Practical Deep RL Implementation Techniques

CS 224R

Reminders

Today:	Homework 1 due, Homework 2 out
Wed next week:	Project proposal due

The Plan

Recap & finish Q learning

Q learning tricks

Improving Q learning

Case studies: games, robotics

Key learning goals:

- Practical Q learning implementation tricks
- Understanding the landscape of Q learning algorithms

The Plan

Recap & finish Q learning

Q learning tricks

Improving Q learning

Case studies: games, robotics

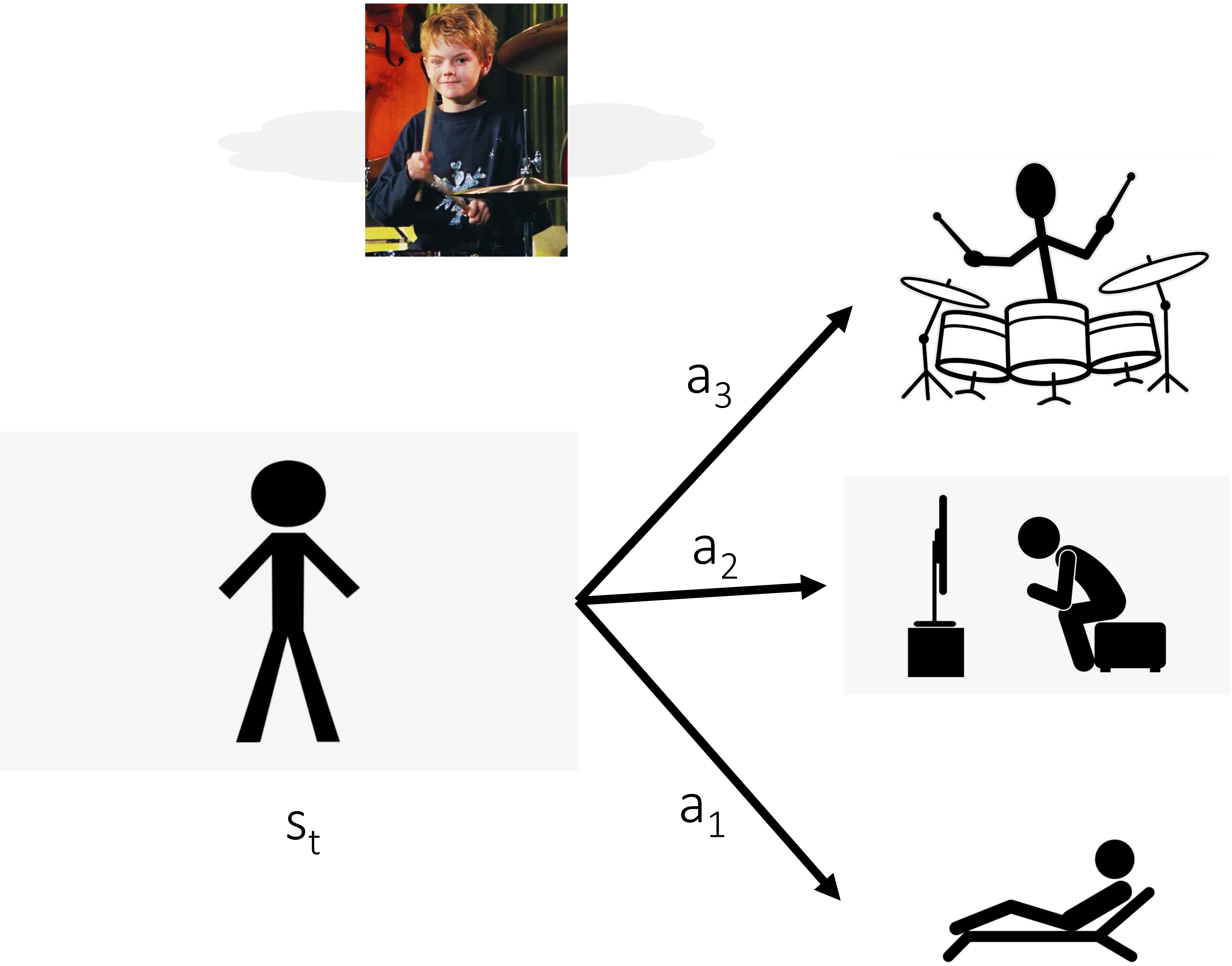
Value-Based RL

Value function: $V^{\pi}(\mathbf{s}_t) = ?$

Q function: $Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = ?$

Advantage function: $A^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = ?$

Reward = 1 if I can play it in a month, 0 otherwise



How can we improve the policy?

Current $\pi(\mathbf{a}_2|\mathbf{s}) = 1$

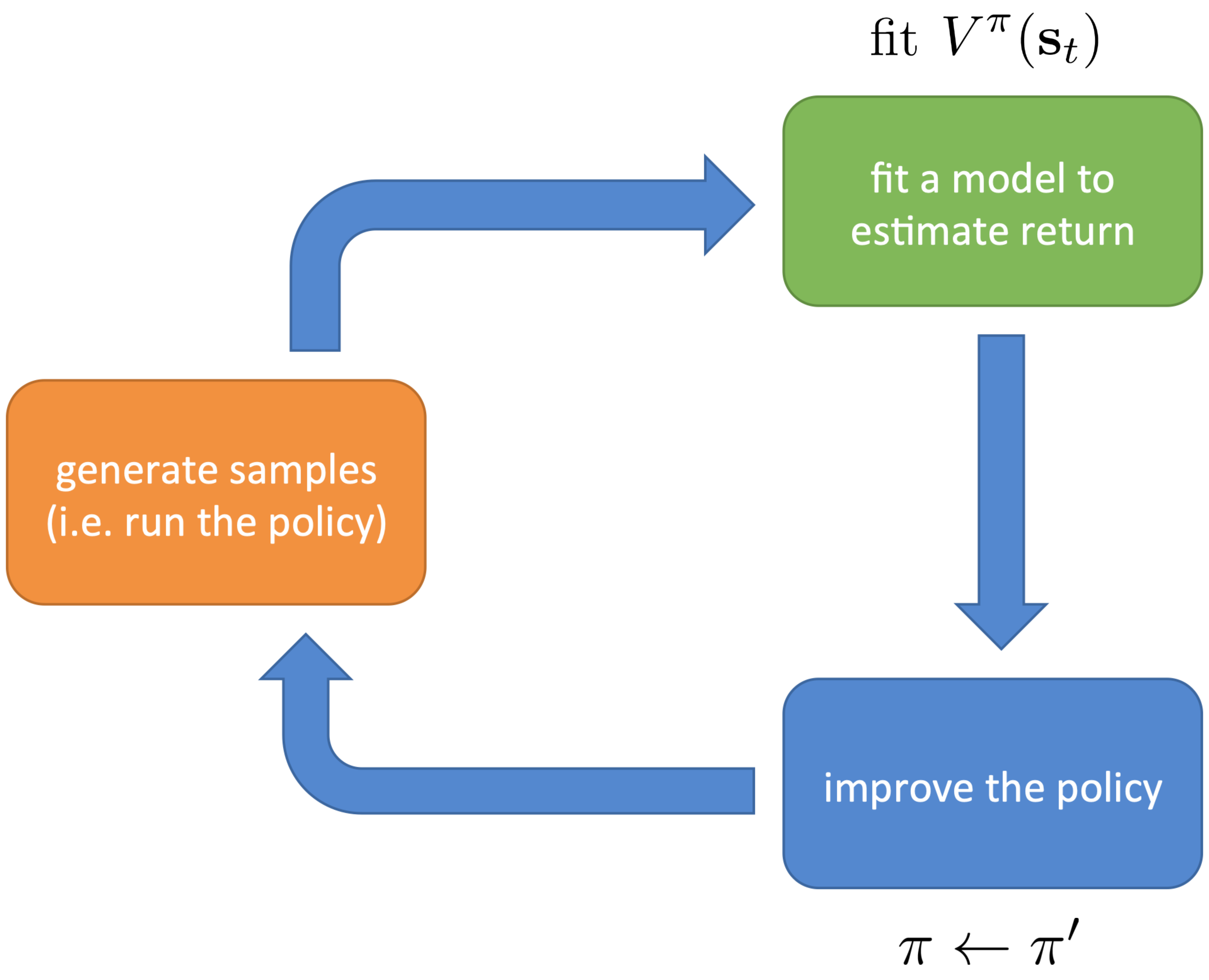
Policy Iteration

policy iteration algorithm:

1. evaluate $A^\pi(\mathbf{s}, \mathbf{a})$
2. set $\pi \leftarrow \pi'$

$$\pi'(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 \text{ otherwise} \end{cases}$$

as before: $A^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma E[V^\pi(\mathbf{s}')] - V^\pi(\mathbf{s})$



Value Iteration

policy iteration algorithm:

1. evaluate $Q^\pi(\mathbf{s},\mathbf{a})$
2. set $\pi \leftarrow \pi'$

$$\pi'(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q^\pi(\mathbf{s},\mathbf{a}) \\ 0 \text{ otherwise} \end{cases}$$

$$A^\pi(\mathbf{s},\mathbf{a}) = r(\mathbf{s},\mathbf{a}) + \gamma E[V^\pi(\mathbf{s}')] - V^\pi(\mathbf{s})$$

$$\arg\max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t,\mathbf{a}_t) = \arg\max_{\mathbf{a}_t} Q^\pi(\mathbf{s}_t,\mathbf{a}_t)$$

$$Q^\pi(\mathbf{s},\mathbf{a}) = r(\mathbf{s},\mathbf{a}) + \gamma E[V^\pi(\mathbf{s}')] \text{ (a bit simpler)}$$

skip the policy and compute values directly!

value iteration algorithm:

