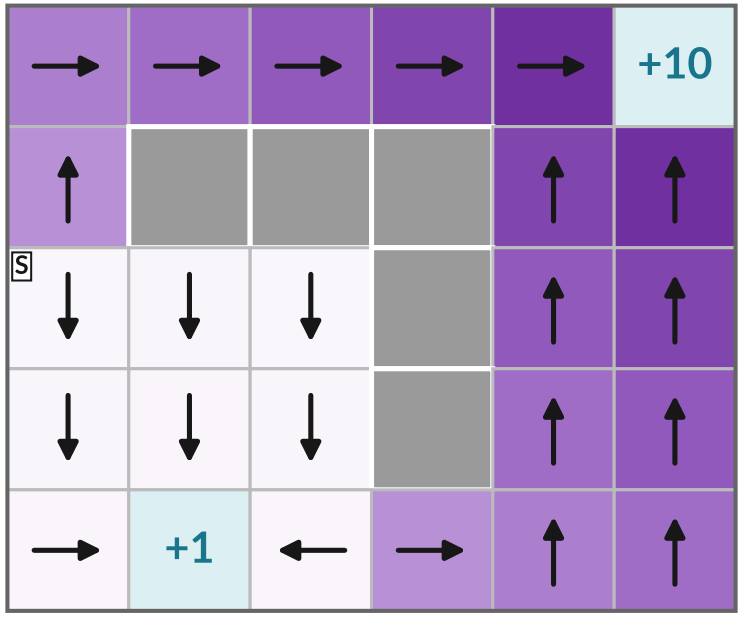
Best policy: $\arg\max_a Q^*(s,a)$, no look-ahead needed

Value iteration, Q-learning and DQN all try to solve this equation

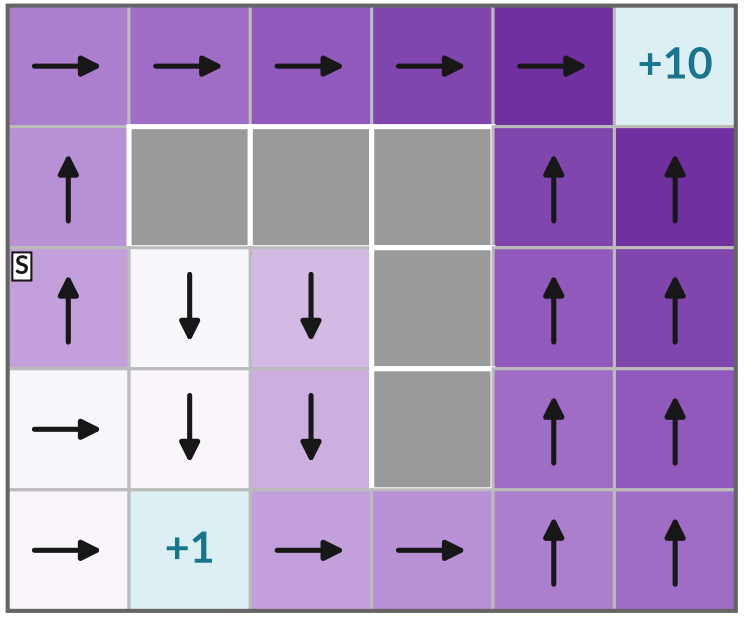
Improving a policy by acting greedily



uniform random: V(S) = 0.30



greedy w.r.t. its Q: V(S) = 0.81



greedy again: V(S) = 5.31

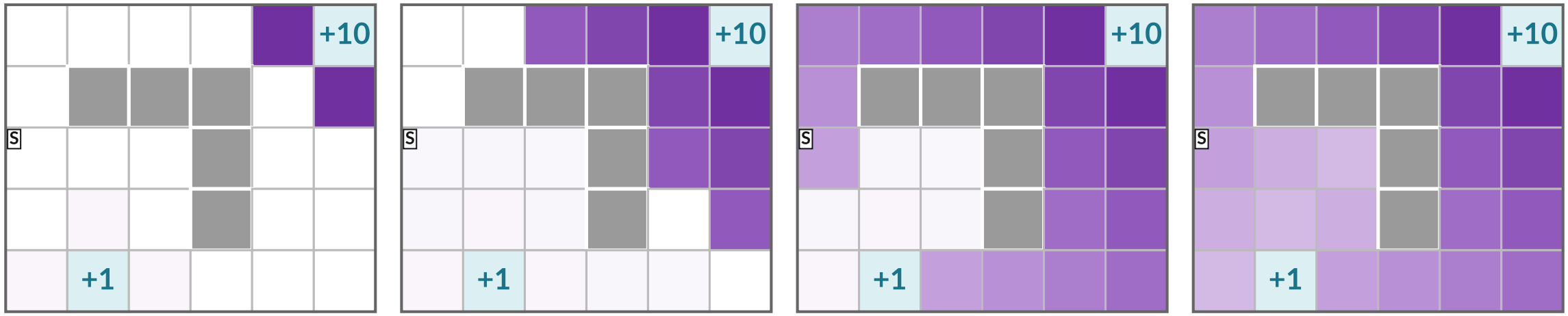
Policy iteration: compute Q^π, switch to $\arg\max_a Q^\pi(s,a)$ in every state, repeat

Never worse: the best action is worth at least the average, $\max_a Q^\pi(s,a) \geq V^\pi(s)$

Here optimal after two rounds. Shade: value; arrows: actions

Value iteration: repeat the Bellman update

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s' \mid s, a) \left[r + \gamma V_k(s') \right]$$



1 sweep: V(S) = 0 | 3 sweeps: V(S) = 0.81 | 7 sweeps: V(S) = 5.31 | 9 sweeps: exact

Start from $V_0 = 0$; one sweep updates every state. Shade: $V_k(s)$, white 0 to purple 10

Reward travels one cell per sweep: the nearby $+1$ reaches S first

Value iteration needs the model

1. The rules: $P(s' \mid s, a)$ and r for every state and action

Not given in Breakout; not known exactly for a robot

2. Every state, every sweep: about $|\mathcal{S}|^2|\mathcal{A}|$ operations

Grid: $25^2 \cdot 4 = 2500$. Go: about 10^{170} positions

What we can always do: act, and record what happens

Section 03: learn V and Q from these records

Lecture outline

01 Decisions and rewards

02 Returns and values

03 Learning from experience

04 Deep Q-networks

05 Capabilities and limits

Experience: one transition at a time

Each step leaves a record, a transition: $(s,\ a,\ r,\ s')$

state s	action a	reward r	next state s'
S	down	0	cell below S
cell below S	up	0	S
S	up	0	cell above S
…			
next to delivery	right	+10	delivery (episode ends)

The 9-move episode from Section 01 as 9 transitions

Model-free: learn values from these records alone, never from P

Monte Carlo: average the returns you saw

$$V(s) \leftarrow V(s) + \underbrace{\alpha}_{\text{step size}} \Big[\underbrace{G_t}_{\text{observed return}} - \underbrace{V(s)}_{\text{current estimate}} \Big]$$

Move a fraction α of the way toward the observed return

$\alpha = 1/n$ on the n-th visit: exactly the running average

9-move episode: return at S is $0.9^8 \cdot 10 = 4.30$

Many episodes average out the randomness

Monte Carlo waits for the end

G_t is known only when the episode is over

Grid: about 10 steps. CartPole: up to 500. Atari: thousands

A task that never ends never produces a return

Each return adds up every later slip and random choice: noisy

Return spread on the slippery grid: slide 20

Wanted: an update after every single step

Temporal-difference learning

$$V(s) \leftarrow V(s) + \alpha \left[\underbrace{r}_{\text{observed reward}} + \underbrace{\gamma\, V(s')}_{\text{estimate for the next state}} - \underbrace{V(s)}_{\text{current estimate}} \right]$$

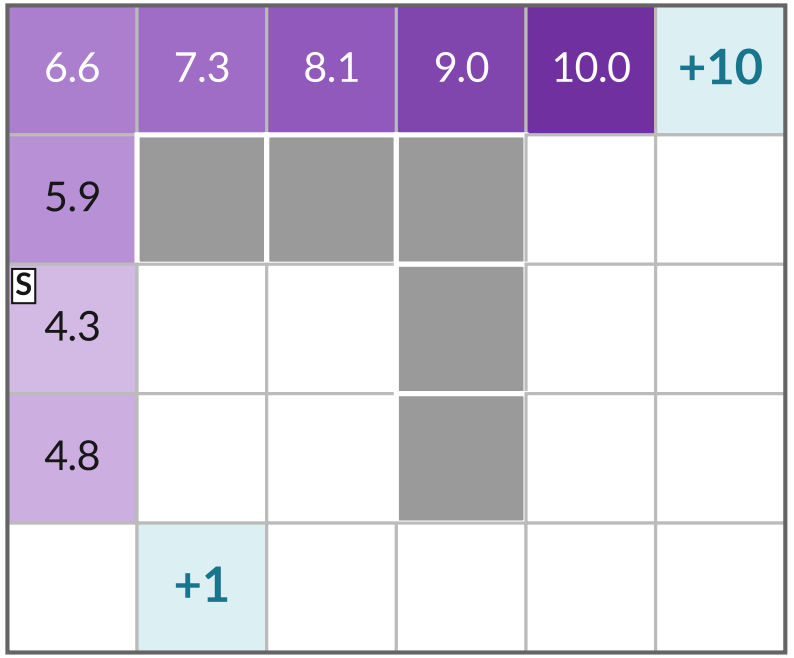
In $G_t = r_{t+1} + \gamma G_{t+1}$, replace G_{t+1} by an estimate

TD target: $r + \gamma V(s')$; TD error: target minus $V(s)$

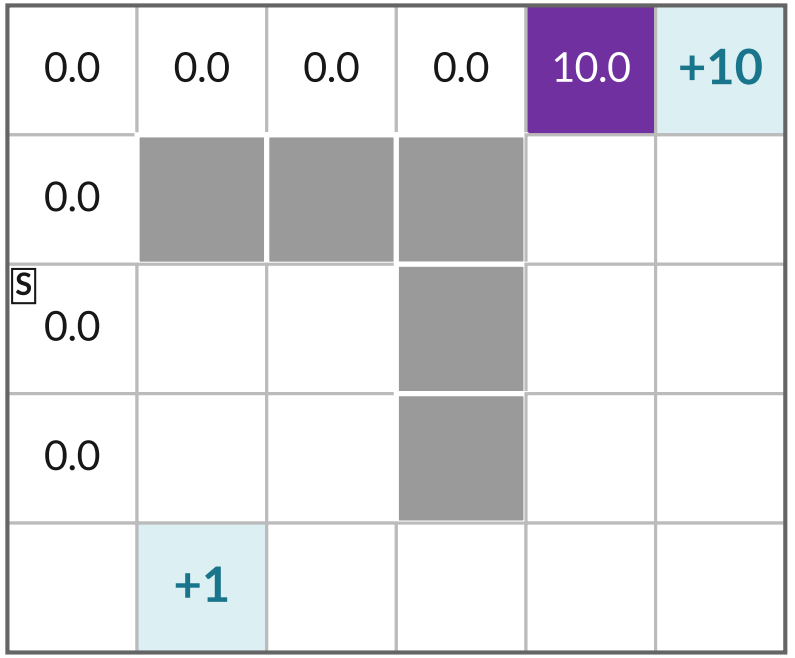
Update after each transition, no waiting

Bootstrapping: the target uses our own estimate

One episode: Monte Carlo against TD



Monte Carlo: every visited cell gets its return

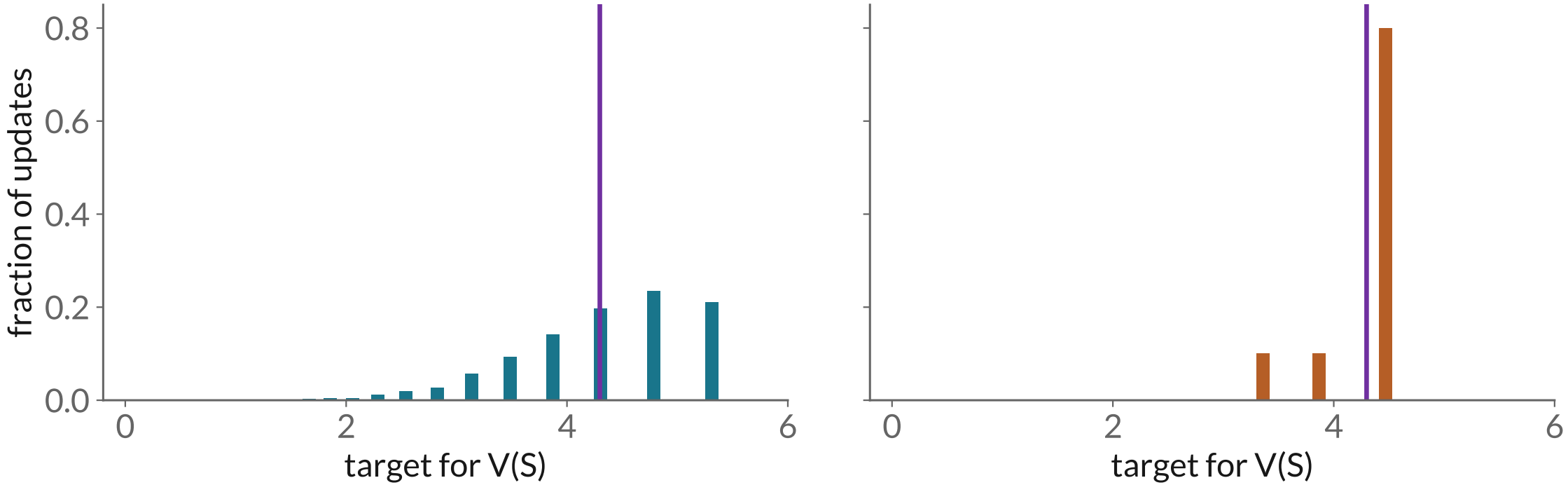


TD: only the cell next to the goal changes

Same 9-move episode; values start at 0; $\alpha = 1$

TD passes reward back one cell per visit, but never waits for an ending

What each target trusts



Monte Carlo: the whole return G_t

TD: $r + \gamma\, V(s')$, with the exact V

Targets for $V(S)$ on the slippery grid; both average 4.30 (purple); standard deviation 0.82 against 0.36

TD's target is only as good as $V(s')$: at the start $V = 0$, so it says 0

Monte Carlo: right on average but noisy. TD: steadier, but it trusts an estimate

To choose actions, learn Q

TD learns V; acting on V needs the dynamics P

(the look-ahead on slide 26)

Learn $Q(s, a)$ instead: compare and pick

Target for $Q(s, a)$: after this step, the best from s' is worth

$$\max_{a'} \; Q(s', a')$$

Use it in the TD target: Q-learning

Q-learning

$$Q(s,a) \leftarrow Q(s,a) + \alpha \Big[\underbrace{r}_{\text{observed}} + \underbrace{\gamma \max_{a'} Q(s',a')}_{\text{estimate}} - \underbrace{Q(s,a)}_{\text{prediction being updated}} \Big]$$

The Bellman optimality equation, sampled

One transition (s, a, r, s') at a time; no P, no waiting

Every (s, a) tried often, α decreasing: Q converges to Q^*

Watkins and Dayan (1992); proof outside this course

Source: Sutton & Barto 2018

Two Q-learning updates by hand

$\alpha = 0.5$, $\gamma = 0.9$, all Q values start at 0

Episode 1, last step: next to the delivery, right, **+10**, episode ends

$$Q(\text{next to delivery}, \text{right}) \leftarrow 0 + 0.5\left[10 + 0 - 0\right] = 5$$

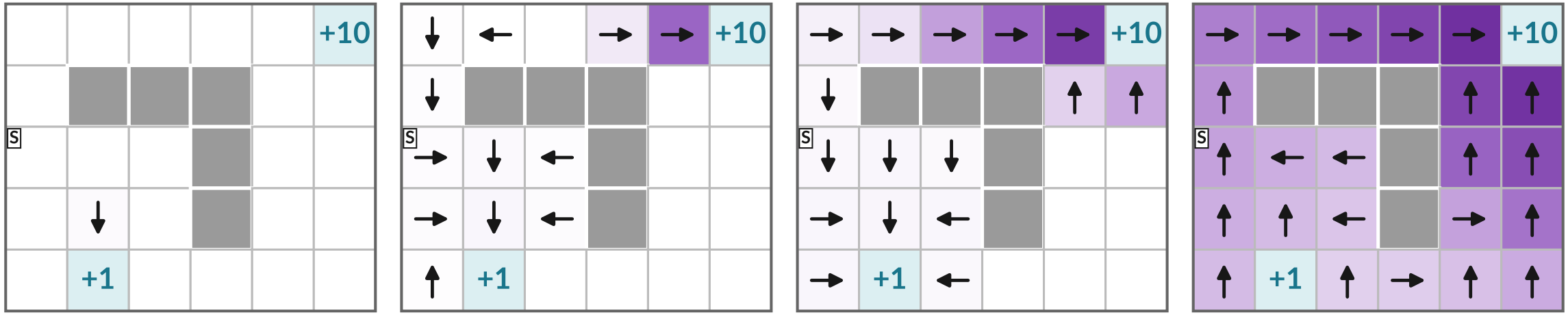
Episode 2: two cells before the delivery, right, reward 0

$$Q(\text{two before}, \text{right}) \leftarrow 0 + 0.5\left[0 + 0.9 \cdot 5 - 0\right] = 2.25$$

First target: no future term, the episode ended (slide 45)

The $+10$ moves back one cell each time the cell before it is updated

Q-learning on the grid



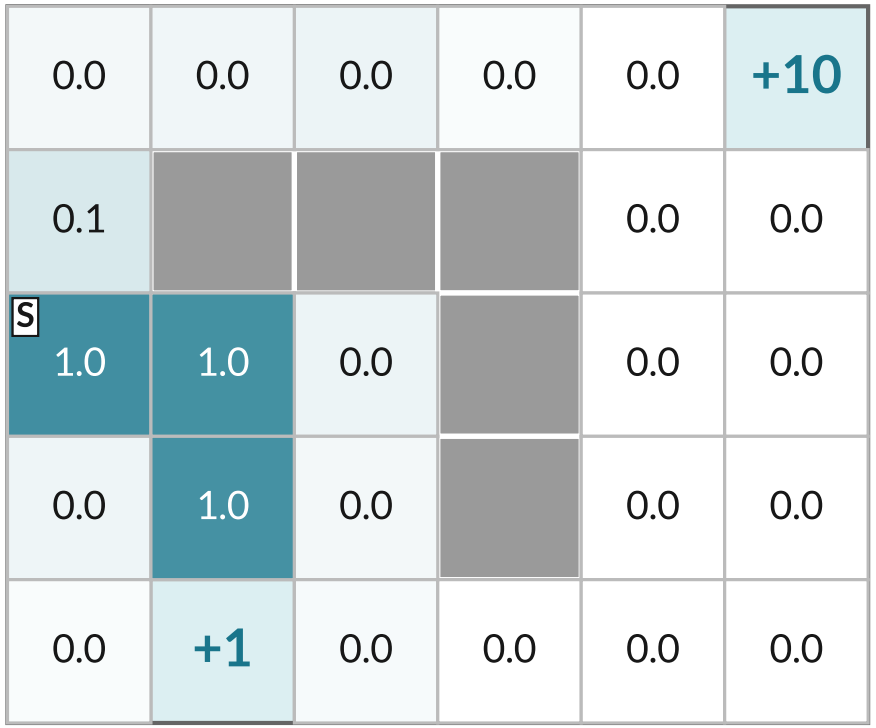
1 episode: 0 of 23 | 5 episodes: 3 of 23 | 20 episodes: 8 of 23 | 100 episodes: 19 of 23

Random moves; $\alpha = 0.5$, $\gamma = 0.9$; arrows where some $Q > 0$

Count: cells whose greedy action is exactly right, of the 23 open cells that are not goals

Reward spreads backward from the goals

Always greedy gets stuck



Always take the current best action; Q starts at 0, ties broken at random

Map: visits per episode, 300 episodes

First reward found: the $+1$; greedy then repeats that route

The $+10$ is never visited, never learned

The agent must sometimes try worse-looking actions

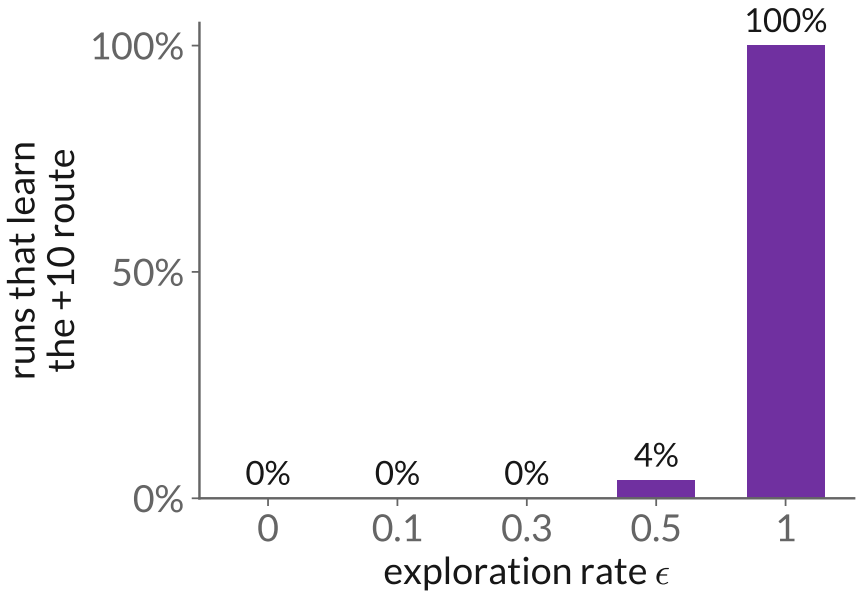
Epsilon-greedy: explore some of the time

$$a = \begin{cases} \text{a random action} & \text{with probability } \epsilon \\ \arg\max_a Q(s,a) & \text{otherwise} \end{cases}$$

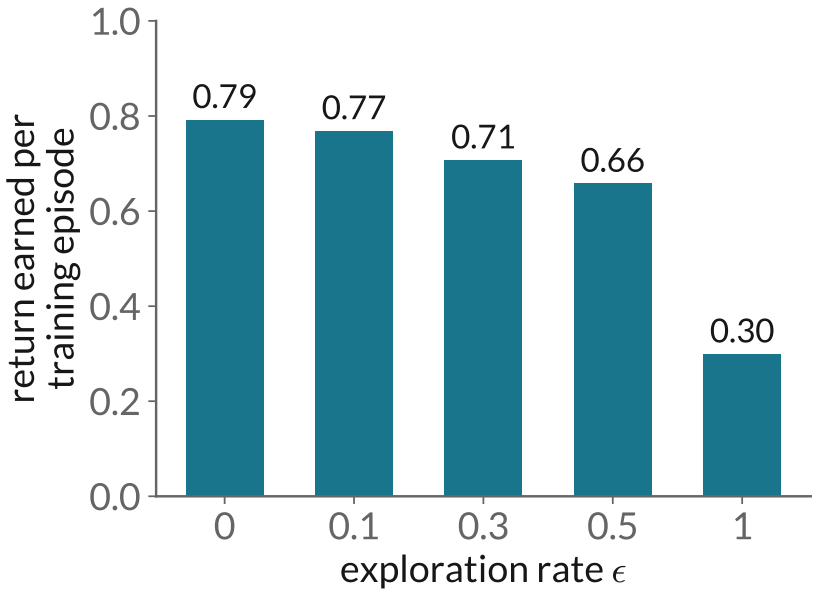
Constant ϵ; 300 episodes, 100 runs each

More exploration finds the $+10$

but earns less while learning



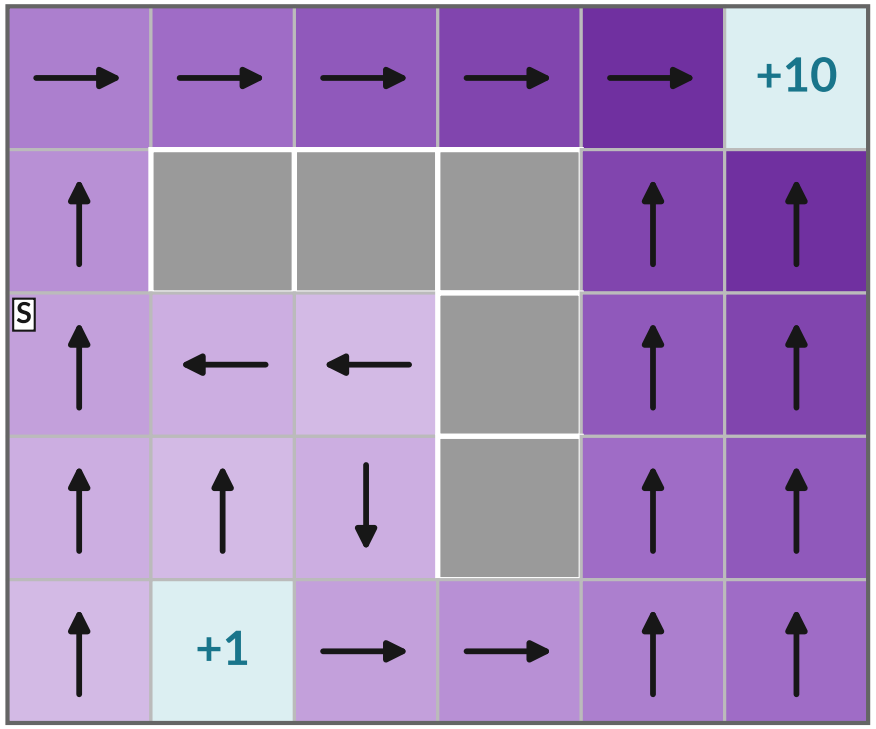
runs whose final greedy route takes the +10



return earned while training

Exploration–exploitation trade-off; DQN starts at $\epsilon = 1$ and lowers it

Learning the best policy while acting randomly



Random moves for 500 episodes

Earned only 0.30 per episode

Arrows: greedy actions of the learned Q, all 23 correct

$\max_a Q(S, a) = 5.3144$, the exact $V^*(S)$

Off-policy: the max evaluates the greedy policy, not the behaviour

When the episode ends

After the last move there is nothing to bootstrap from:

$$y = \underbrace{r}_{\text{observed}} + \underbrace{\gamma\,(1-d)\,\max_{a'} Q(s', a')}_{\text{dropped when the episode has ended}}$$

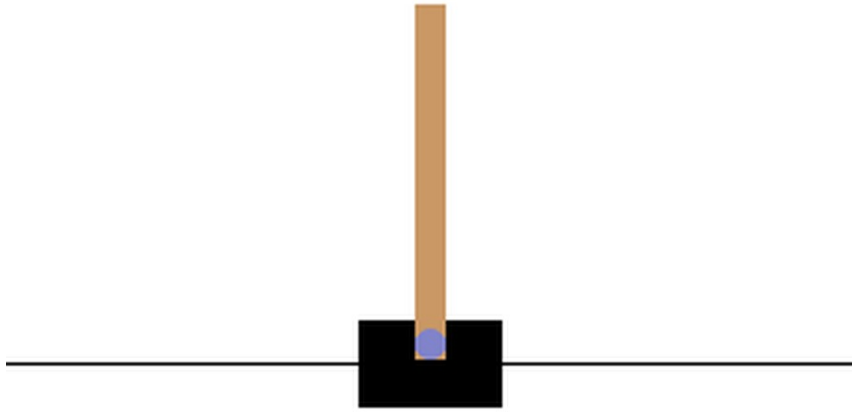
y: the target; $d = 1$ if the move ended the episode, else 0