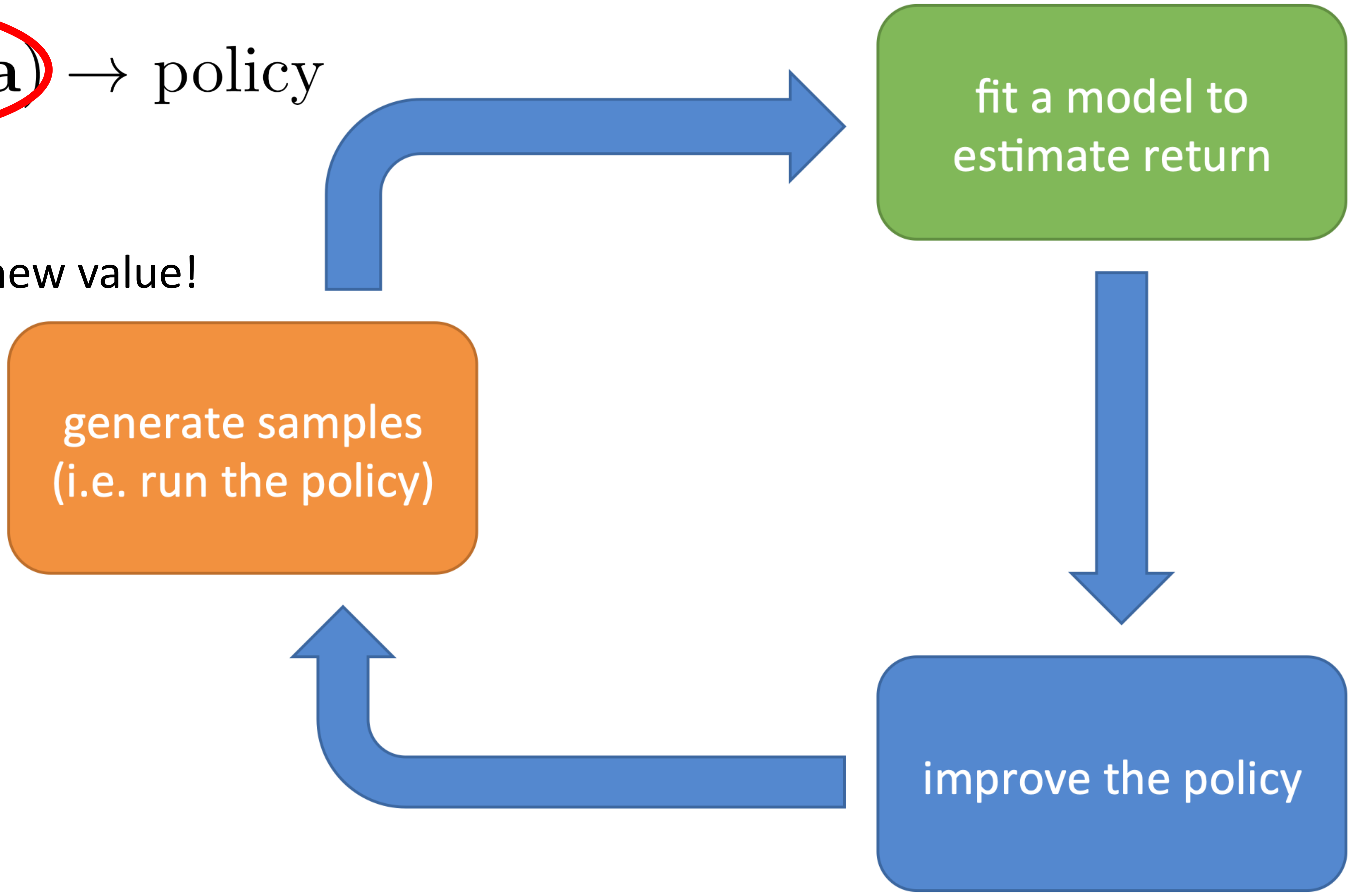
1. set $Q(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + \gamma E[V(\mathbf{s}')]$
2. set $V(\mathbf{s}) \leftarrow \max_{\mathbf{a}} Q(\mathbf{s},\mathbf{a})$

$$Q^\pi(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + \gamma E_{\mathbf{s}'\sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})}[V^\pi(\mathbf{s}')]$$

$$\arg\max_{\mathbf{a}} Q(\mathbf{s},\mathbf{a}) \rightarrow \text{policy}$$

$$\downarrow$$

approximates the new value!



$$V^\pi(\mathbf{s}) \leftarrow \max_{\mathbf{a}} Q^\pi(\mathbf{s},\mathbf{a})$$

Slide adapted from Sergey Levine

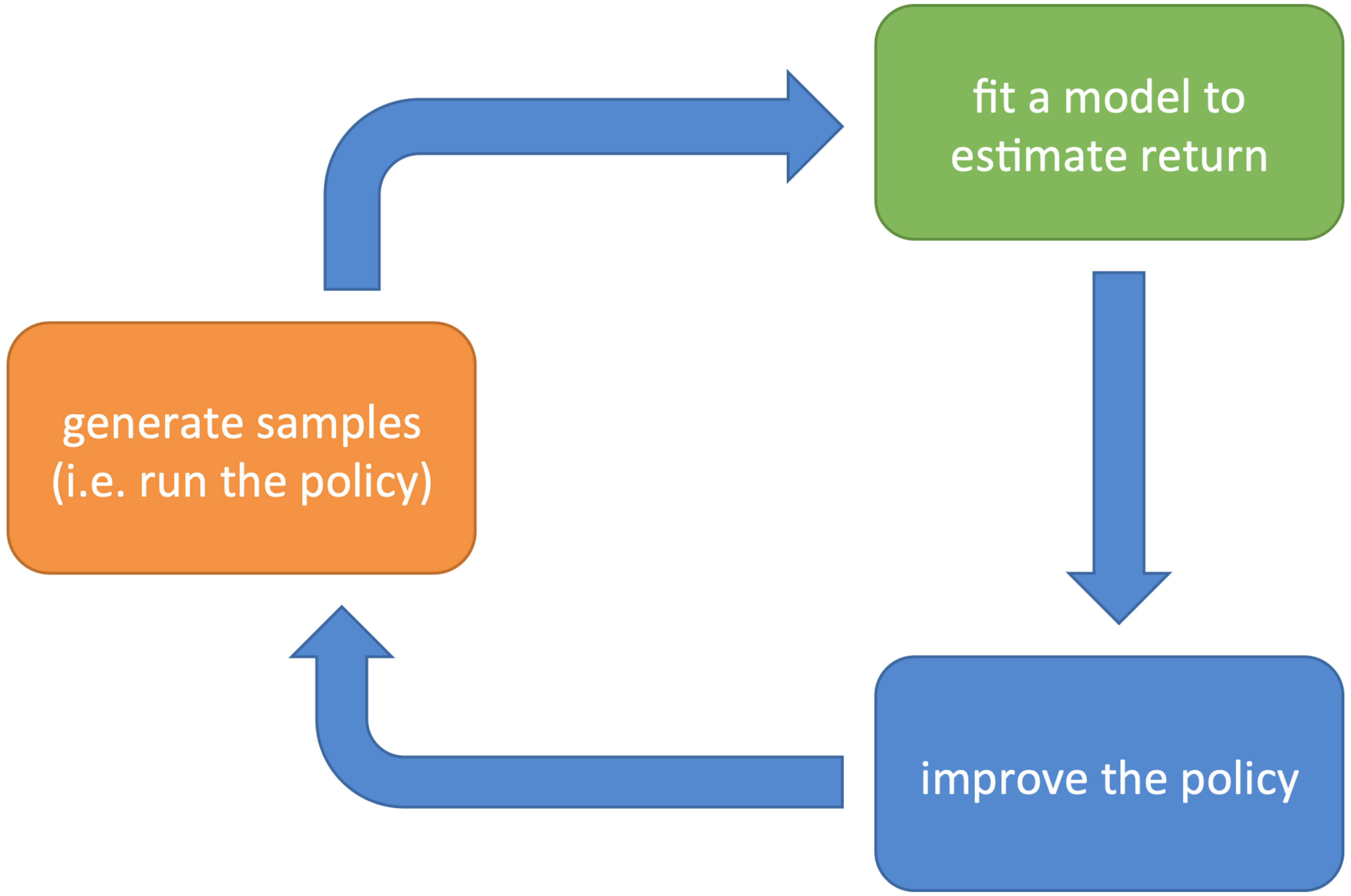
Q learning

$$\pi'(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q^\pi(\mathbf{s},\mathbf{a}) \\ 0 \text{ otherwise} \end{cases}$$

value iteration algorithm:

1. set $Q(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + \gamma E[V(\mathbf{s}')]$
2. set $V(\mathbf{s}) \leftarrow \max_{\mathbf{a}} Q(\mathbf{s},\mathbf{a})$

$$Q^\pi(\mathbf{s},\mathbf{a}) \leftarrow r(\mathbf{s},\mathbf{a}) + \gamma E_{\mathbf{s}'\sim p(\mathbf{s}'|\mathbf{s},\mathbf{a})}[V^\pi(\mathbf{s}')]$$



$$V^\pi(\mathbf{s}) \leftarrow \max_{\mathbf{a}} Q^\pi(\mathbf{s},\mathbf{a})$$

fitted Q iteration algorithm:

1. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i,\mathbf{a}_i) + \gamma E[V_\phi(\mathbf{s}'_i)]$ $\longleftarrow$ approxiate $E[V(\mathbf{s}'_i)] \approx \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i,\mathbf{a}'_i)$
2. set $\phi \leftarrow \arg\min_\phi \frac{1}{2}\sum_i \|Q_\phi(\mathbf{s}_i,\mathbf{a}_i) - \mathbf{y}_i\|^2$

doesn't require simulation of actions!

$$Q^*(\mathbf{s}_t,\mathbf{a}_t) = E_{\mathbf{s}_{t+1}\sim p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}\left[r(\mathbf{s}_t,\mathbf{a}_t) + \gamma \max_{\mathbf{a}'} Q^*(\mathbf{s}_{t+1},\mathbf{a}')\right]$$

Value-Based RL: Definitions

$Q^\pi(\mathbf{s}_t,\mathbf{a}_t) = \sum_{t'=t}^T E_{\pi_\theta}\left[r(\mathbf{s}_{t'},\mathbf{a}_{t'})|\mathbf{s}_t,\mathbf{a}_t\right]\text{: total reward from taking } \mathbf{a}_t \text{ in } \mathbf{s}_t$ "how good is a state-action pair"

$V^\pi(\mathbf{s}_t) = E_{\mathbf{a}_t\sim\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}[Q^\pi(\mathbf{s}_t,\mathbf{a}_t)]\text{: total reward from } \mathbf{s}_t$ "how good is a state"

If you know Q^π, you can use it to improve π.

$$\pi'(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t,\mathbf{a}_t) \\ 0 \text{ otherwise} \end{cases}$$

For the optimal policy $\pi^\star$: $Q^*(\mathbf{s}_t,\mathbf{a}_t) = E_{\mathbf{s}_{t+1}\sim p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}\left[r(\mathbf{s}_t,\mathbf{a}_t) + \gamma \max_{\mathbf{a}'} Q^*(\mathbf{s}_{t+1},\mathbf{a}')\right]$

Bellman equation

Value-Based RL

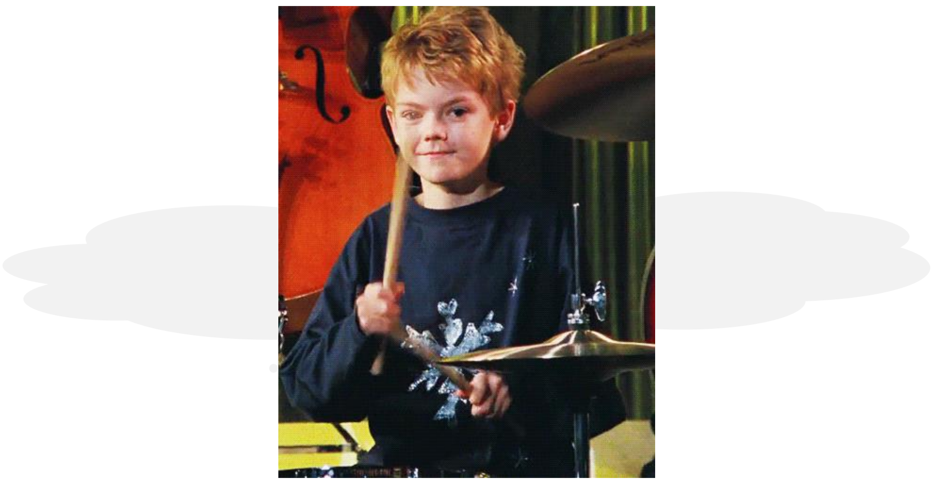
Value function: $V^{\pi}(\mathbf{s}_t) = ?$