Grid: reaching the delivery gives $y = 10$

CartPole: the pole falls, $d = 1$, $y = 1$

Without the mask, value leaks in from a state that never comes

Time limits are not endings



CartPole stops at 500 steps, even with the pole up

The state still has a future

Terminated (pole fell): $d = 1$

Truncated (time limit): $d = 0$, keep bootstrapping

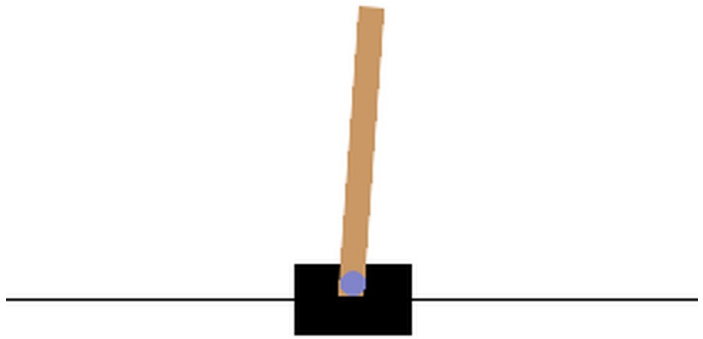
Gymnasium reports the two separately

Source: CartPole-v1, Gymnasium

What the table assumes

Grid: $23 \times 4 = 92$ entries, each tried again and again

One entry teaches nothing about its neighbours



CartPole: 4 real numbers per state

No state repeats exactly: a table never sees a row twice

Atari: $84 \times 84 \times 4$ gray pixels per state

Source: CartPole-v1, Gymnasium

Lecture outline

01 Decisions and rewards

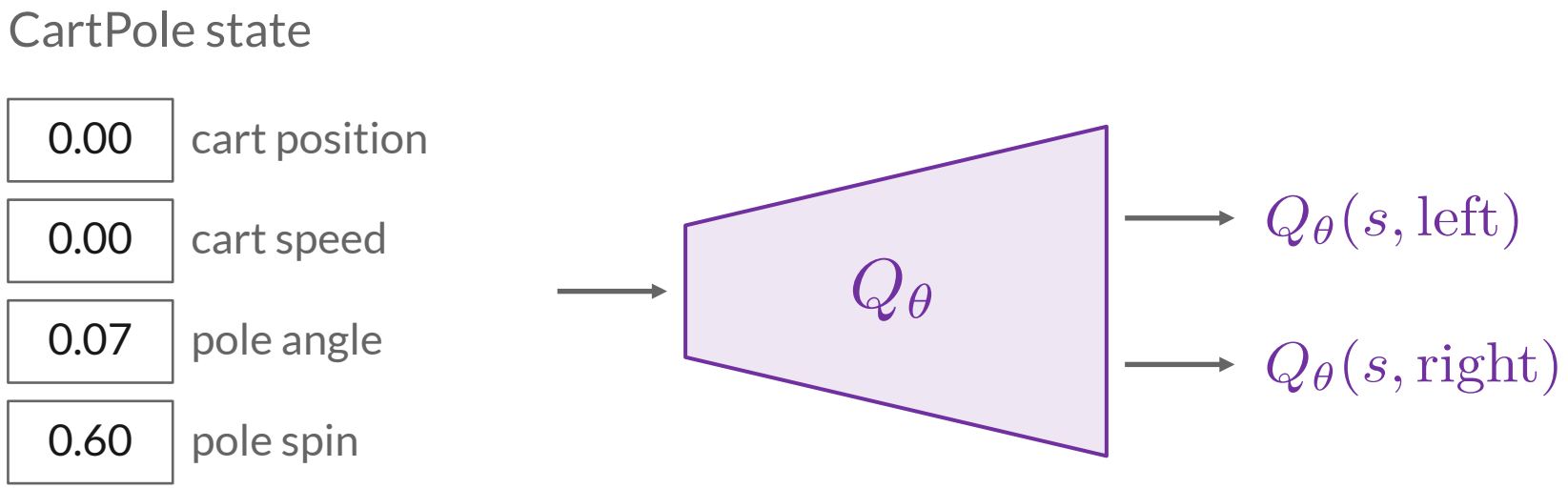
02 Returns and values

03 Learning from experience

04 Deep Q-networks

05 Capabilities and limits

A network instead of a table



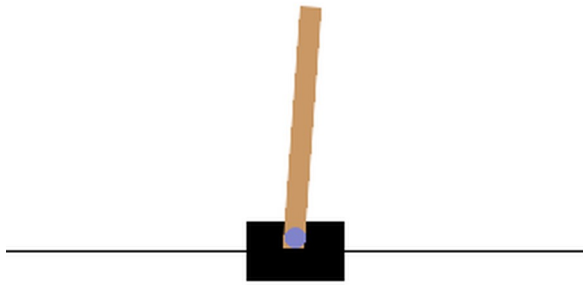
θ: all weights; one forward pass gives every action's value

Similar states get similar values

An update changes θ, not one table entry

What a trained network says

pole falling right

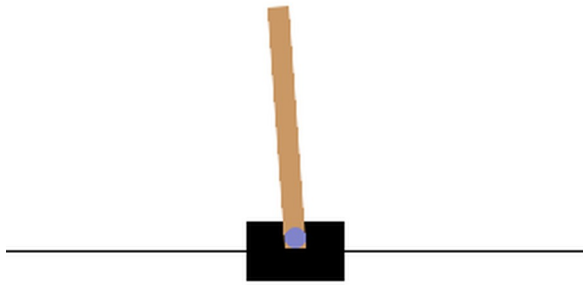


$Q(s, \text{left}) = 93.05$

$Q(s, \text{right}) = 93.86$

push right

pole falling left



$Q(s, \text{left}) = 94.84$

$Q(s, \text{right}) = 93.64$

push left

Our CartPole network after 80,000 steps; states set by hand

Values: expected discounted returns (at most 99.3), not probabilities

Actions differ by about 1; the argmax pushes under the falling pole

Source: CartPole-v1, Gymnasium

Q-learning as regression

For a transition (s, a, r, s', d): compute a target, fit the prediction

$$y \;=\; r + \gamma\,(1-d)\,\max_{a'} Q_\theta(s', a')$$

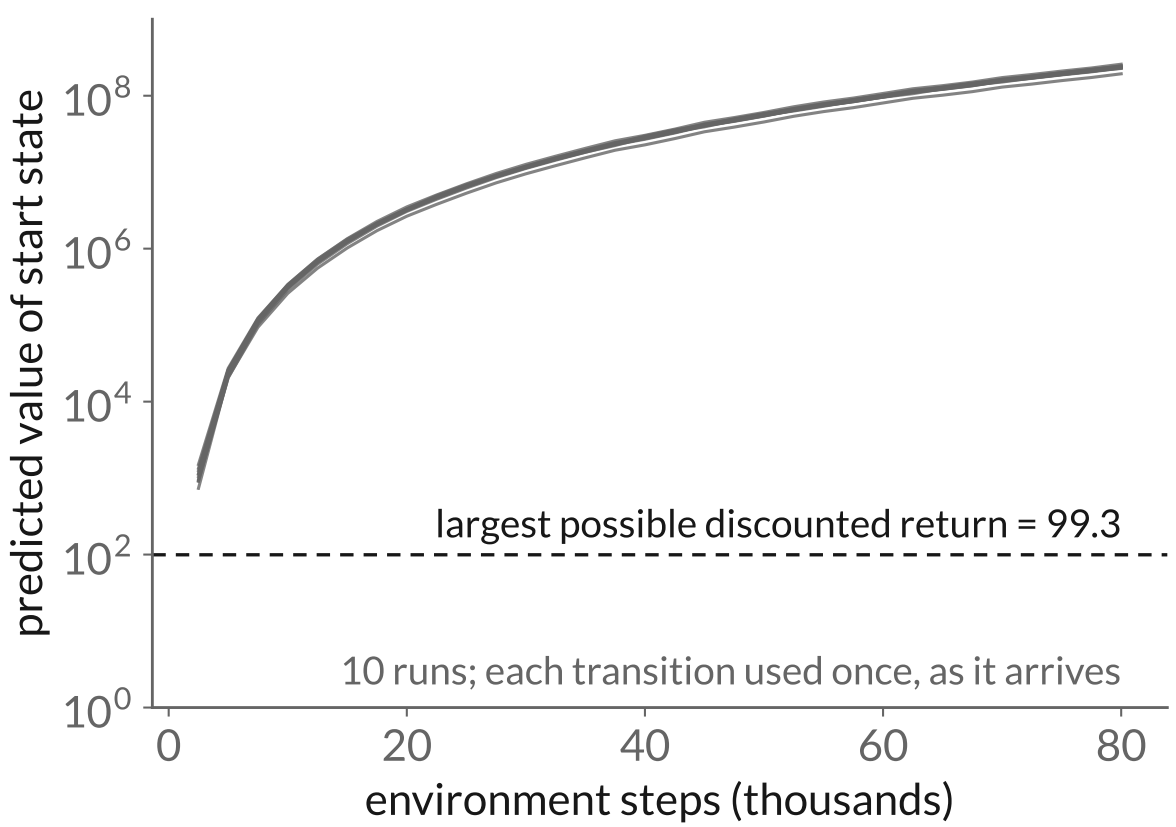
$$\ell(\theta) \;=\; \big(Q_\theta(s,a) - y\big)^2, \qquad \theta \;\leftarrow\; \theta - \eta\,\nabla_\theta\,\ell(\theta)$$

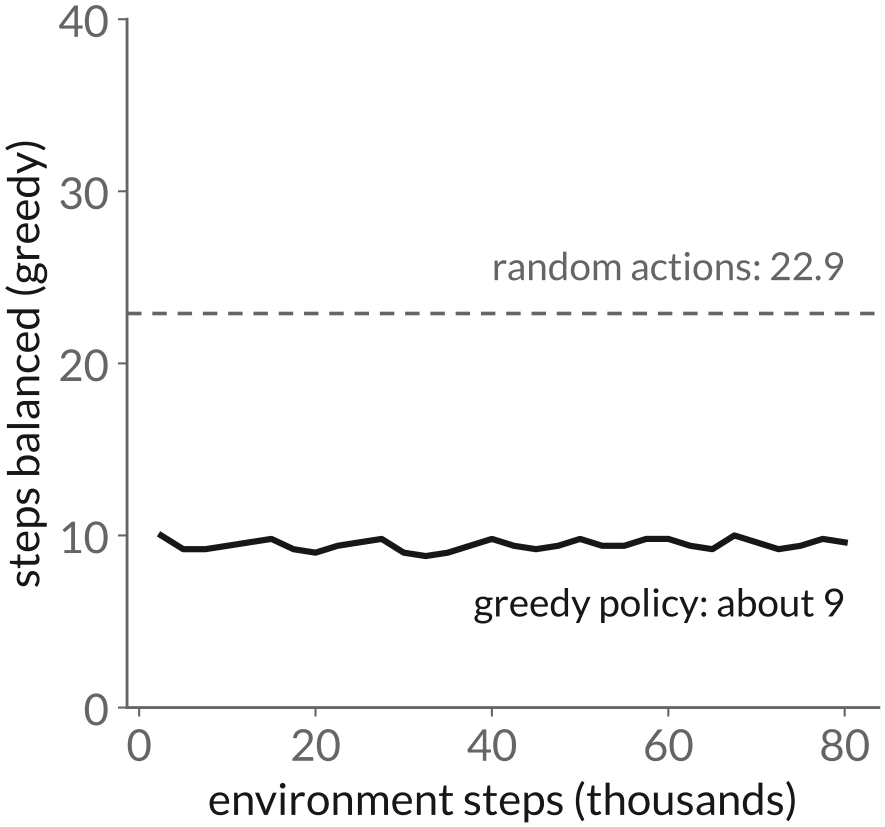
η: learning rate; ∇_θ: gradient with respect to the weights

Regression as in L1–L18, but the network computes its own label y

Simplest version: one update per new transition, then discard it

Plugging a network into Q-learning





CartPole, 10 runs; learning rates down to 2×10^{-5} also blow up

Values pass 10^8; no return can exceed 99.3

Greedy policy: 9 steps, worse than random

Two things went wrong

1. Consecutive data

Neighbouring transitions: almost the same state

Updates pull the network toward one narrow slice

Supervised learning shuffles for this reason

2. A target that moves

Target y uses the same θ being updated

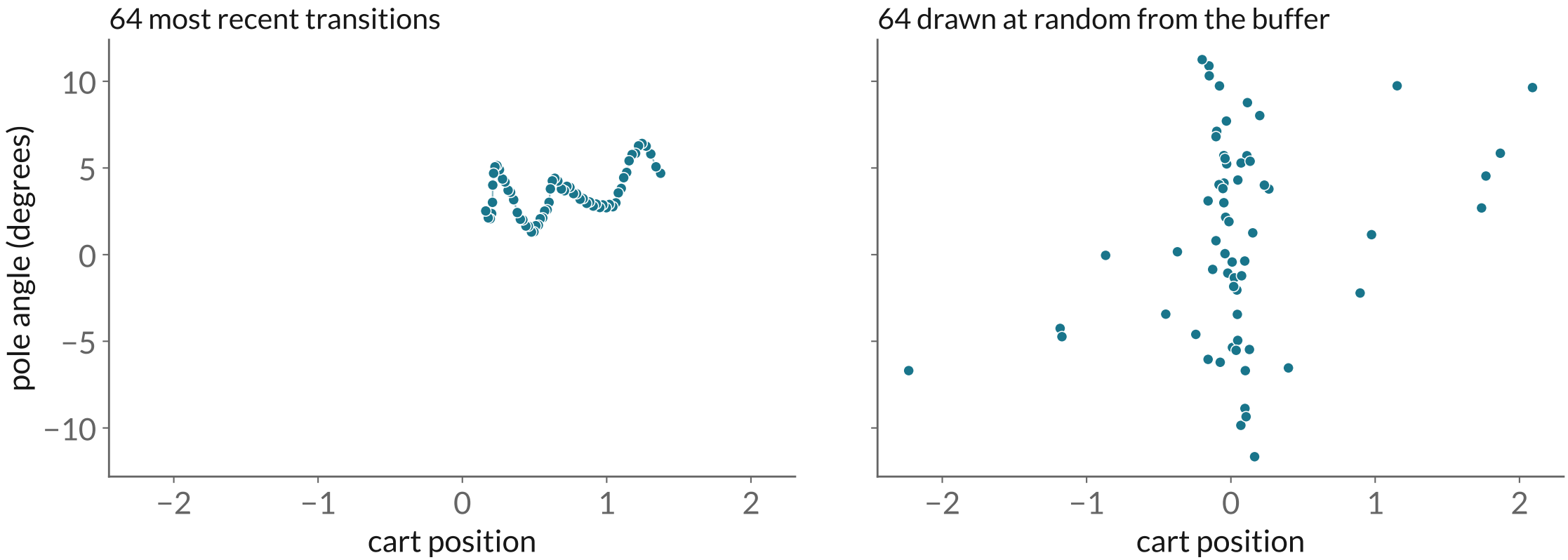
Raising $Q_\theta(s, a)$ also raises $Q_\theta(s', a')$

The network chases its own predictions

DQN: one fix for each, experience replay and a target network

Source: Mnih et al., Nature 2015

Consecutive transitions look alike

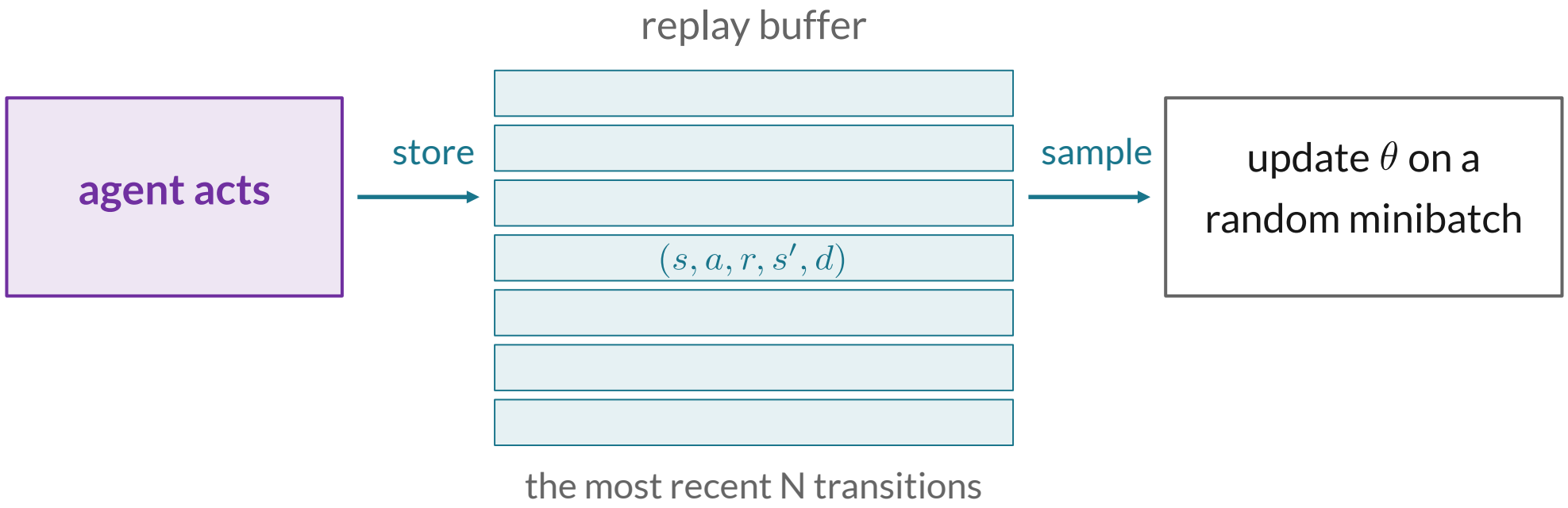


One minibatch of our CartPole run, at step 20,000

Left: one short stretch of one episode

Right: random picks from storage cover the whole range

Experience replay



Minibatches mix many episodes

Each transition is reused: about 63 times in our CartPole run

CartPole run: $N = 50{,}000$, batch 64. Atari DQN: $N = 10^6$, batch 32

Replay needs an off-policy method

Stored actions come from older, more random policies

The Q-learning target uses $\max_{a'}$, not the old policy's next action

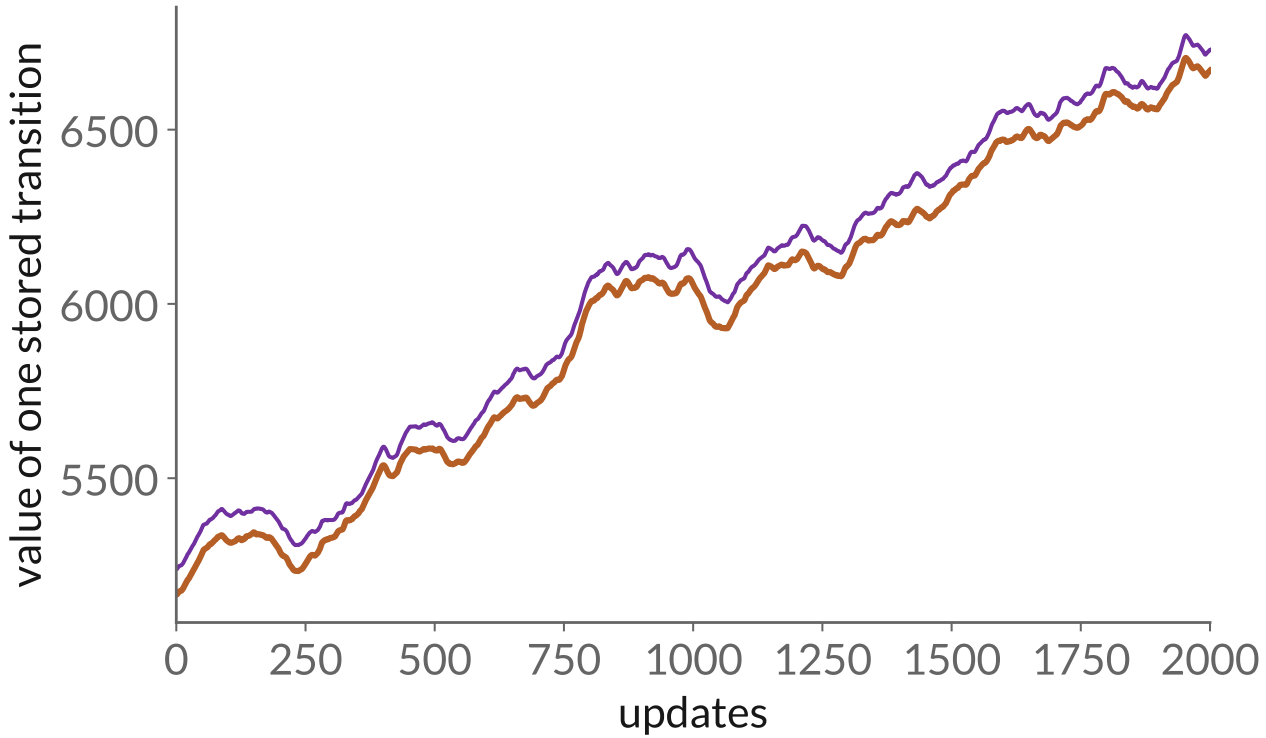
Old transitions stay valid: the grid on slide 44

Methods that evaluate the current behaviour cannot reuse old data

L20: policy gradients face exactly this problem

Off-policy learning is what makes replay possible

The target moves with every update



One stored transition over 2000 updates; target from the online network

Orange: its target y

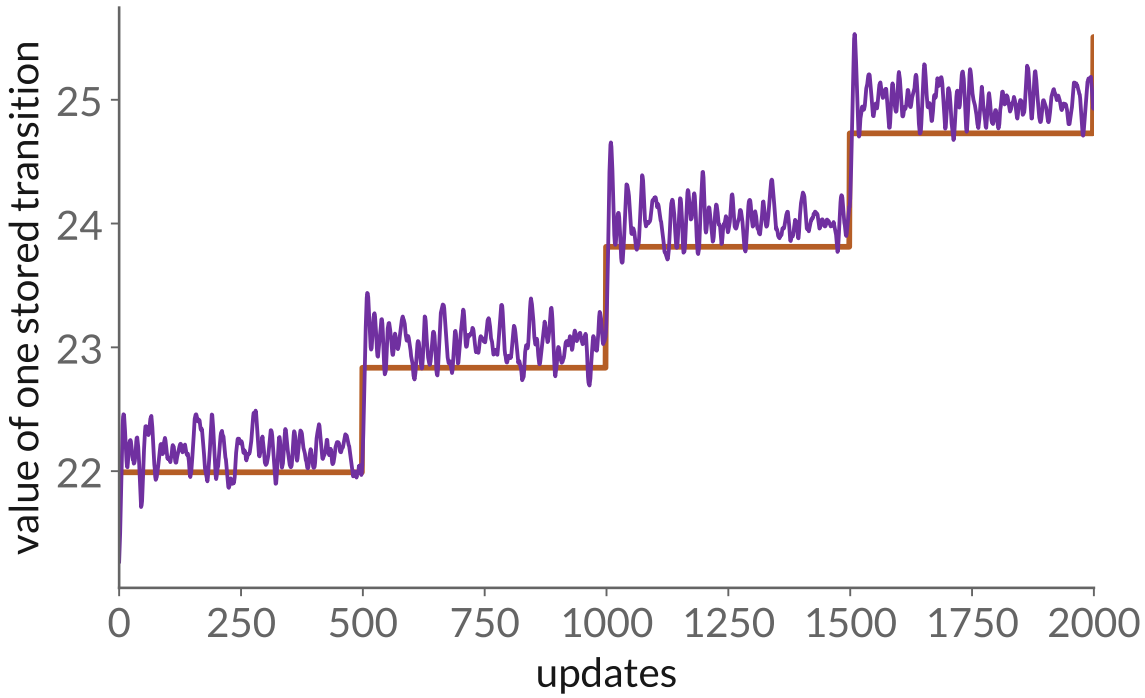
Purple: the prediction $Q_\theta(s, a)$

Target climbs from 5200 to 6700; the limit is 99.3

Each step toward the target moves the target

Target network: hold the target still

$$y = r + \gamma\,(1-d)\,\max_{a'} Q_{\theta^-}(s', a')$$



θ^- : a frozen copy of the weights

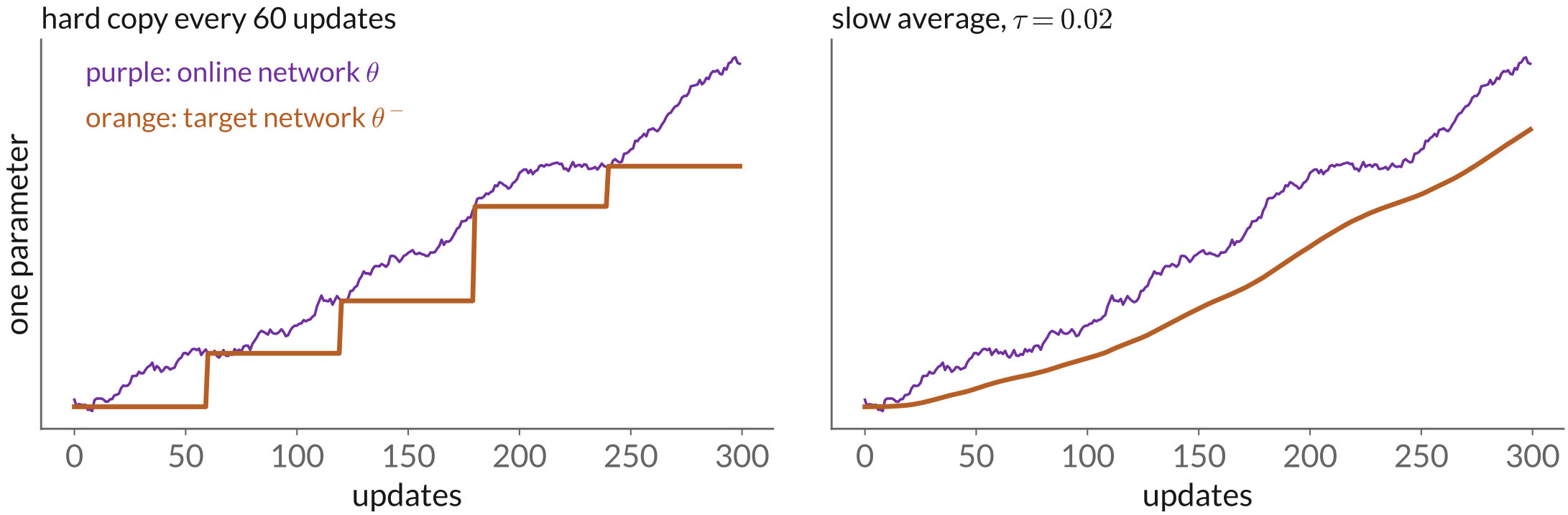
Every C updates: $\theta^- \leftarrow \theta$