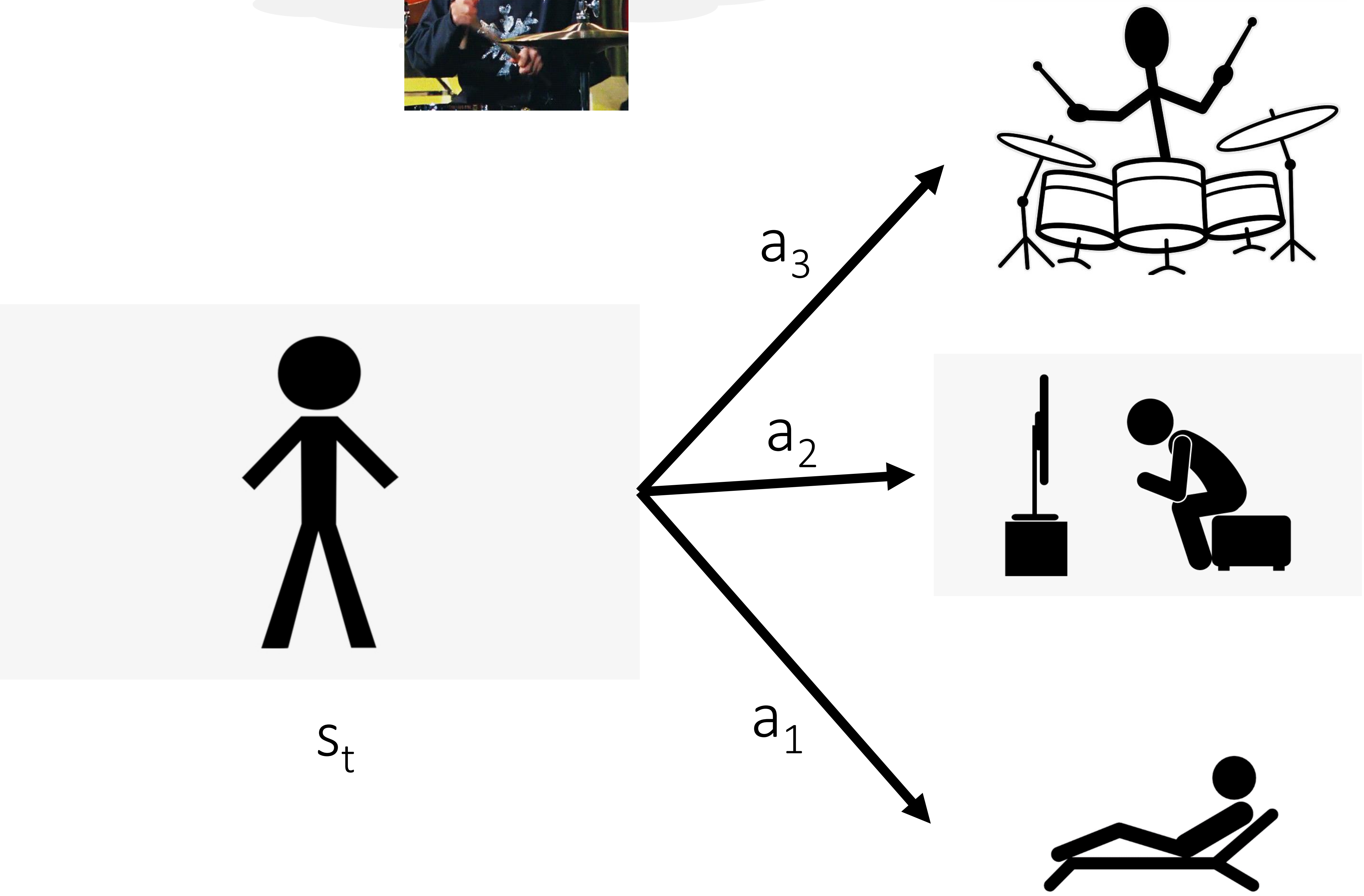
Q function: $Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = ?$

Q* function: $Q^{*}(\mathbf{s}_t, \mathbf{a}_t) = ?$

Value* function: $V^{*}(\mathbf{s}_t) = ?$

Reward = 1 if I can play it in a month, 0 otherwise





Current $\pi(\mathbf{a}_2|\mathbf{s}) = 1$

Fitted Q-iteration Algorithm

full fitted Q-iteration algorithm:

$K\times$

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy
2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

Algorithm hyperparameters

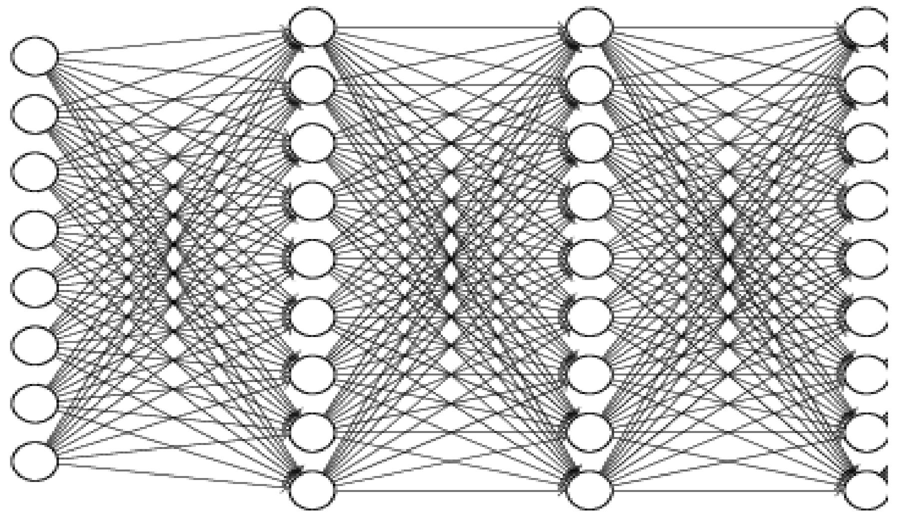
dataset size N, collection policy

iterations K

gradient steps S

$\mathbf{s}$

$\mathbf{a}$



$Q_\phi(\mathbf{s}, \mathbf{a})$

parameters ϕ

Result: get a policy $\pi(\mathbf{a}|\mathbf{s})$ from $\arg\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})$

Important notes:

We can reuse data from previous policies!

an off-policy algorithm   using replay buffers

Q learning animation

Q-learning

Bellman equation: $Q^*(\mathbf{s}_t, \mathbf{a}_t) = E_{\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)} \left[r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \max_{\mathbf{a}'} Q^*(\mathbf{s}_{t+1}, \mathbf{a}') \right]$

Pros:

+ More sample efficient than on-policy methods
+ Can incorporate off-policy data (including a fully offline setting)
+ Can updates the policy even without seeing the reward
+ Relatively easy to parallelize

Cons:

- Lots of "tricks" to make it work
- Potentially could be harder to learn than just a policy

The Plan

Recap

Q learning tricks

Improving Q learning

Case studies: games, robotics

Q-learning

fitted Q iteration algorithm:

1. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
2. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

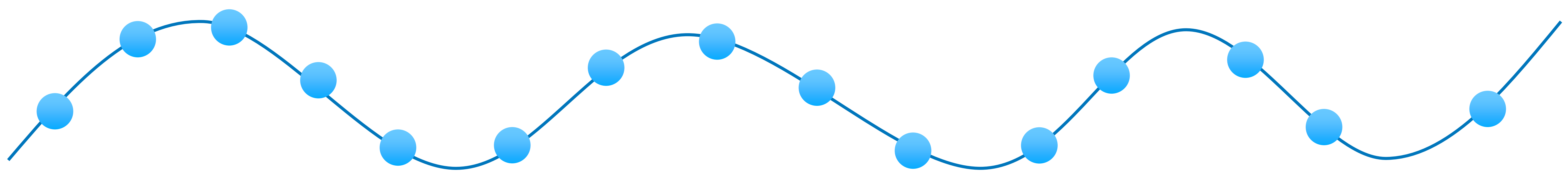
Questions:

- Is this a gradient descent algorithm?
- Is this algorithm off or on policy?
- What could be potential problems with it?

Correlated samples in online Q-learning

online Q iteration algorithm:

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$



- sequential states are strongly correlated
- target value is always changing

Solution: replay buffers

online Q iteration algorithm:

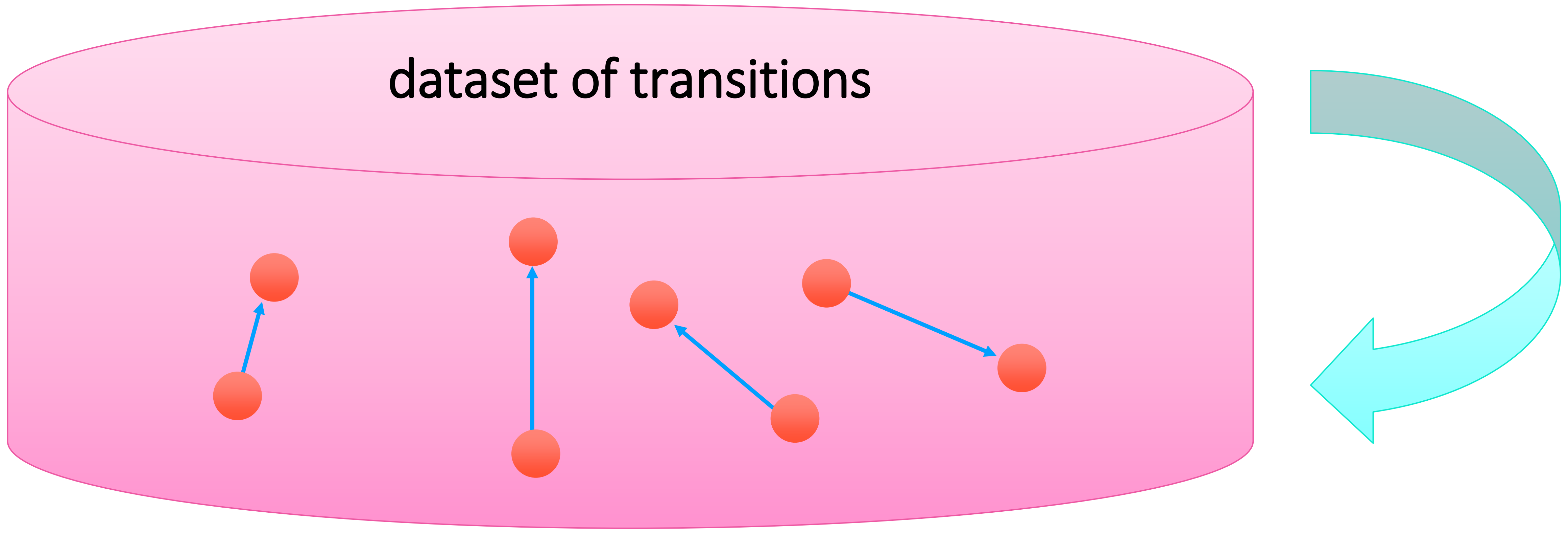
1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

full fitted Q-iteration algorithm:

~~1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy~~ any policy will work!