Our run: $C = 500$

Same transition: the target is a staircase

In between: regression toward fixed labels

Hard copies or a slow average



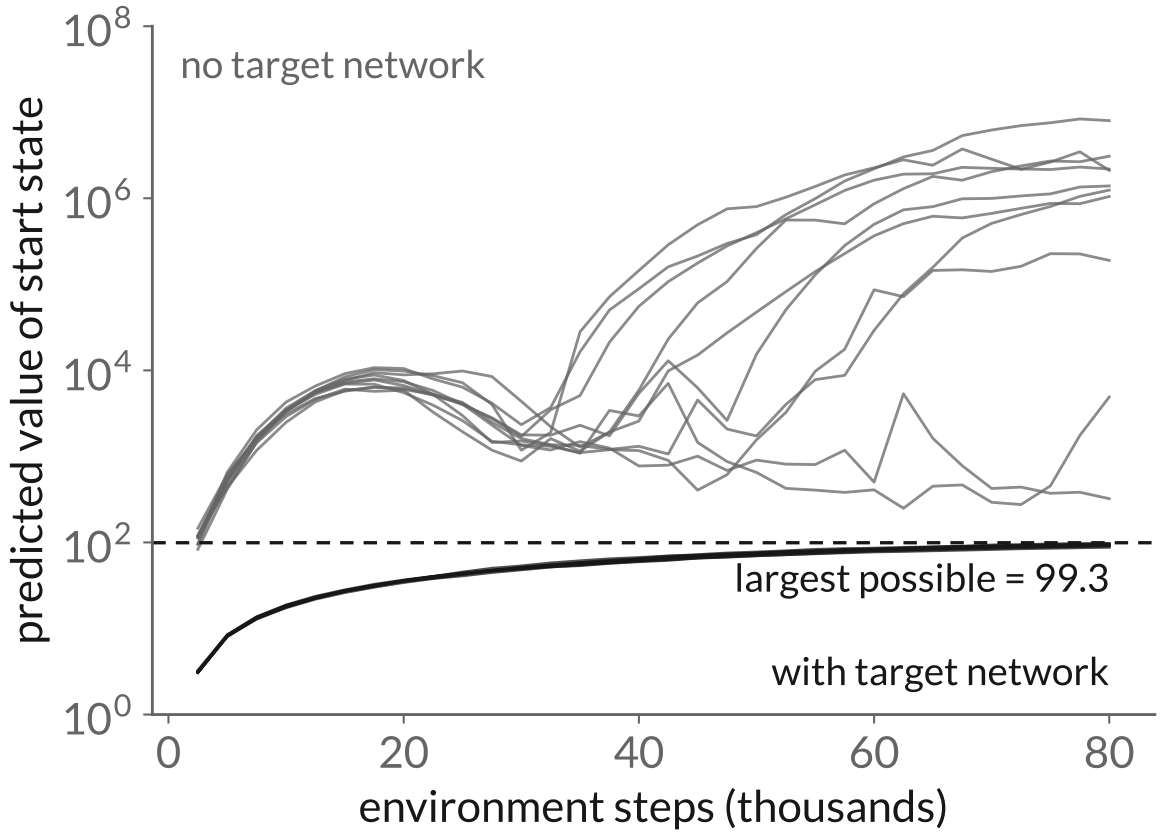
$$\text{slow average:} \quad \theta^- \leftarrow \tau\,\theta + (1-\tau)\,\theta^-$$

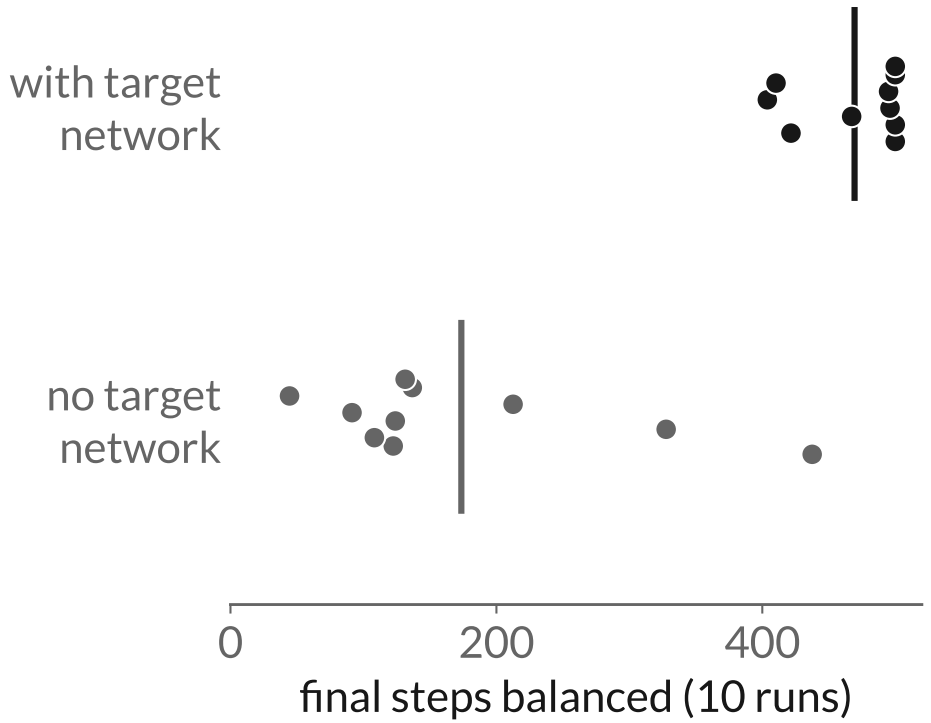
Hard copy: Atari DQN, every 10,000 updates. Slow average: often $\tau = 0.005$

Illustration with one made-up parameter

Source: Mnih et al., Nature 2015

With a target network, values stay sane





Both with replay; 10 runs each, differing only in the target

Without: peaks of 8×10^3 to 8×10^6. With: all below 99.3

Final steps balanced: 469 with the target network, 174 without

Stable in practice, not guaranteed

Three ingredients that together can make value estimates diverge:

		slide
Function approximation	an update changes θ: many values move at once	49
Bootstrapping	the target uses the network's own estimate	35
Off-policy data	transitions from older, more random behaviour	56

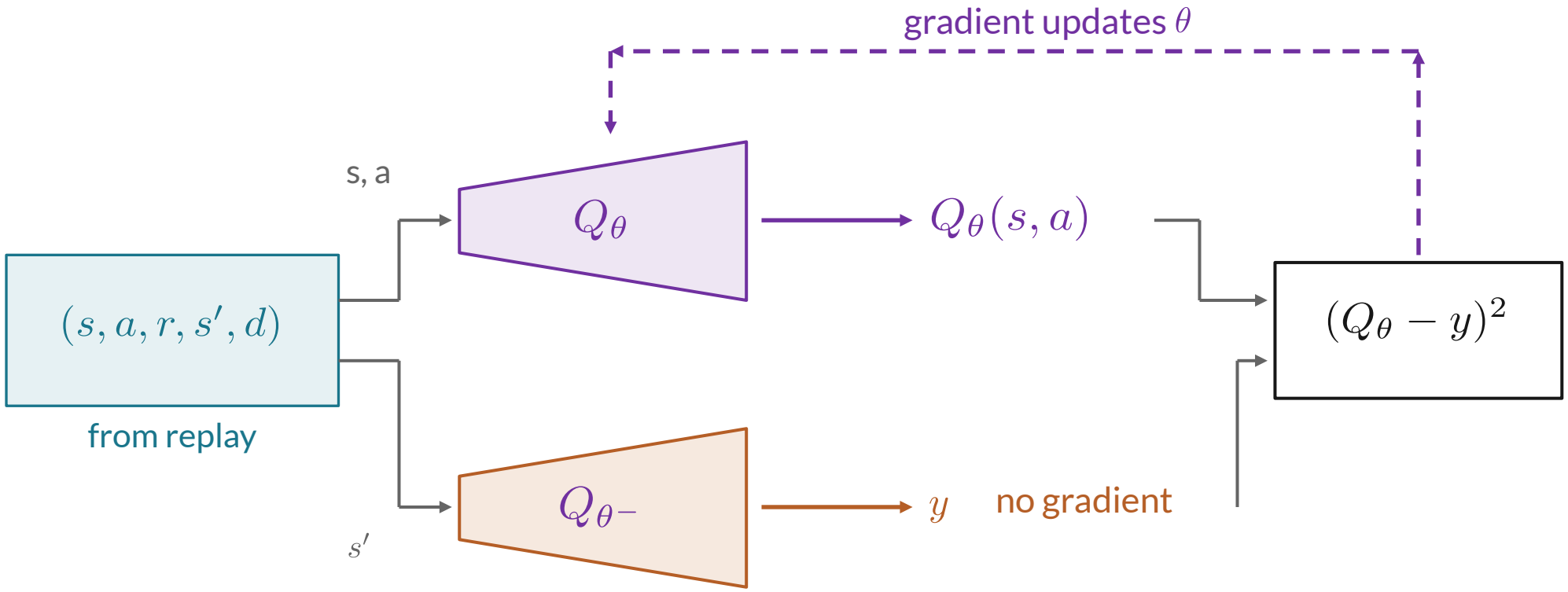
Sutton and Barto call this the deadly triad. DQN needs all three

The table in Section 03 had no function approximation: each update moved one entry

Replay and a target network make divergence rare, not impossible

Source: Sutton & Barto 2018

Gradients flow through the prediction only



The target y is a fixed label, as in supervised regression

Even if $\theta^- = \theta$: no gradient through y

The DQN loss

$$\mathcal{L}(\theta) = \underbrace{\mathbb{E}_{(s,a,r,s',d)\sim\mathcal{D}}}_{\text{minibatch from replay}}\left[\left(r + \underbrace{\gamma\,(1-d)\max_{a'} Q_{\theta^-}(s',a')}_{\text{target network, no gradient}} - \underbrace{Q_\theta(s,a)}_{\text{online network}} \right)^2\right]$$

$\mathcal{D}$: the replay buffer; $\mathbb{E}$: average over a minibatch

$d = 1$ if the stored step ended the episode (not a time limit)

Q-learning target, replay, target network and terminal mask in one line

The DQN algorithm

1. Fill the buffer with random play (Atari: 50,000 frames)

Repeat at every step:

2. Act ϵ-greedily with Q_θ; observe r, s', d; store (s, a, r, s', d)
3. Sample a random minibatch from the buffer
4. Targets from Q_{θ^-}; one gradient step for θ
5. Every C updates: $\theta^- \leftarrow \theta$
6. Lower ϵ slowly (Atari: 1 to 0.1 over a million frames)

Atari DQN: one update per 4 actions

Source: Mnih et al., Nature 2015

What the published ablation shows

	replay + target	replay only	target only	neither
Breakout	**316.8**	240.7	10.2	3.2
Enduro	**1006.3**	831.4	141.9	29.1
River Raid	**7446.6**	4102.8	2867.7	1453.0
Seaquest	**2894.4**	822.6	1003.0	275.8
Space Invaders	**1088.9**	826.3	373.2	302.0

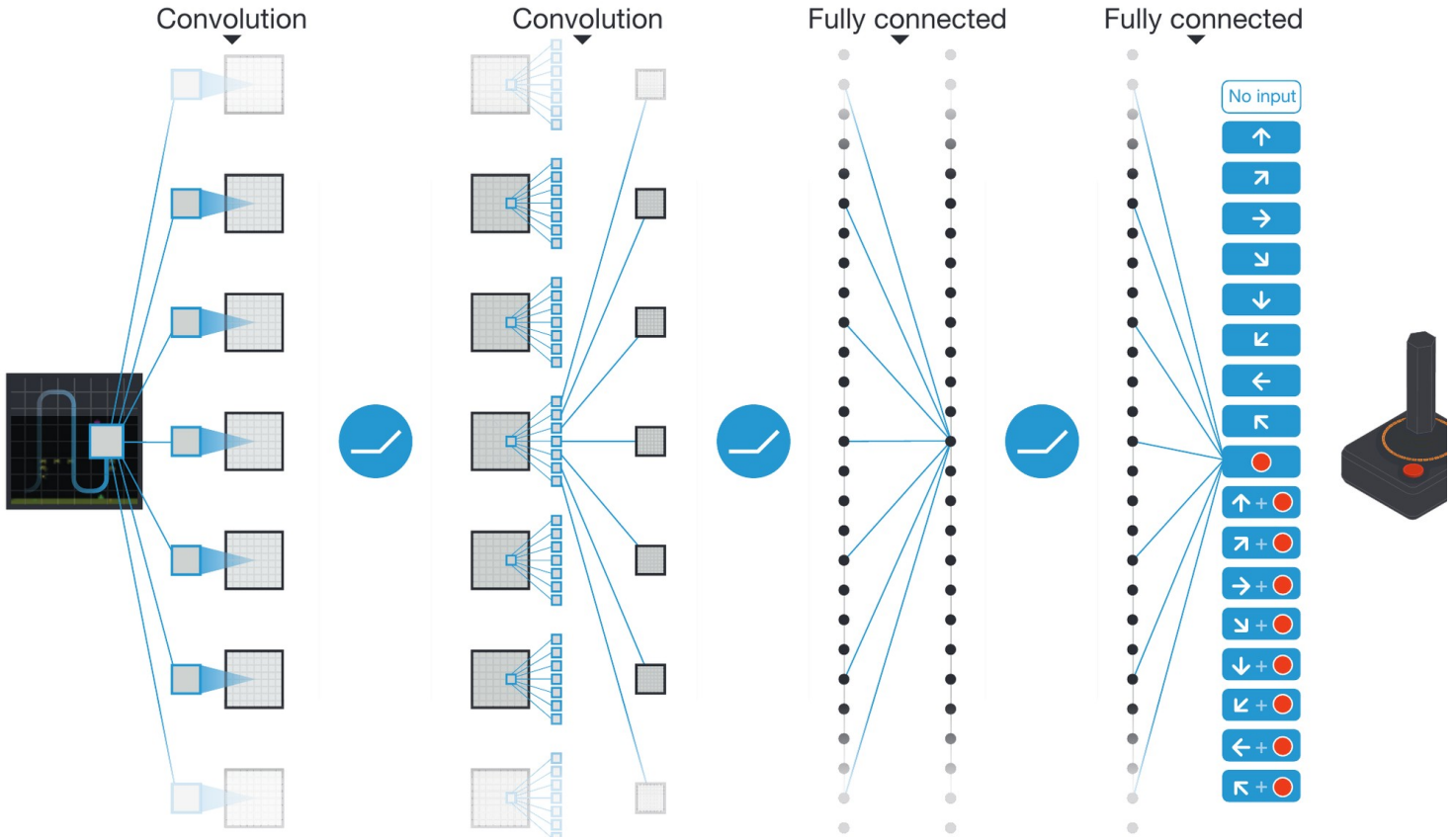
Scores after 10 million frames

Removing either part hurts every game

Replay matters most in 4 of 5 games; the target network adds on top

Source: Mnih et al., Nature 2015

DQN on Atari: the network



Input: last 4 frames, 84×84 (slide 11)