$K\times$

2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$ just load data from a buffer here
3. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$



Fitted Q-iteration

Solution: replay buffers

Q-learning with a replay buffer:

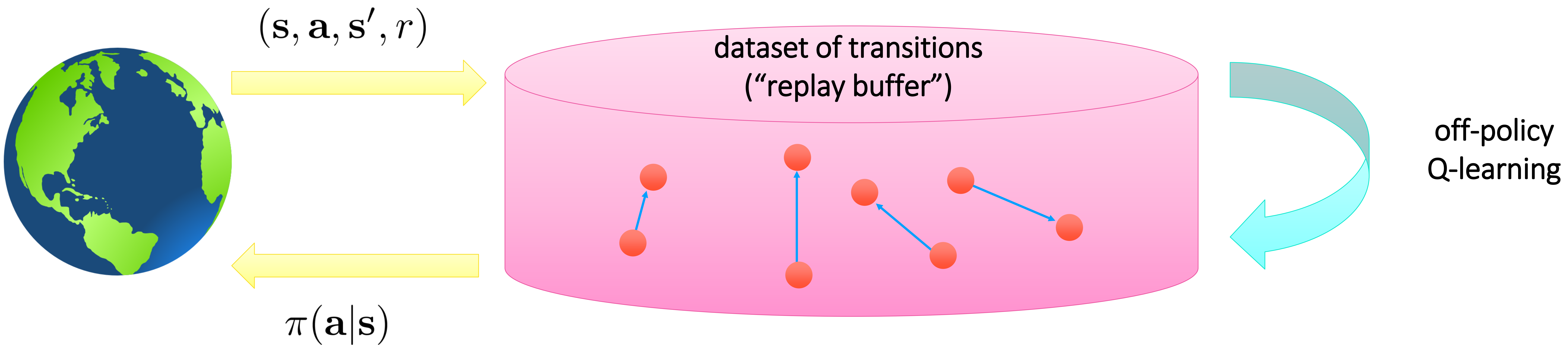
1. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
2. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

+ samples are no longer correlated

+ multiple samples in the batch (low-variance gradient)

but where does the data come from?

need to periodically feed the replay buffer…



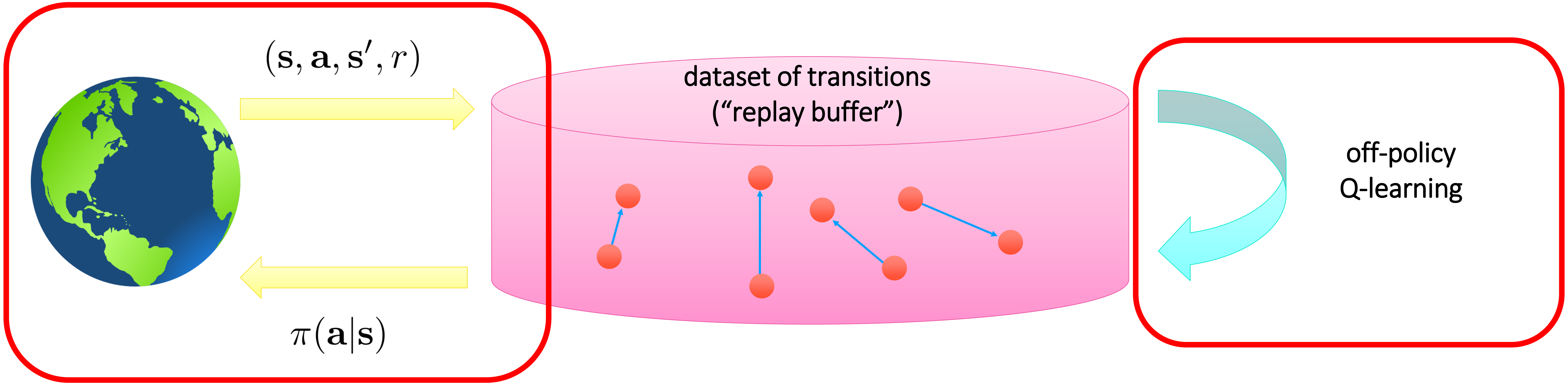
Putting it together

full Q-learning with replay buffer:

$K\times$

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{B}$
2. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
3. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

K = 1 is common, though larger K more efficient



Target Networks

What's wrong?

online Q iteration algorithm:

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

~~these are correlated!~~

use replay buffer

- sequential states are strongly correlated
- target value is always changing

Q-Learning and Regression

full fitted Q-iteration algorithm:

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy

$K\times$

2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$

Moving targets!

3. set $\underbrace{\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2}_{\text{perfectly well-defined, stable regression}}$

Q-Learning with target networks

Q-learning with replay buffer and target network:

1. save target network parameters: $\phi' \leftarrow \phi$

$N\times$

2. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{B}$

$K\times$

3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
4. set $\phi \leftarrow \arg\min_\phi \frac{1}{2}\sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \underbrace{[r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)]}_{\text{targets don't change in inner loop!}}\|^2$

supervised regression

"Classic" deep Q-learning algorithm (DQN)

"classic" deep Q-learning algorithm:

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target* network $Q_{\phi'}$
4. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_j ||Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j||^2$
5. update ϕ': copy ϕ every N steps

$K = 1$

Mnih et al. '13

The Plan

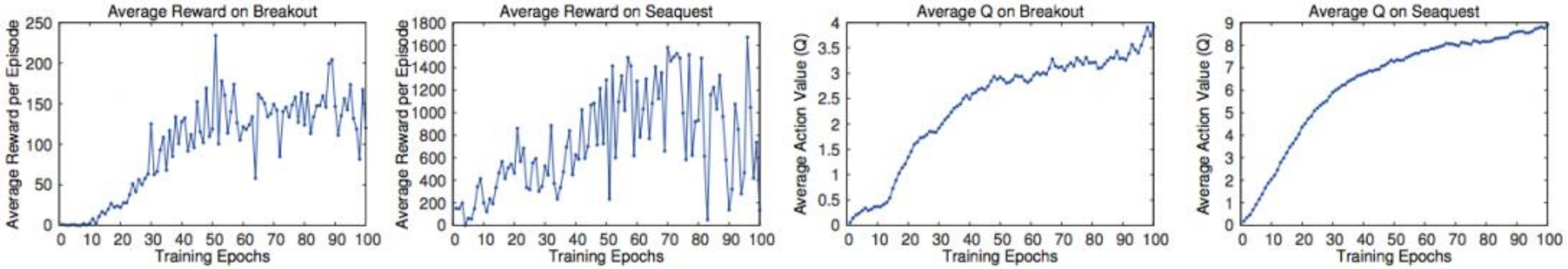
Recap

Q learning tricks

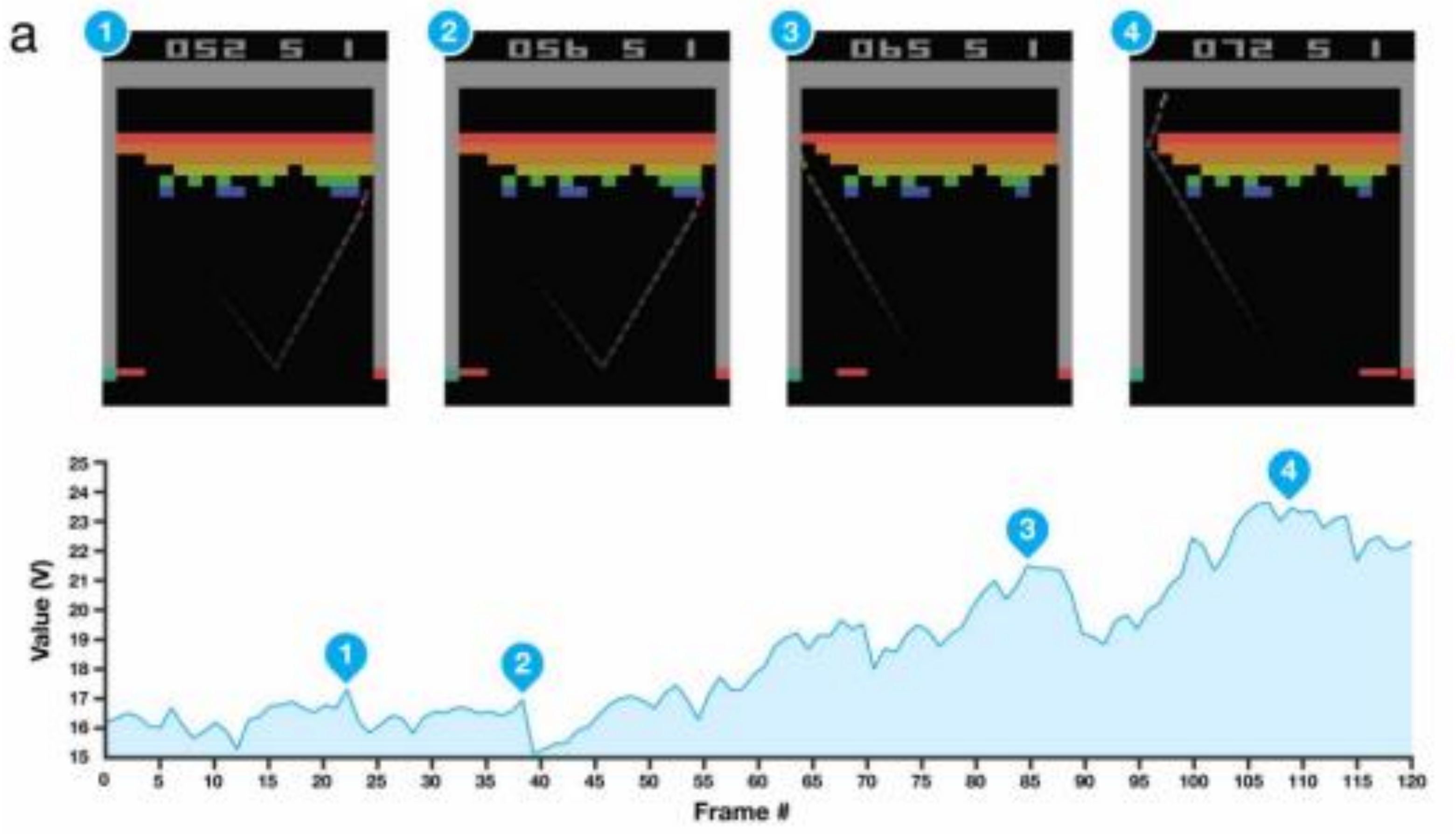
Improving Q learning

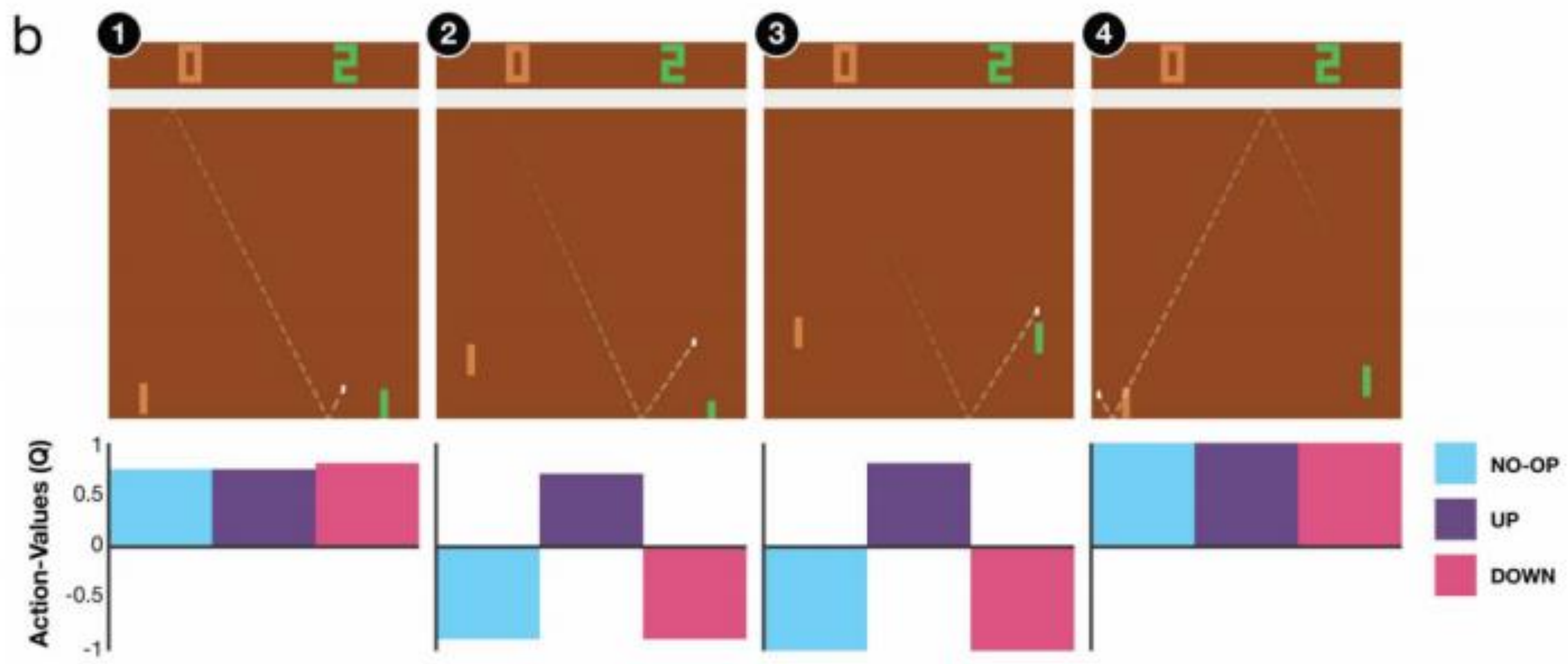
Case studies: games, robotics

Are the Q-values accurate?

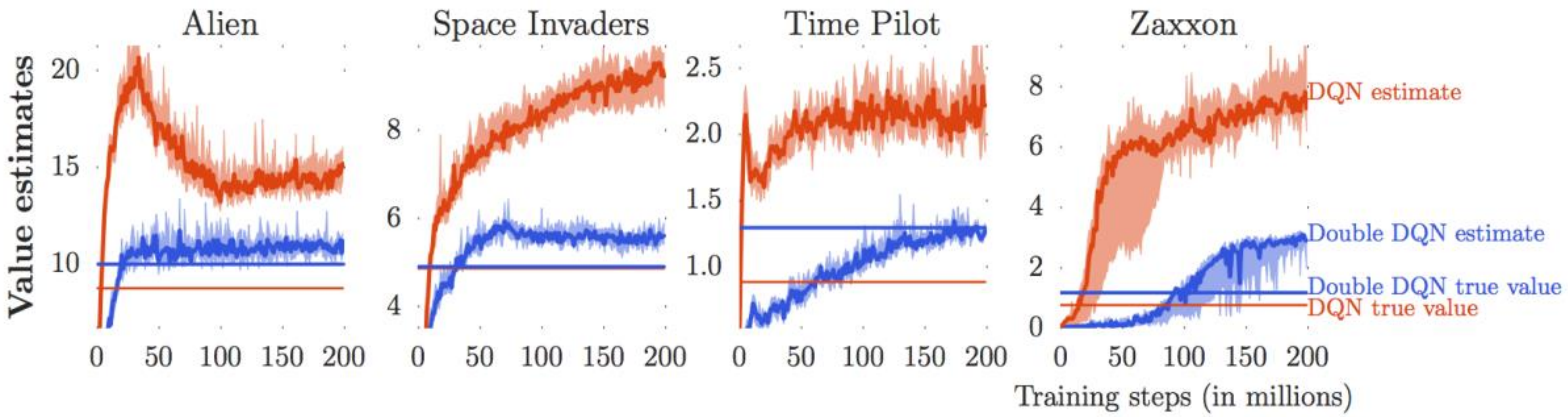


As predicted Q increases, so does the return





Are the Q-values accurate?



Overestimation in Q-learning

target value $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$

this last term is the problem

imagine we have two random variables: X_1 and X_2

$E[\max(X_1, X_2)] \geq \max(E[X_1], E[X_2])$

$Q_{\phi'}(\mathbf{s}', \mathbf{a}')$ is not perfect – it looks "noisy"

hence $\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}')$ *overestimates* the next value!

note that $\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}') = \underbrace{Q_{\phi'}}_{\text{value } also \text{ comes from } Q_{\phi'}}(\mathbf{s}', \underbrace{\arg\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}')}_{\text{action selected according to } Q_{\phi'}})$

Double Q-learning

$E[\max(X_1, X_2)] \geq \max(E[X_1], E[X_2])$

note that $\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}') = \underbrace{Q_{\phi'}}_{\text{value } also \text{ comes from } Q_{\phi'}}(\mathbf{s}', \underbrace{\arg\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}')}_{\text{action selected according to } Q_{\phi'}})$

if the noise in these is decorrelated, the problem goes away!

idea: don't use the same network to choose the action and evaluate value!

"double" Q-learning: use two networks:

$$Q_{\phi_A}(\mathbf{s}, \mathbf{a}) \leftarrow r + \gamma Q_{\phi_B}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi_A}(\mathbf{s}', \mathbf{a}'))$$

$$Q_{\phi_B}(\mathbf{s}, \mathbf{a}) \leftarrow r + \gamma Q_{\phi_A}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi_B}(\mathbf{s}', \mathbf{a}'))$$

if the two Q's are noisy in *different* ways, there is no problem

Double Q-learning in practice

where to get two Q-functions?

just use the current and target networks!

standard Q-learning: $y = r + \gamma Q_{\phi'}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}'))$

double Q-learning: $y = r + \gamma Q_{\phi'}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi}(\mathbf{s}', \mathbf{a}'))$

just use current network (not target network) to evaluate action

still use target network to evaluate value!

Multi-step returns

Q-learning target: $y_{j,t} = r_{j,t} + \gamma \max_{\mathbf{a}_{j,t+1}} Q_{\phi'}(\mathbf{s}_{j,t+1}, \mathbf{a}_{j,t+1})$

these are the only values that matter if $Q_{\phi'}$ is bad!

these values are important if $Q_{\phi'}$ is good

where does the signal come from?

Q-learning does this: max bias, min variance

remember this?

Actor-critic: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\left(r(\mathbf{s}_{i,t},\mathbf{a}_{i,t}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1}) - \hat{V}_\phi^\pi(\mathbf{s}_{i,t})\right)$

+ lower variance (due to critic)
- not unbiased (if the critic is not perfect)

Policy gradient: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\left(\left(\sum_{t'=t}^{T}\gamma^{t'-t} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\right) - b\right)$

+ no bias
- higher variance (because single-sample estimate)

can we construct multi-step targets, like in actor-critic?

$y_{j,t} = \sum_{t'=t}^{t+N-1} \gamma^{t-t'} r_{j,t'} + \gamma^N \max_{\mathbf{a}_{j,t+N}} Q_{\phi'}(\mathbf{s}_{j,t+N}, \mathbf{a}_{j,t+N})$

N-step return estimator

- Does it still work off-policy?

Q-learning with N-step returns

$$y_{j,t} = \sum_{t'=t}^{t+N-1} \gamma^{t-t'} r_{j,t'} + \gamma^N \max_{\mathbf{a}_{j,t+N}} Q_{\phi'}(\mathbf{s}_{j,t+N}, \mathbf{a}_{j,t+N})$$

this is supposed to estimate $Q^\pi(\mathbf{s}_{j,t}, \mathbf{a}_{j,t})$ for π

+ less biased target values when Q-values are inaccurate

+ typically faster learning, especially early on

- only actually correct when learning on-policy

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 \text{ otherwise} \end{cases}$$

why?

we need transitions $\mathbf{s}_{j,t'}, \mathbf{a}_{j,t'}, \mathbf{s}_{j,t'+1}$ to come from π for $t' - t < N - 1$

(not an issue when $N = 1$)

how to fix?

- ignore the problem
  - often works very well
- cut the trace – dynamically choose N to get only on-policy data
  - works well when data mostly on-policy, and action space is small
- importance sampling

For more details, see: “Safe and efficient off-policy reinforcement learning.” Munos et al. ‘16

Aside: exploration with Q-learning

online Q iteration algorithm

1. take some action $\mathbf{a}_i$ and observe: $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \| Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i \|^2$

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 \text{ otherwise} \end{cases}$$

- Why could that be a bad idea?

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 - \epsilon & \text{if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q^\phi(\mathbf{s}_t, \mathbf{a}_t) \\ \epsilon/(|\mathcal{A}| - 1) & \text{otherwise} \end{cases}$$

- Epsilon greedy

- Why could that be a bad idea?