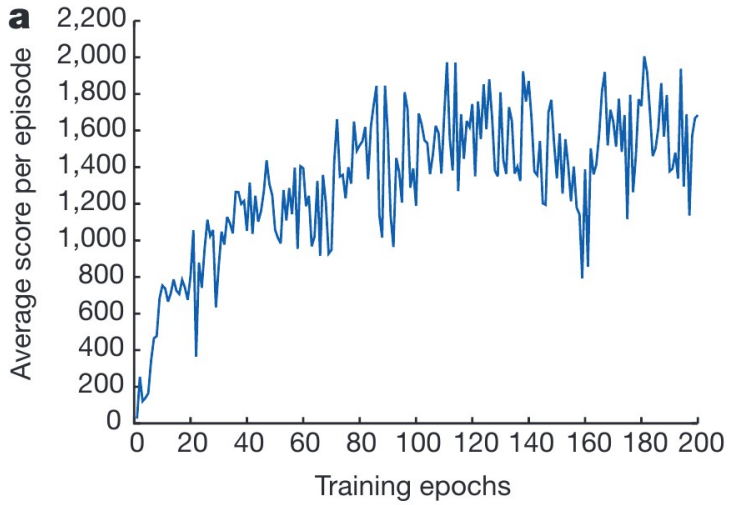
3 convolution layers, then 1 fully connected hidden layer

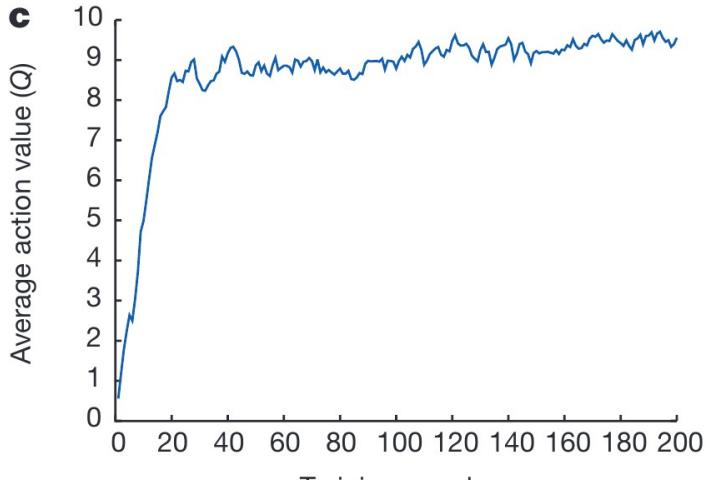
Output layer: one value per action

Rewards clipped to $-1, 0, +1$

Source: Mnih et al., Nature 2015

Training curves on Atari





Space Invaders; one epoch = 50,000 updates

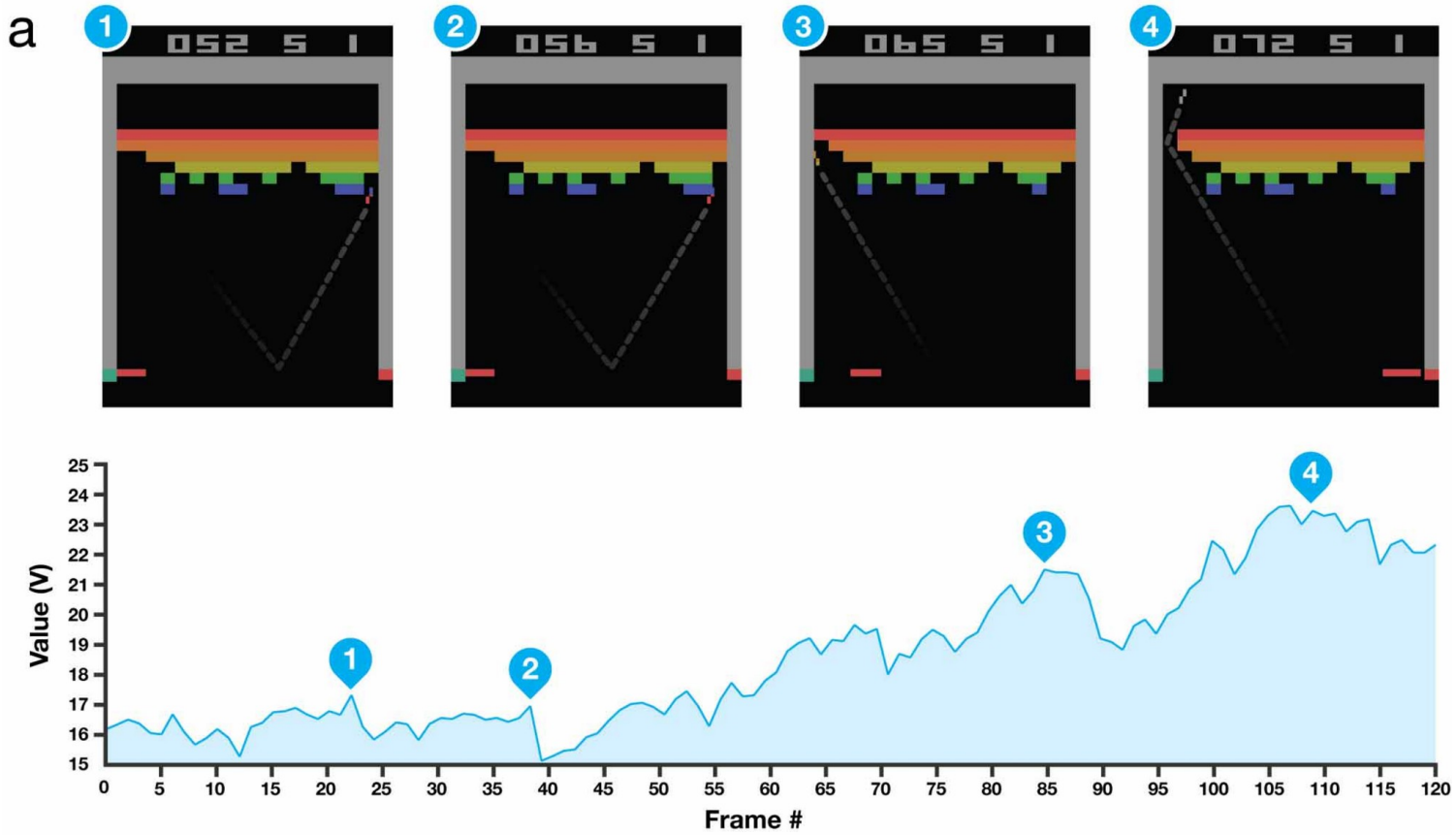
Top: score per episode, $\epsilon = 0.05$