Simple practical tips for Q-learning

- Q-learning takes some care to stabilize
  - Test on easy, reliable tasks first, make sure your implementation is correct

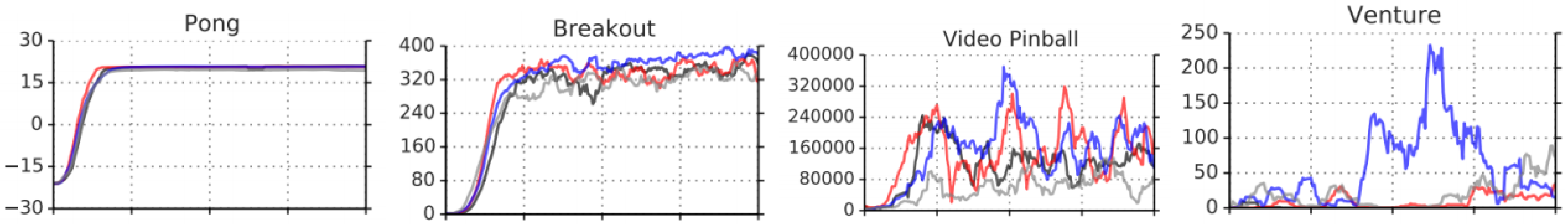


Figure: From T. Schaul, J. Quan, I. Antonoglou, and D. Silver. "Prioritized experience replay". *arXiv preprint arXiv:1511.05952* (2015), Figure 7

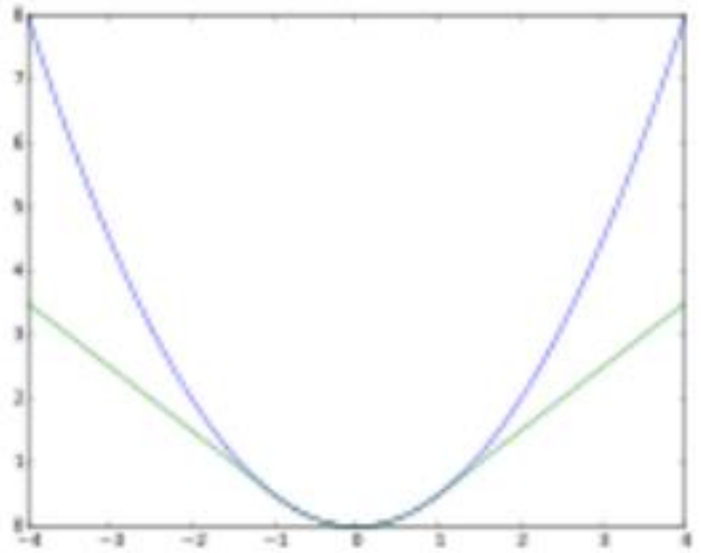
- Large replay buffers help improve stability
  - Looks more like fitted Q-iteration
- It takes time, be patient – might be no better than random for a while
- Start with high exploration (epsilon) and gradually reduce

Slide partly borrowed from J. Schulman

Advanced tips for Q-learning

- Bellman error gradients can be big; clip gradients or use Huber loss

$$L(x) = \begin{cases} x^2/2 & \text{if } |x| \le \delta \\ \delta|x| - \delta^2/2 & \text{otherwise} \end{cases}$$



- Double Q-learning helps *a lot* in practice, simple and no downsides
- N-step returns also help a lot, but have some downsides
- Schedule exploration (high to low) and learning rates (high to low), Adam optimizer can help too
- Run multiple random seeds, it's very inconsistent between runs

Slide partly borrowed from J. Schulman

The Plan

Recap

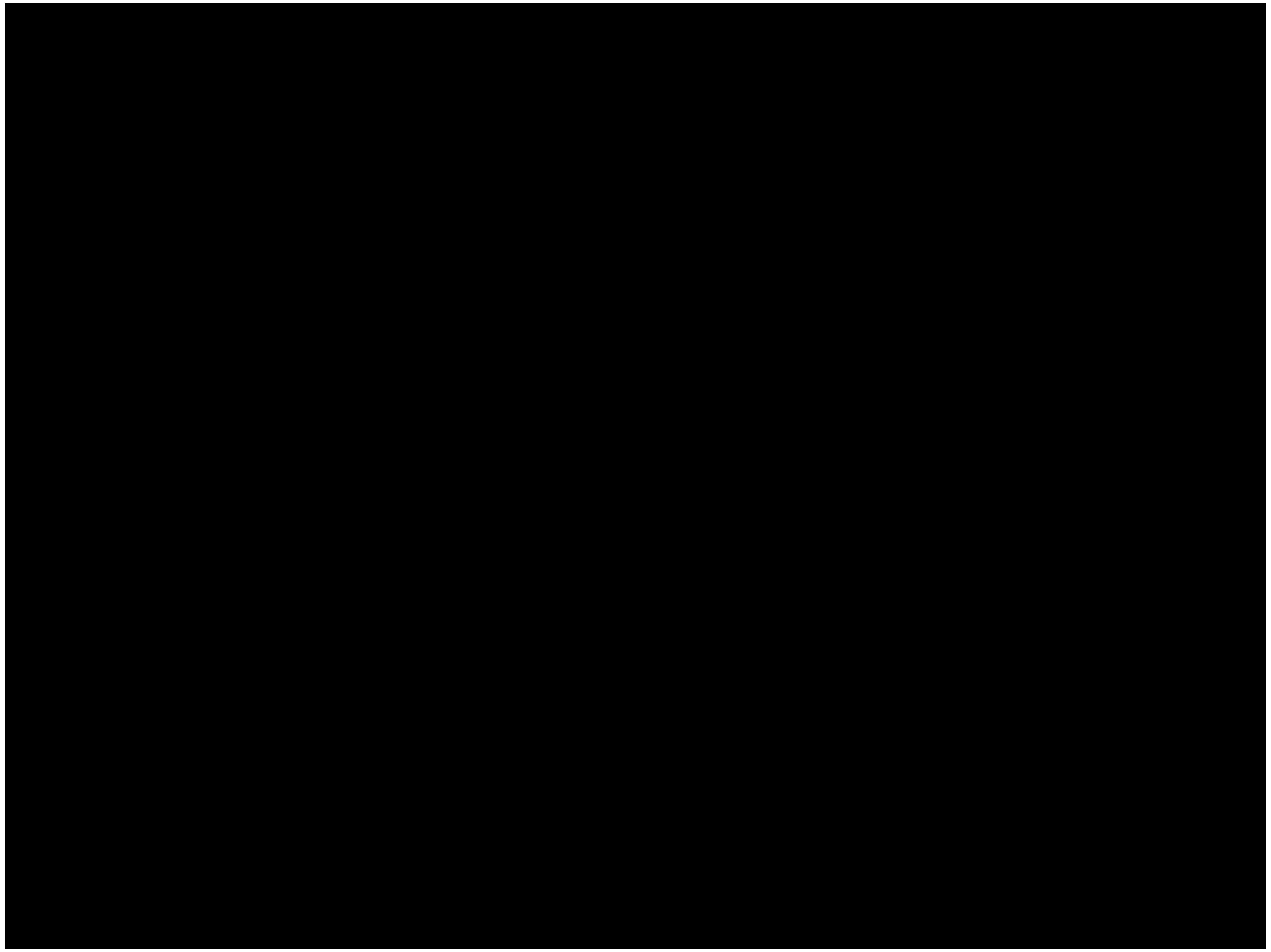
Q learning tricks

Improving Q learning

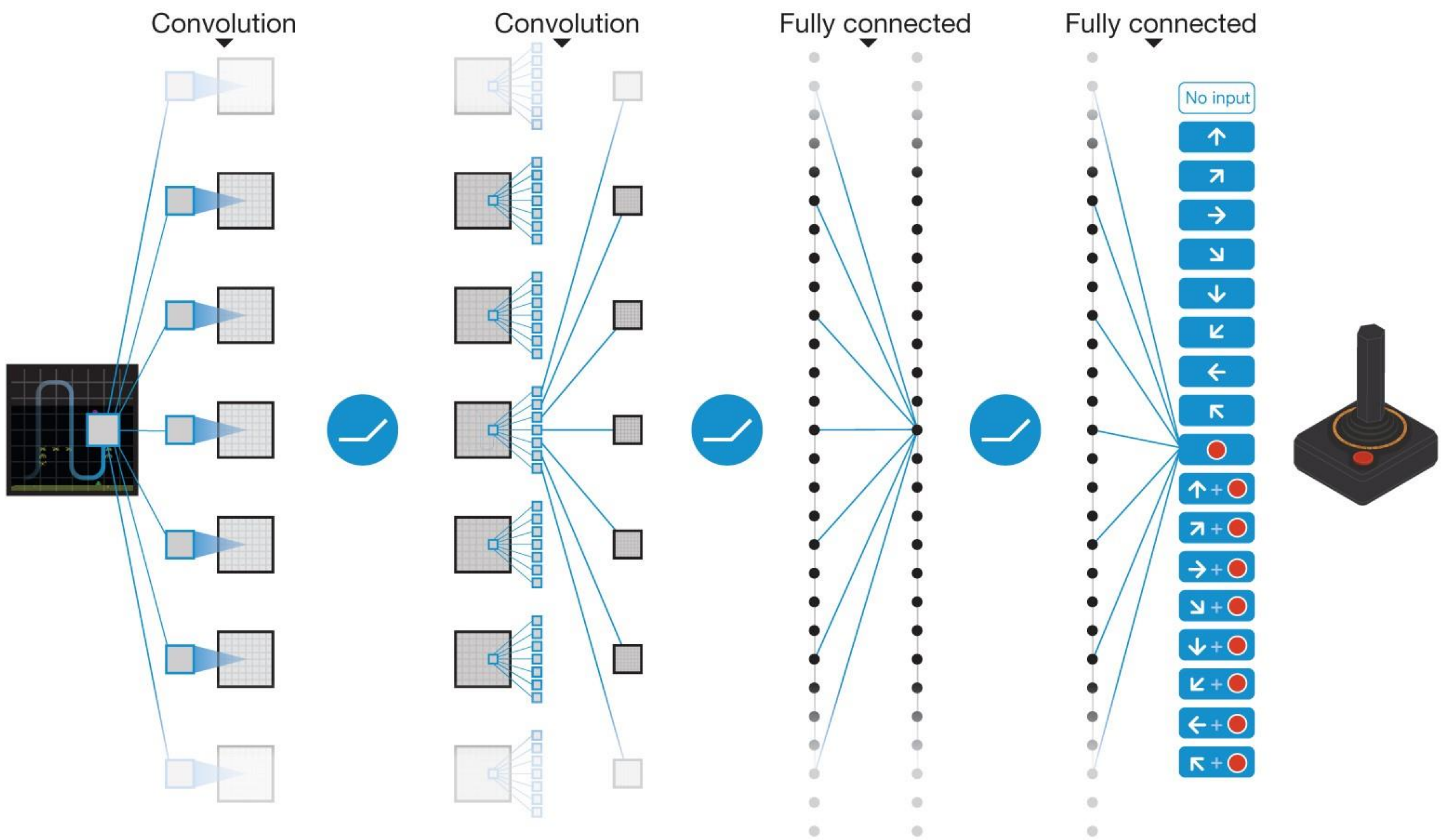
Case studies: games, robotics

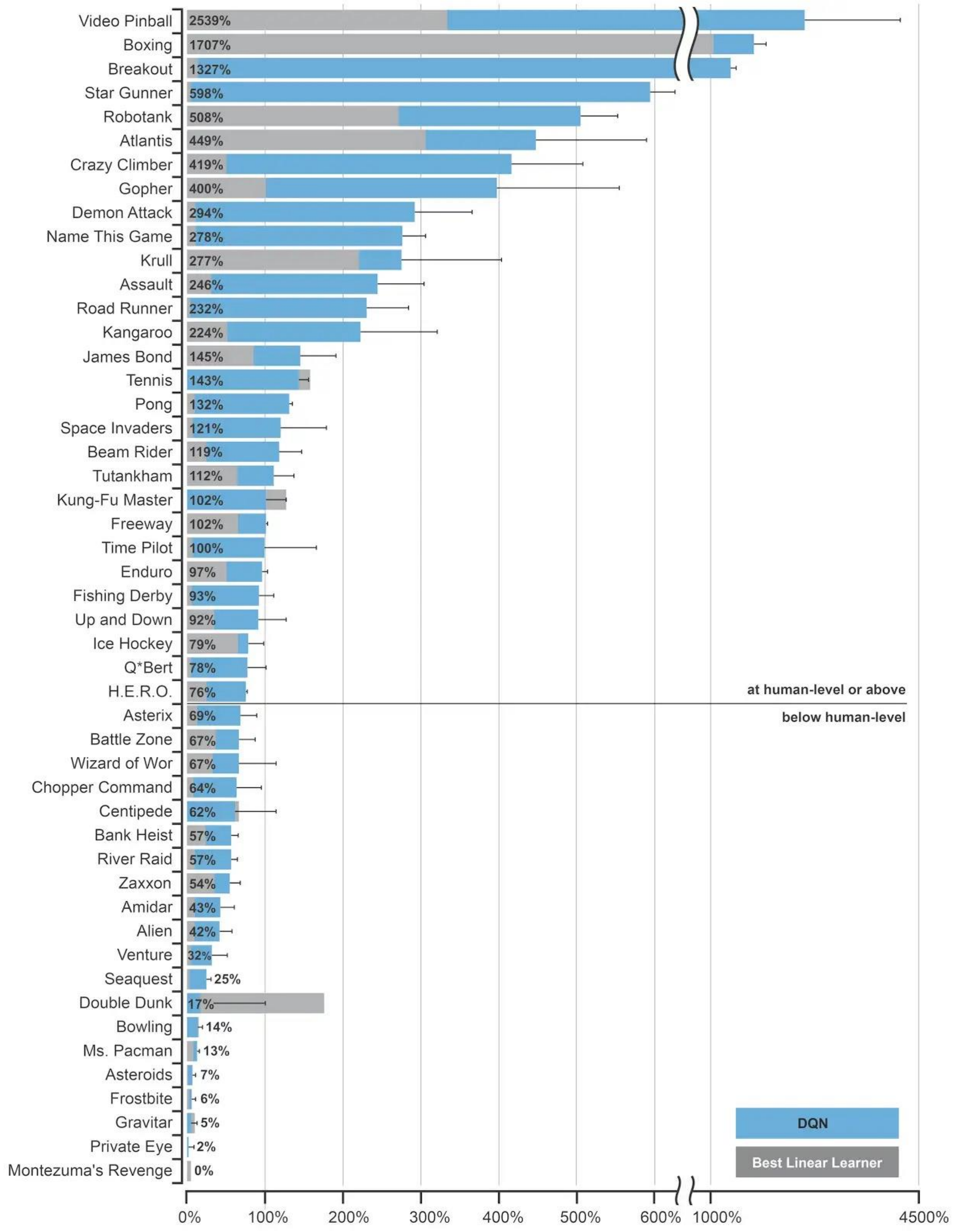
Example: Deep Q Learning (DQN) Applied to Atari

- Human-level control through deep RL, Mnih et. al, 2013
- Uses target network and replay buffer
- One step back-up (no n-step returns)
- Became a popular benchmark since



Example: DQN


Convolution
Convolution
Fully connected
Fully connected
No input

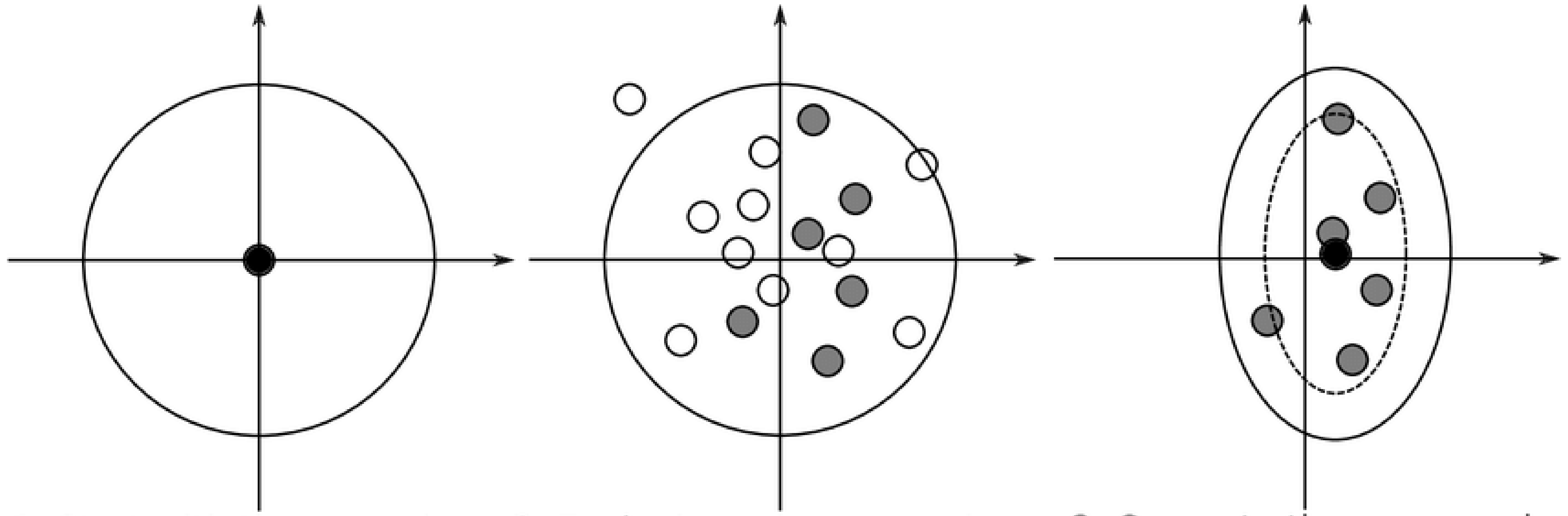

Video Pinball 2539%
Boxing 1707%
Breakout 1327%
Star Gunner 598%
Robotank 508%
Atlantis 449%
Crazy Climber 419%
Gopher 400%
Demon Attack 294%
Name This Game 278%
Krull 277%
Assault 246%
Road Runner 232%
Kangaroo 224%
James Bond 145%
Tennis 143%
Pong 132%
Space Invaders 121%
Beam Rider 119%
Tutankham 112%
Kung-Fu Master 102%
Freeway 102%
Time Pilot 100%
Enduro 97%
Fishing Derby 93%
Up and Down 92%
Ice Hockey 79%
Q*Bert 78%
H.E.R.O. 76%
at human-level or above
below human-level
Asterix 69%
Battle Zone 67%
Wizard of Wor 67%
Chopper Command 64%
Centipede 62%
Bank Heist 57%
River Raid 57%
Zaxxon 54%
Amidar 43%
Alien 42%
Venture 32%
Seaquest 25%
Double Dunk 17%
Bowling 14%
Ms. Pacman 13%
Asteroids 7%
Frostbite 6%
Gravitar 5%
Private Eye 2%
Montezuma's Revenge 0%
DQN
Best Linear Learner
0%
100%
200%
300%
400%
500%
600%
1000%
4500%

Example: Q-learning Applied to Robotics

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy
2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

Continuous action space?

Simple optimization algorithm -> Cross Entropy Method (CEM)