Bottom: average predicted $\max_a Q$ on fixed states

Both rise: values track how well the agent plays

Source: Mnih et al., Nature 2015

Reading a learned value function



Breakout: predicted value V

1, 2: bottom bricks, about 17

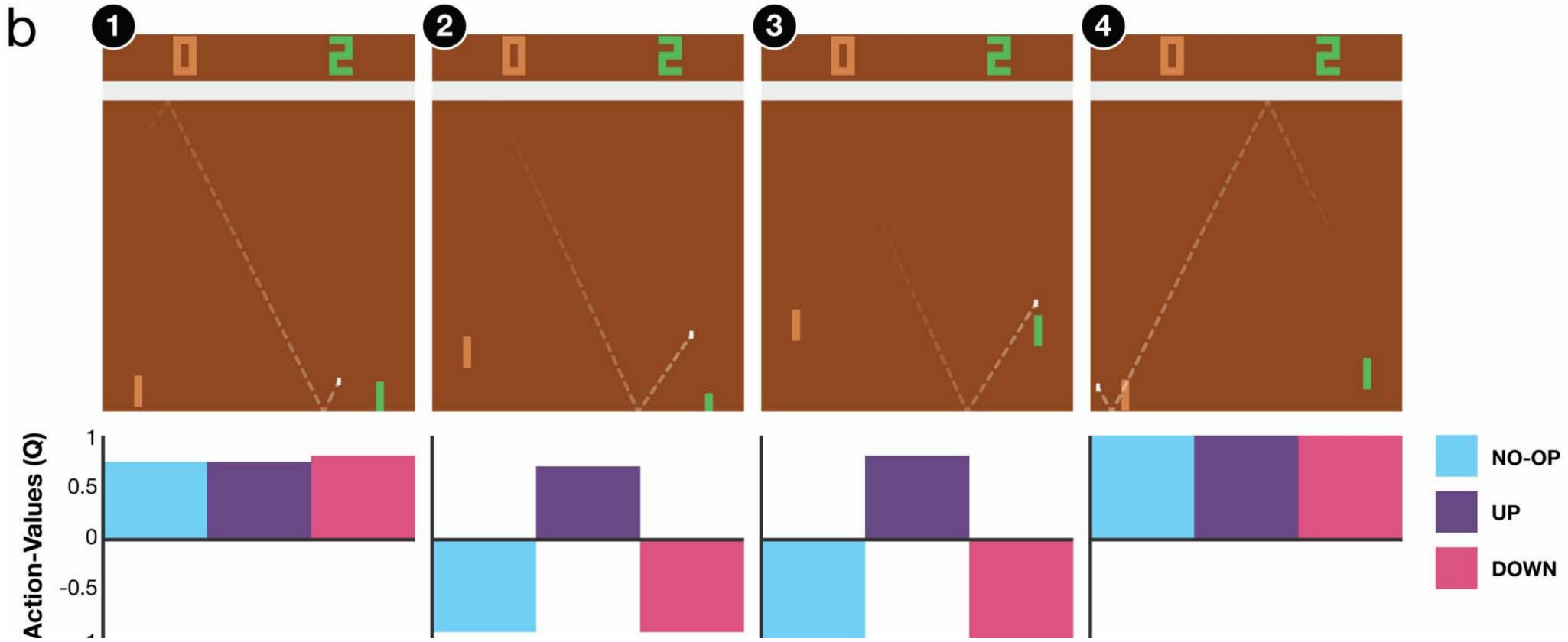
3: about to break through, about 21

4: through the top, above 23

The value rises before the reward arrives

Source: Mnih et al., Nature 2015

Action values in Pong

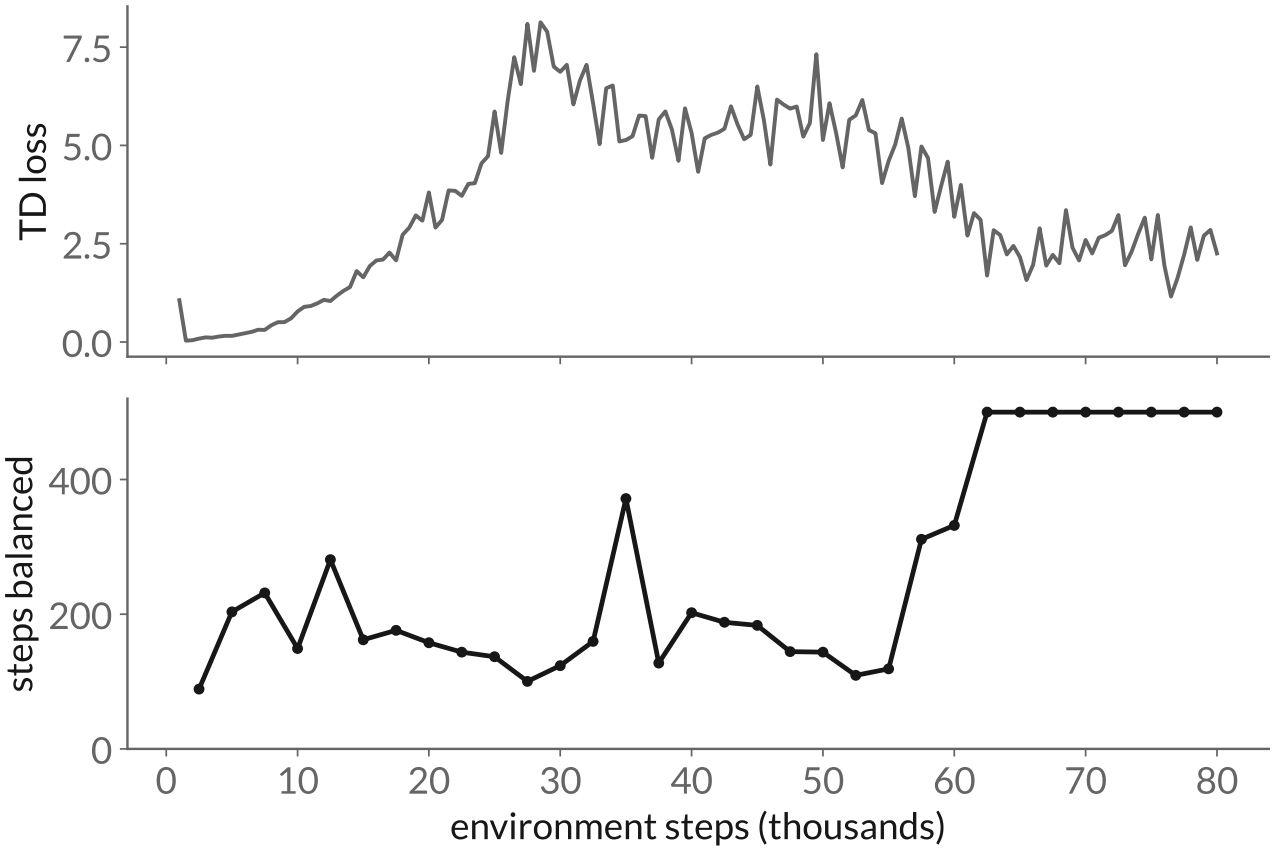


Bars: predicted Q for no-op, up, down

2: down would lose the ball, about -0.9. 4: all near $+1$

Source: Mnih et al., Nature 2015

TD loss is not a progress meter



Our CartPole run with the target network

Loss grows from 0.05 to 8 while play improves

Larger values: larger targets and errors

Measure progress by playing: average return

Lecture outline

01 Decisions and rewards

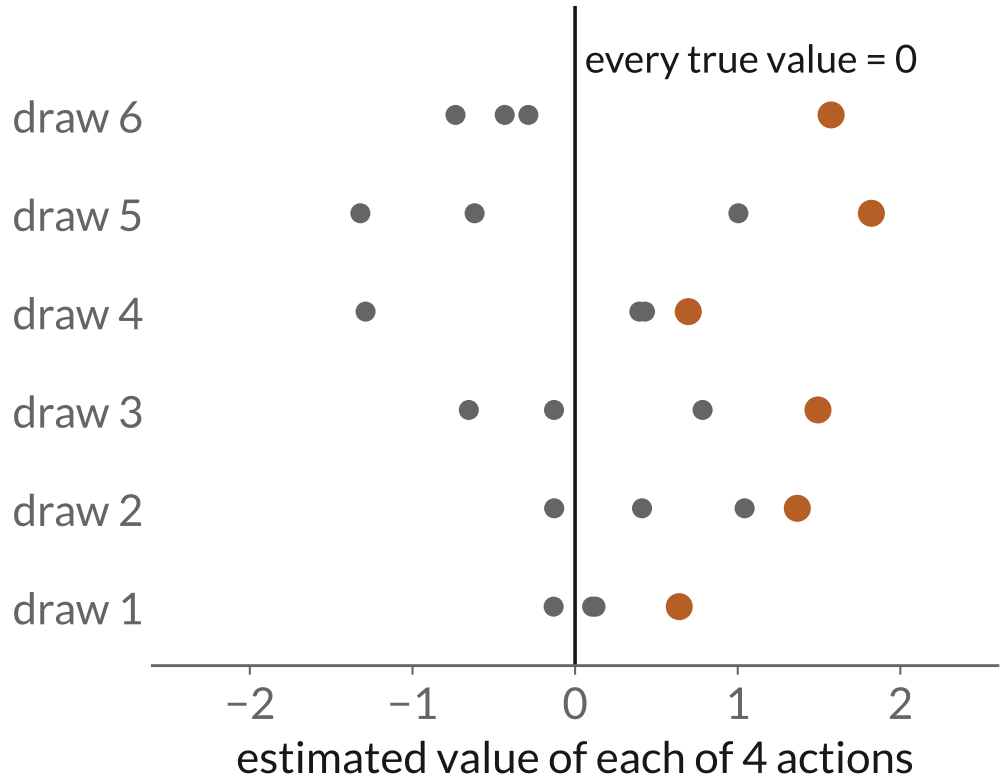
02 Returns and values

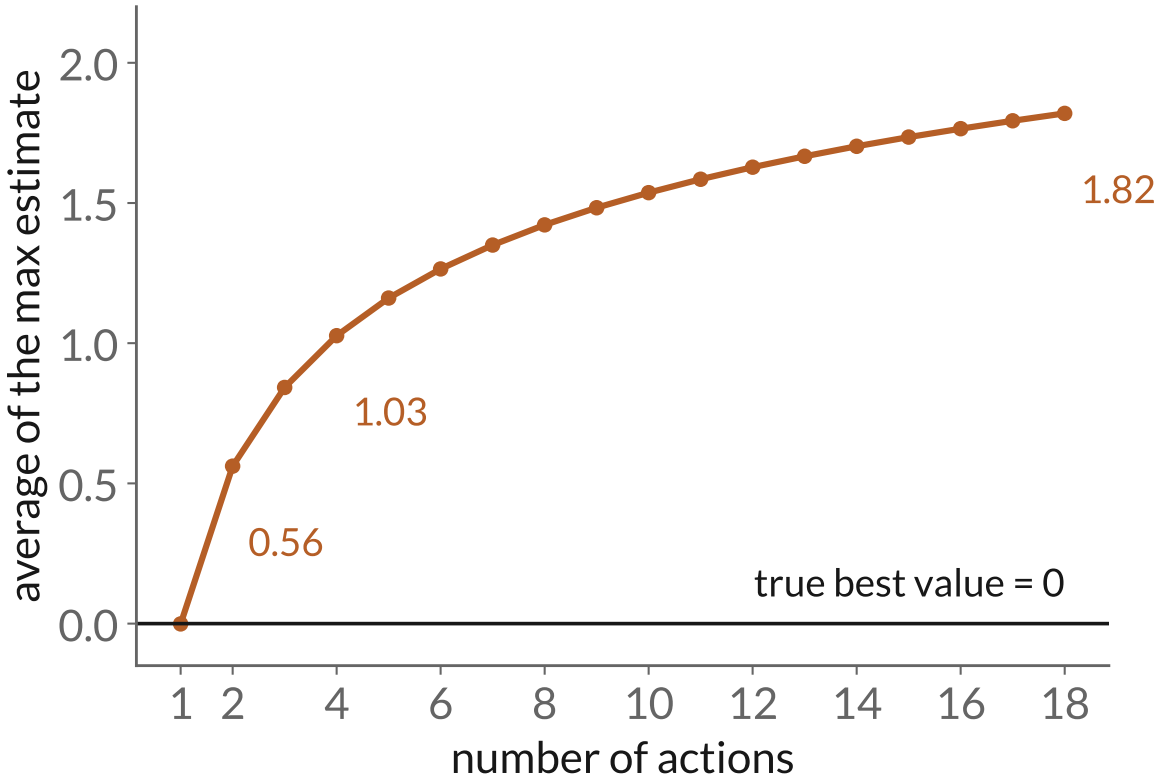
03 Learning from experience

04 Deep Q-networks

05 Capabilities and limits

The max picks lucky estimates



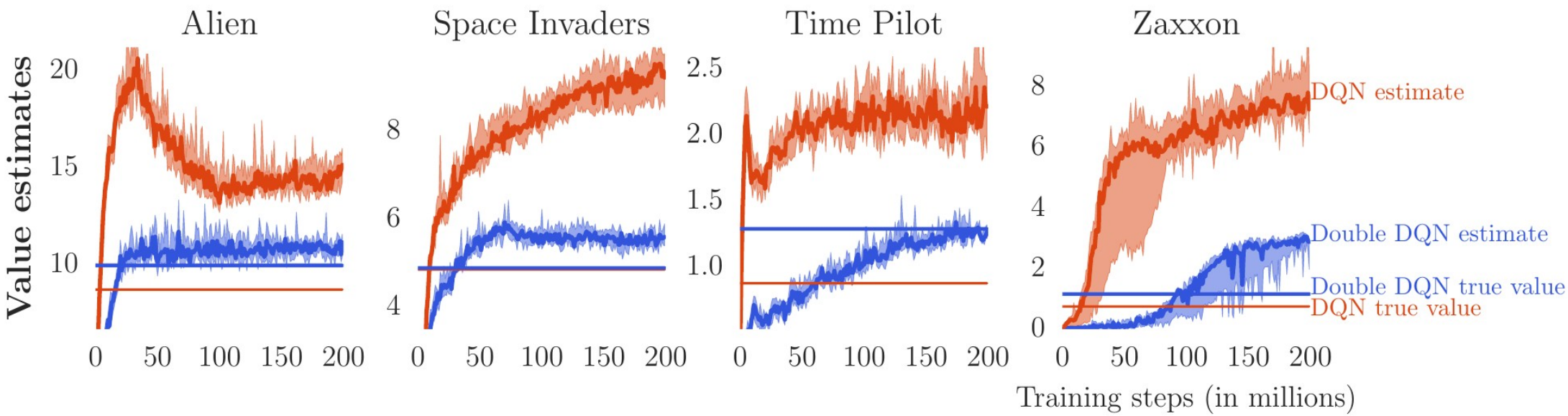


Every action truly worth 0; estimates off by noise (standard deviation 1)

Their max averages $+1.03$ for 4 actions, $+1.82$ for 18

The Q-learning target takes this max: errors lean upward

DQN overestimates on Atari



Curves: predicted value. Flat lines: actual return of the final policy

Orange: DQN. Blue: Double DQN (next slide)

DQN's estimates end well above what it really earns

Source: van Hasselt et al. 2016

Double DQN: choose and score separately

$$y = r + \gamma\,(1 - d)\; Q_{\theta^-}\Big(s',\; \arg\max_{a'} Q_\theta(s', a')\Big)$$

score: target network

choose: online network

DQN: the same estimates pick the action and report its value

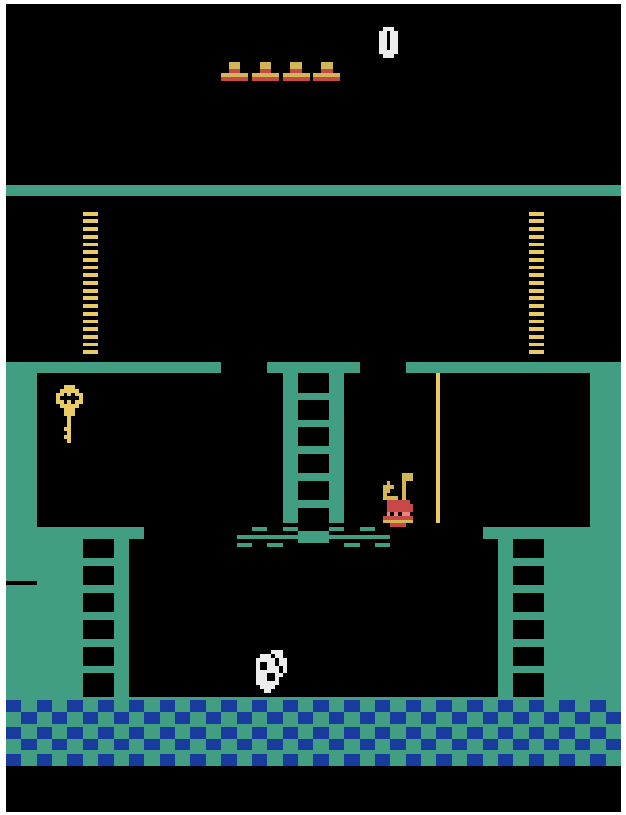
Double DQN: a lucky overestimate in one network is rarely shared by the other