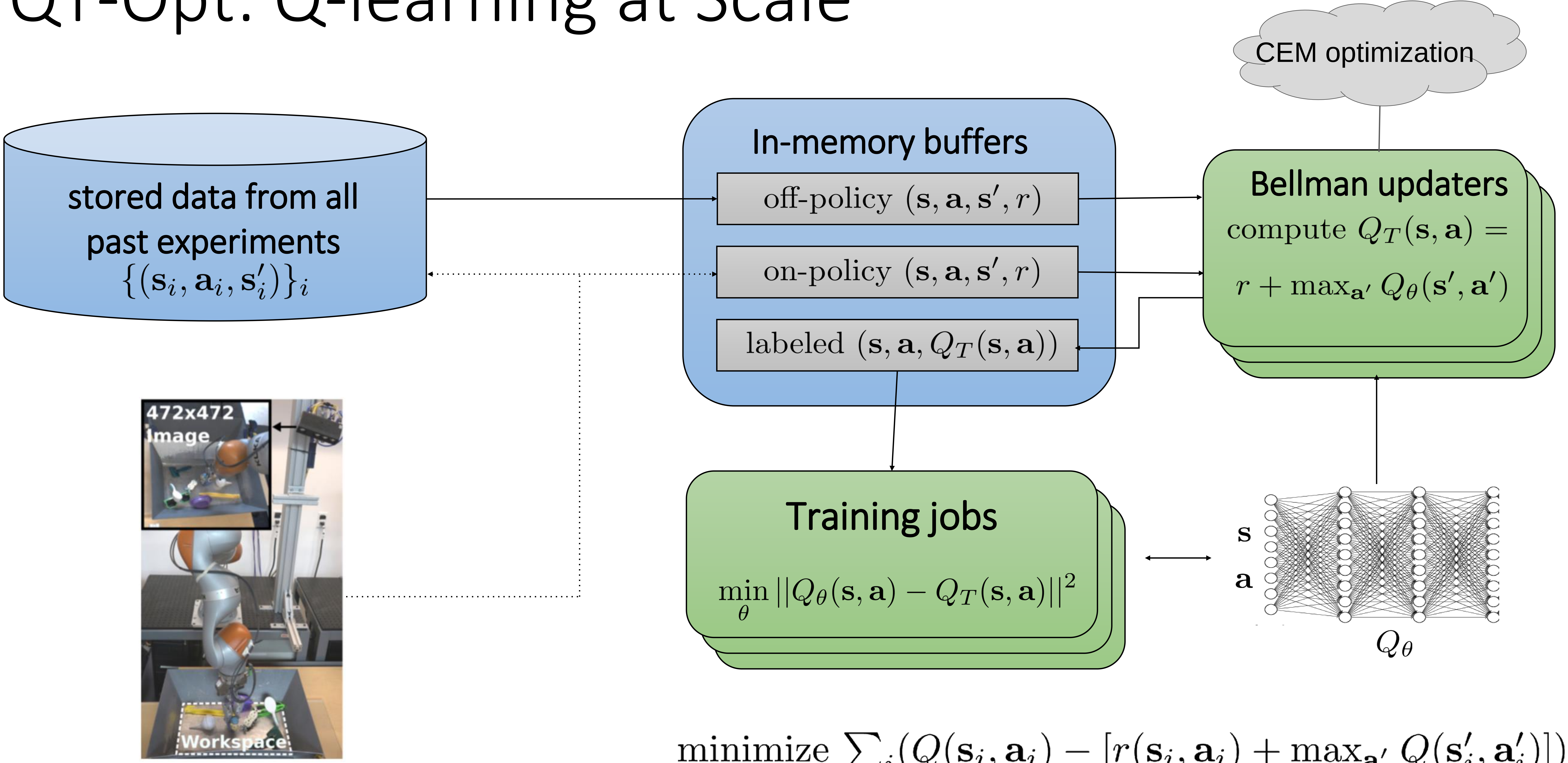


1. Start with the normal distribution N(μ,σ²).

2. Evaluate some parameters from this distribution and select the best (in grey)

3. Compute the mean and std.dev. of the best, add some noise and goto to 1

QT-Opt: Q-learning at Scale

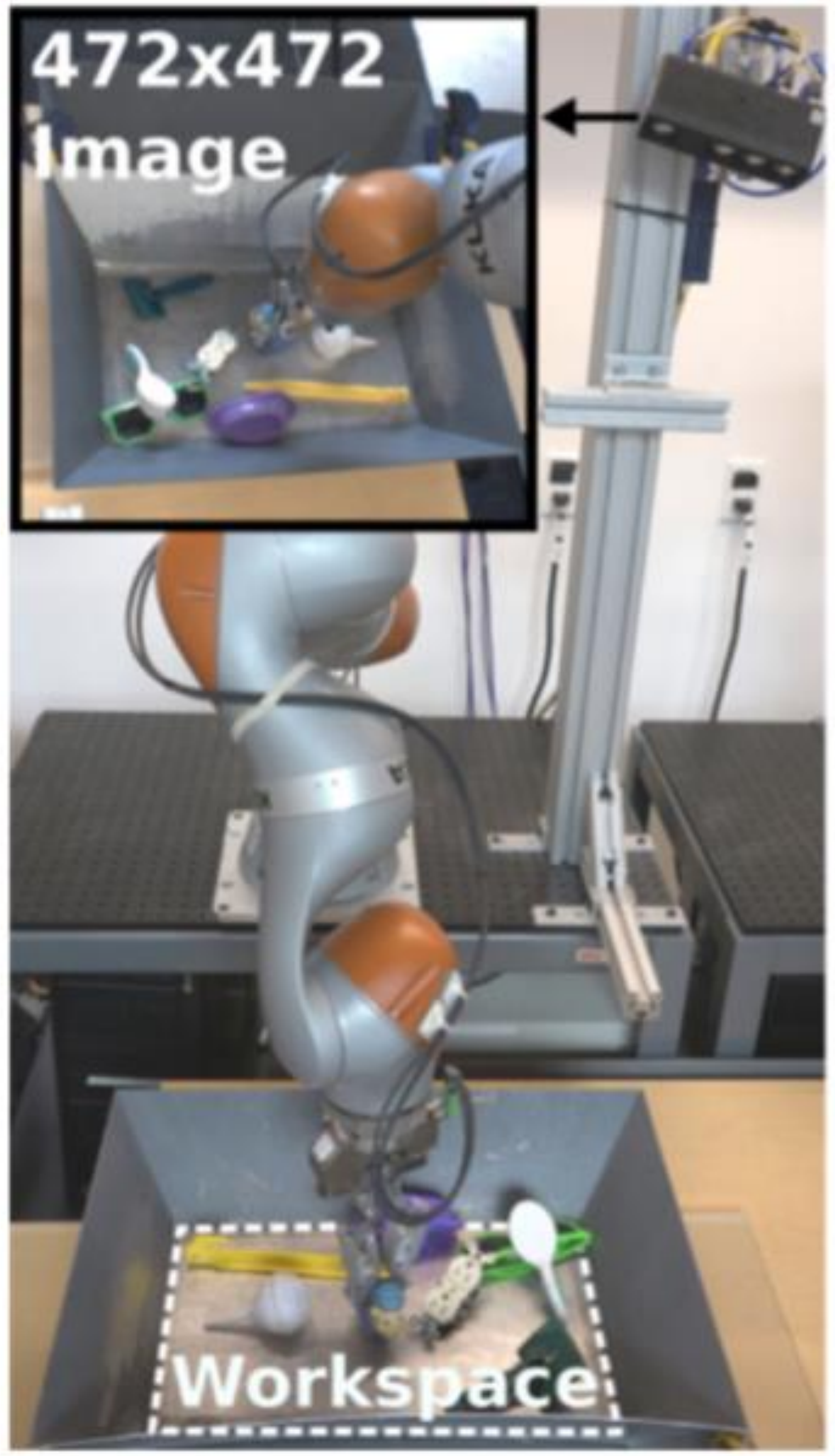


$$\text{minimize} \sum_i (Q(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \max_{\mathbf{a}'_i} Q(\mathbf{s}'_i, \mathbf{a}'_i)])^2$$

Slide adapted from D. Kalashnikov

QT-Opt: Kalashnikov et al. '18, Google Brain

QT-Opt: MDP Definition for Grasping

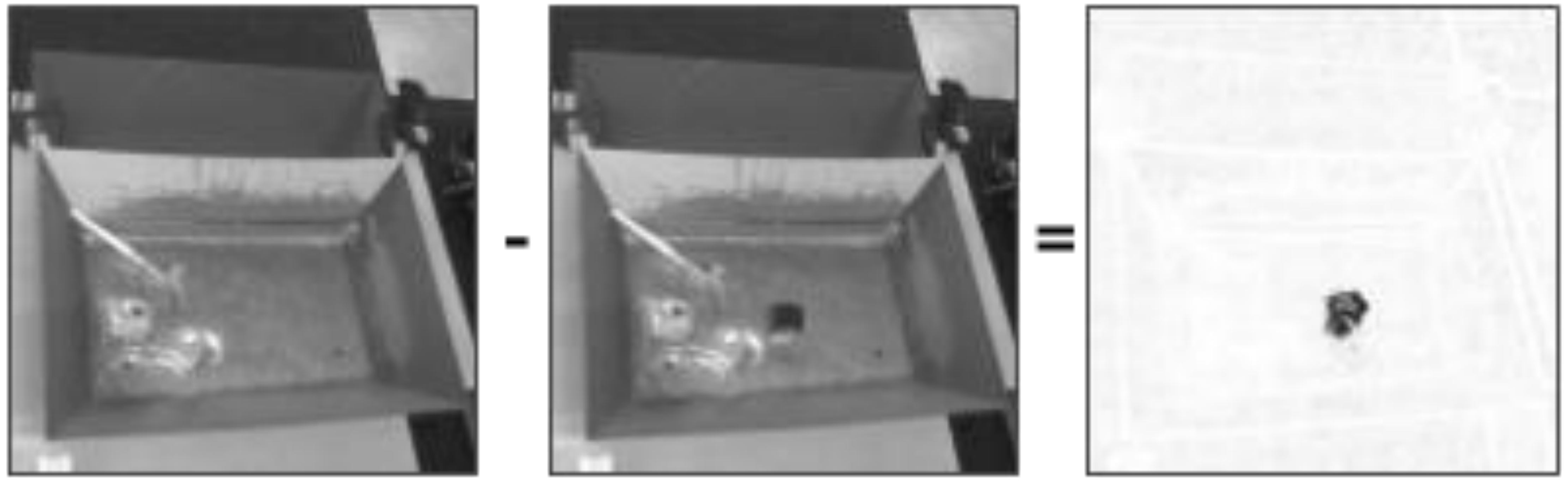


State: over the shoulder RGB camera image, no depth

Action: 4DOF pose change in Cartesian space + gripper control

Reward: binary reward at the end, if the object was lifted. Sparse. No shaping

Automatic success detection:



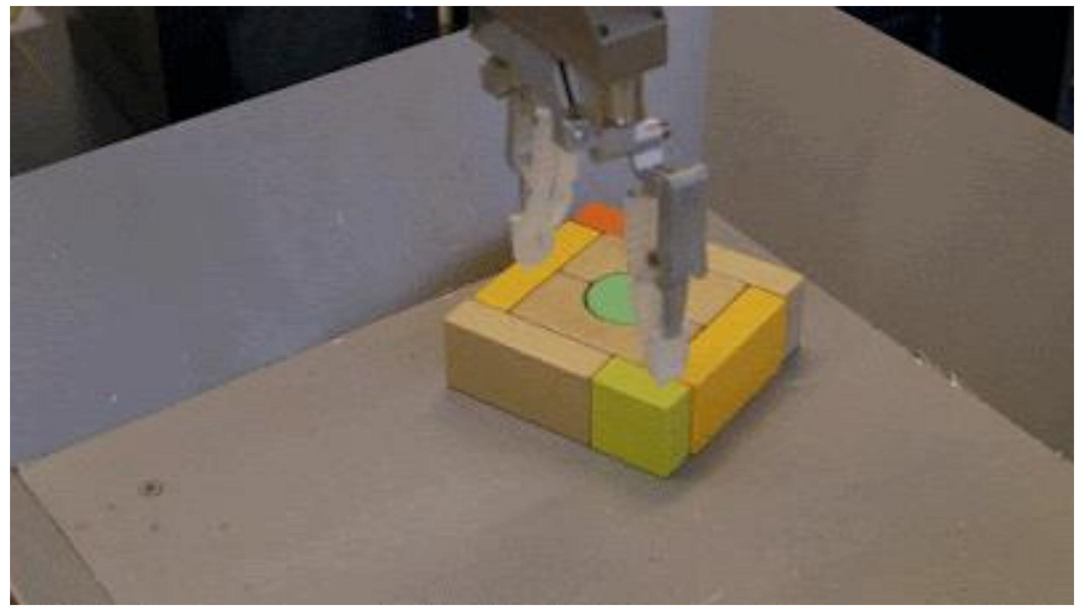
Slide adapted from D. Kalashnikov

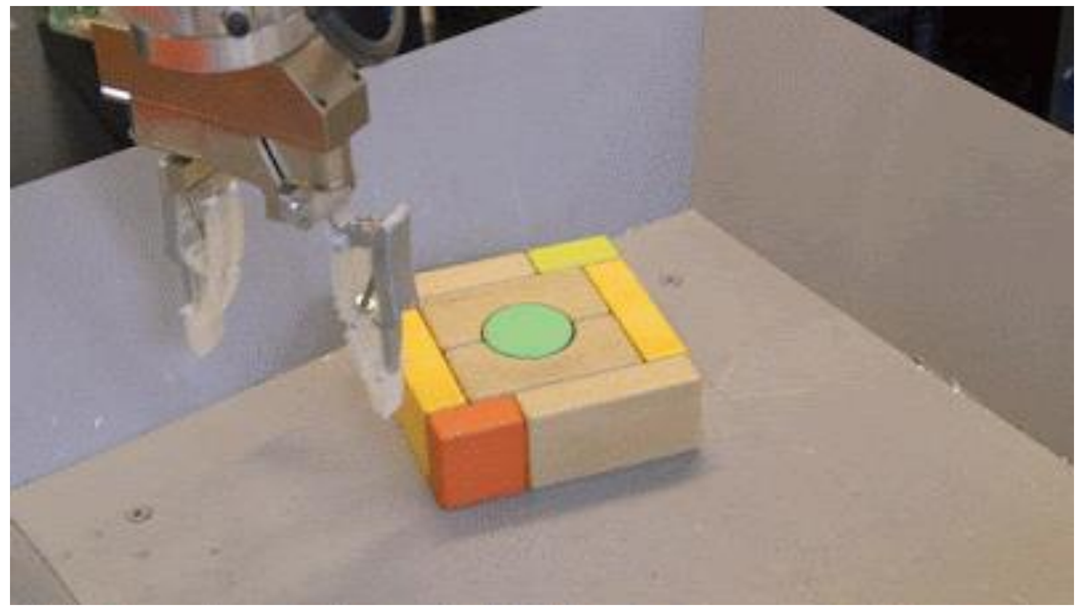
QT-Opt: Setup and Results

7 robots collected 580k grasps

Unseen test objects

96% test success rate!





Recap

Key learning goals:

- Practical Q learning implementation tricks
- Understanding the landscape of Q learning algorithms

Q learning implementation:

- Replay buffer & target networks
- Double Q-learning & n-step returns

Landscape of Q learning:

- Q learning w/ continuous actions
- Examples

Next

Any other way to learn a policy?

What about the dynamics of the environment?

Model-based RL