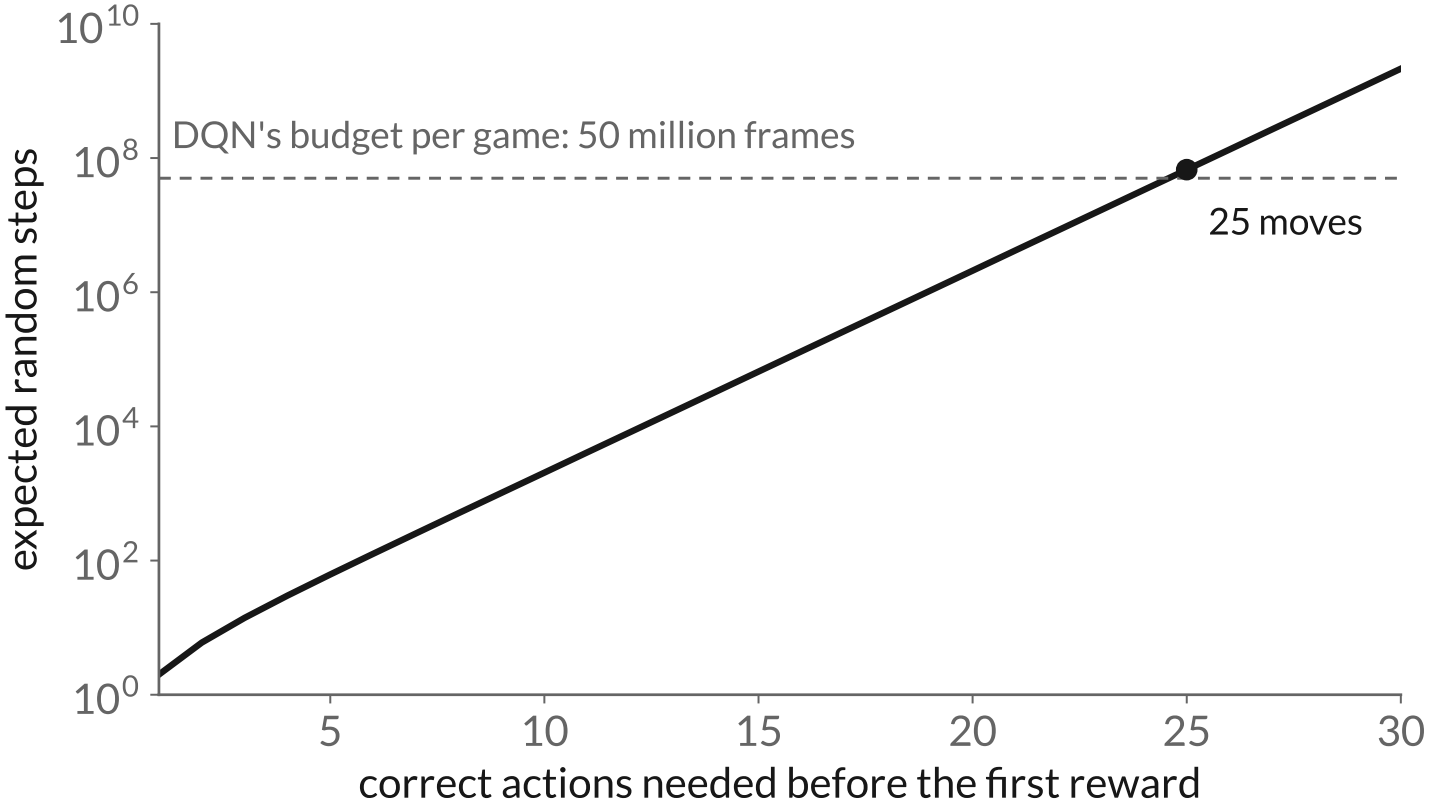
One-line change to the target

49 games, median vs. the human tester: 93.5% (DQN), 114.7% (Double DQN)

Source: van Hasselt et al. 2016

Sparse rewards: nothing to learn from





Montezuma's Revenge: a long exact sequence before the first point; DQN 0%

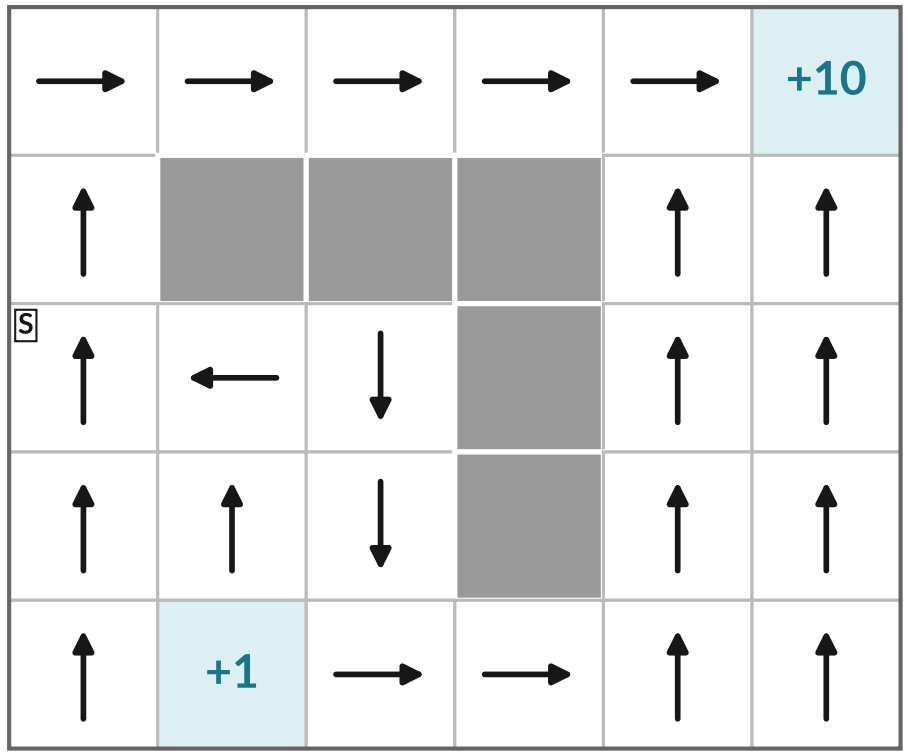
Plot: k correct moves in a row, mistakes restart; random play needs $2^{k+1} - 2$ steps

No reward: every Q target is 0, nothing to learn

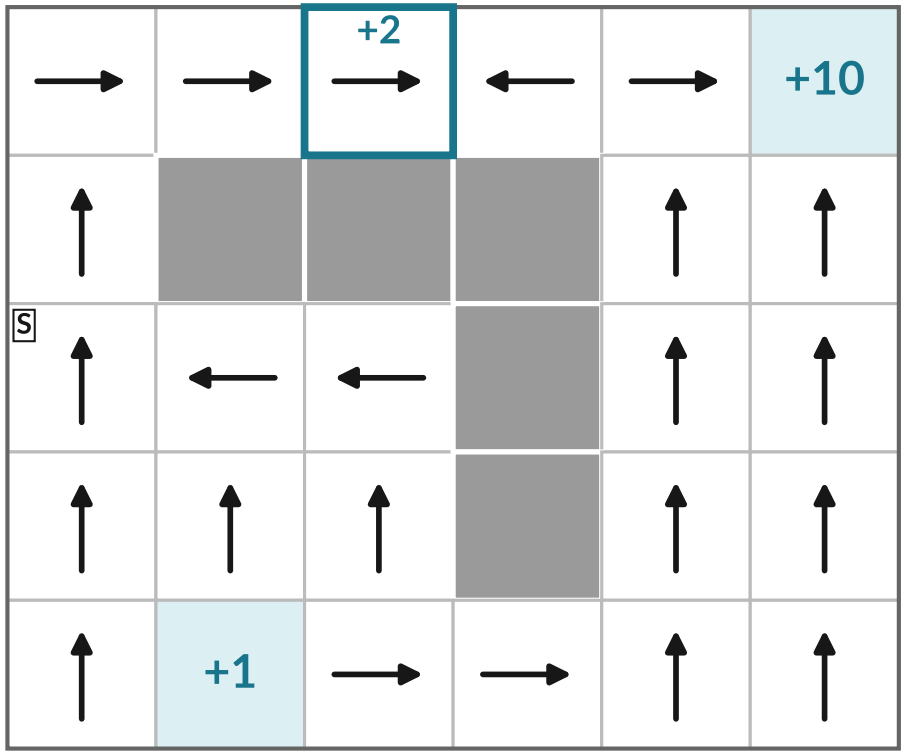
Source: frames: Arcade Learning Environment · Mnih et al., Nature 2015

The agent optimizes the reward you wrote

reward only at the deliveries



plus a +2 bonus for entering the corridor cell



Right: a +2 bonus meant to encourage the trip to the delivery

Best policy (exact): step out and back in forever, $2/(1-0.9^2) = 10.5$

Reward is the task's only specification; shortcuts get exploited

Learning needs a lot of play

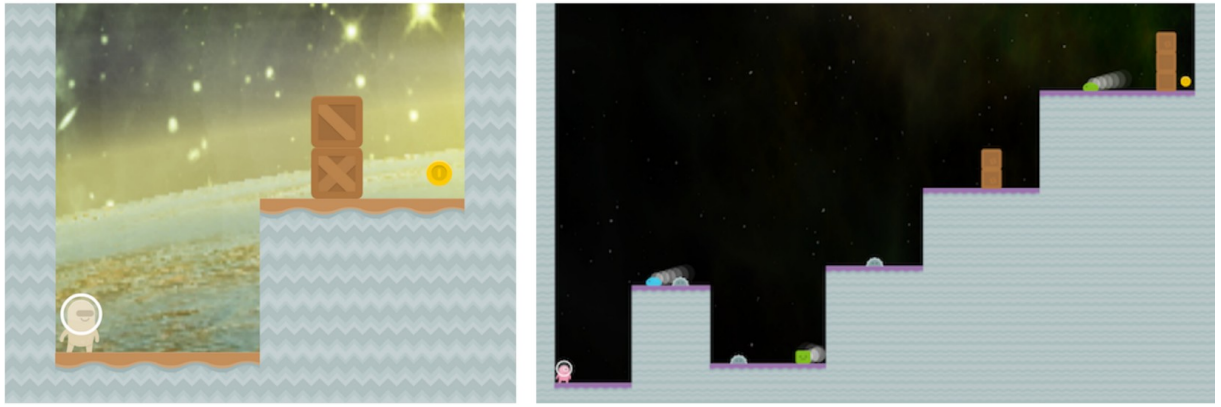
50 million frames	DQN's training per game: about 38 days of play
about 2 hours	the human tester's practice
80,000 steps	our CartPole runs: 4 numbers, 2 actions

A real robot cannot afford 38 days of crashes

Replay reuses data; learning from reward alone is still slow

Source: Mnih et al., Nature 2015

Trained levels and new levels

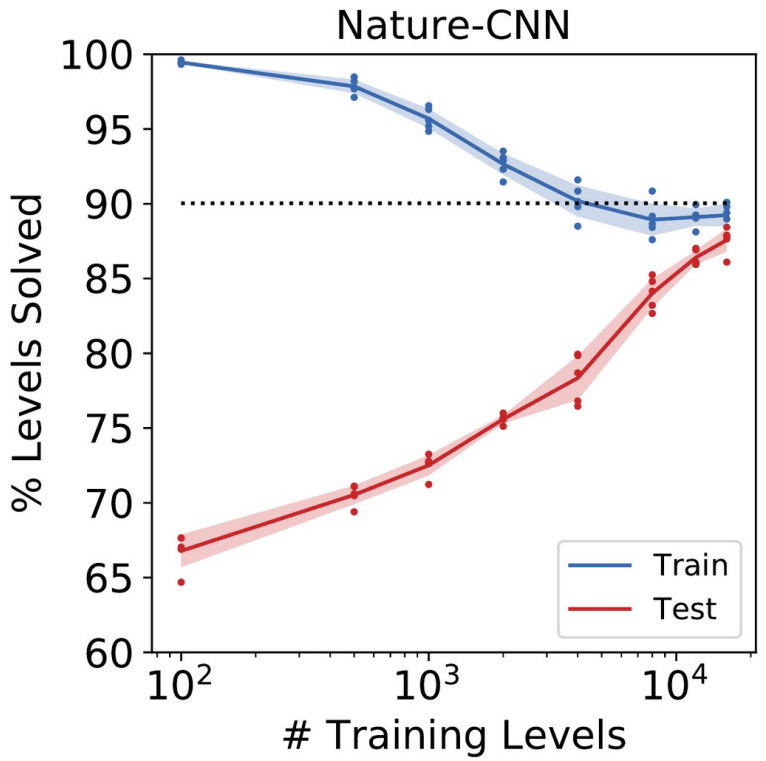


CoinRun: random levels; reach the coin

DQN's network, trained with PPO (L20)

100 training levels: 99% solved, 67% of new ones

16,000 training levels: 89% and 88%



RL agents can memorize their training worlds

Source: Cobbe et al. 2019

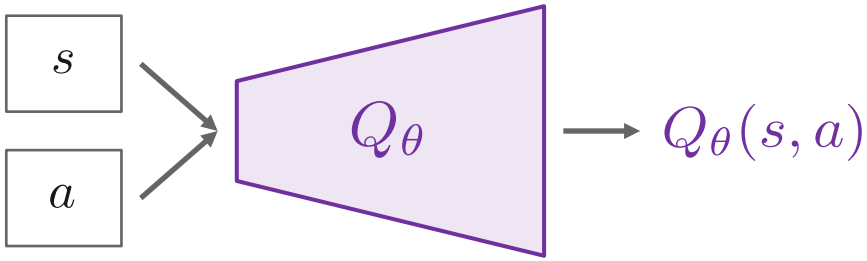
Continuous actions break the max

Atari: 18 joystick actions



One pass: every action's value
Pick the tallest bar

Robot arm: 7 torques, each a real number



One pass scores one action a
Infinitely many a: finding the best is a search

Acting and every target need $\max_a Q$; 10 levels per joint: 10^7 outputs

L20: learn the policy $\pi_\theta(a \mid s)$ directly, helped by a value function

Policy gradients, actor–critic, PPO

Each idea answers a problem

Problem		Answer	slide
A plain sum: quick and slow score alike; can be infinite	→	**Discount γ**	17
Value iteration needs the rules P and r	→	**Learn from transitions**	32
Monte Carlo waits for the end of the episode	→	**TD: bootstrap every step**	35
Acting on V needs a look-ahead through P	→	**Learn Q, take the argmax**	38
Always greedy never finds the $+10$	→	**ϵ-greedy exploration**	43
Too many states for a table	→	**A network Q_θ**	49
Consecutive transitions look alike	→	**Experience replay**	55
The target moves with every update	→	**Target network**	58
The max picks lucky estimates	→	**Double DQN**	74

Still open: sparse rewards, reward design, sample cost, generalization, continuous actions

Summary

No labels: late rewards; the agent's actions decide its data

Return: discounted reward sum. V , Q: its average from a state (and action)

Bellman: value now = reward + $\gamma \times$ value next

Q-learning: single transitions, greedy policy from any behaviour

DQN = Q-learning + network + experience replay + target network

Fixable: overestimation (Double DQN)

Harder: sparse rewards, reward design, sample cost, generalization, continuous actions

Next lecture

L20 · Policy gradients, actor–critic, and PPO

Parameterize the policy and climb the expected return

Today's V and Q reduce the variance of that climb

Continuous actions work; needs fresh data from the current policy