

# Deep RL with Q-Functions

CS 285: Deep Reinforcement Learning, Decision Making, and Control

Sergey Levine

# Class Notes

1. Homework 2 is due next Monday
2. Project proposal due 9/25, that's today!
  - Remember to upload to **both** Gradescope and CMT (see Piazza post)

# Today's Lecture

1. How we can make Q-learning work with deep networks
  2. A generalized view of Q-learning algorithms
  3. Tricks for improving Q-learning in practice
  4. Continuous Q-learning methods
- Goals:
    - Understand how to implement Q-learning so that it can be used with complex function approximators
    - Understand how to extend Q-learning to continuous actions

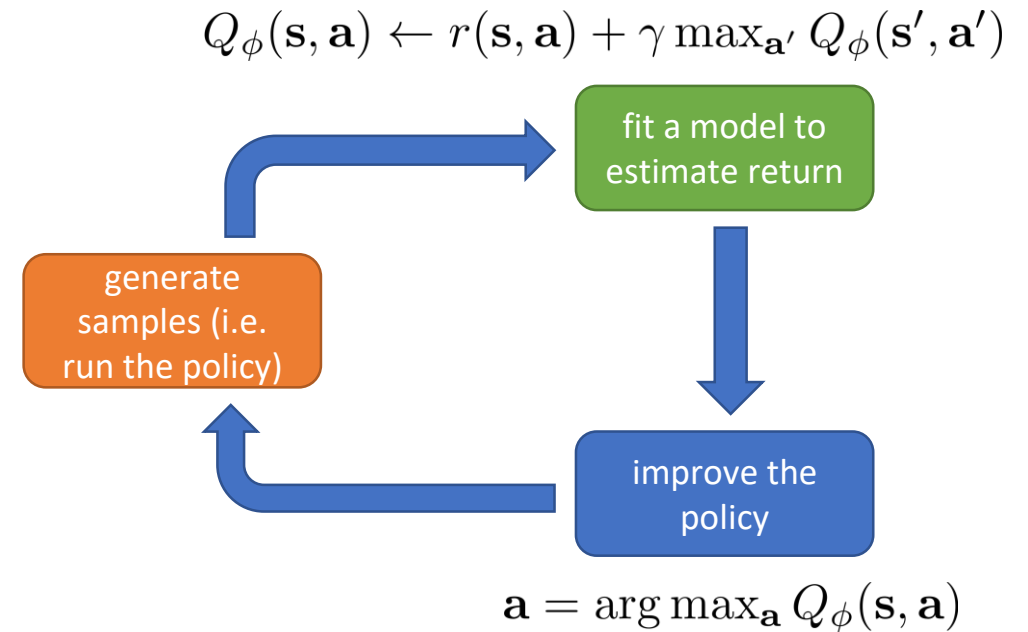
# Recap: Q-learning

full fitted Q-iteration algorithm:

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

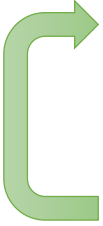
online Q iteration algorithm:

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.  $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$



# 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.  $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$
- these are correlated!
- isn't this just gradient descent? that converges, right?


Q-learning is *not* gradient descent!

$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - (r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)))$$

no gradient through target value

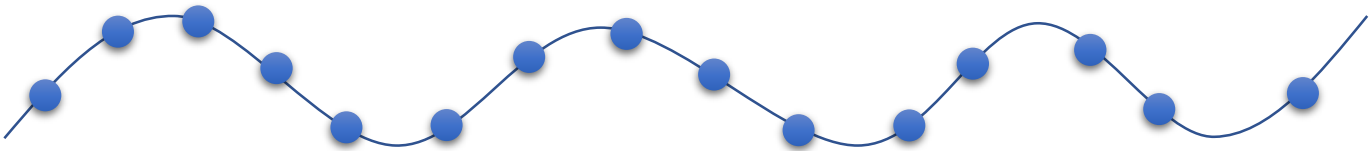
# Correlated samples in online Q-learning

online Q iteration algorithm:

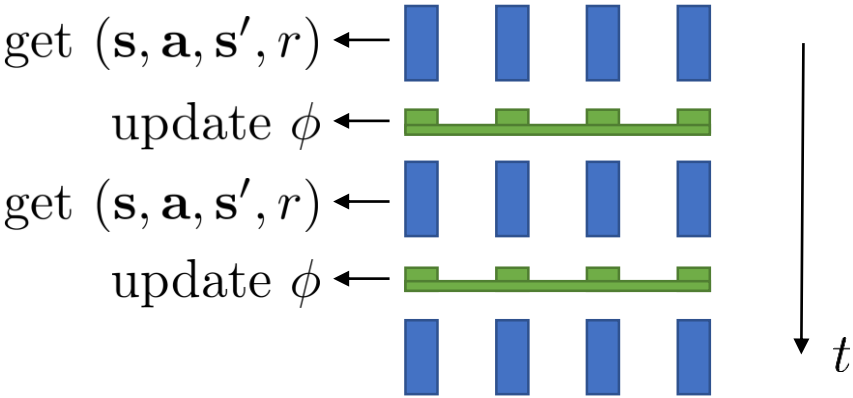
- 
1. take some action  $\mathbf{a}_i$  and observe  $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$

2.  $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$

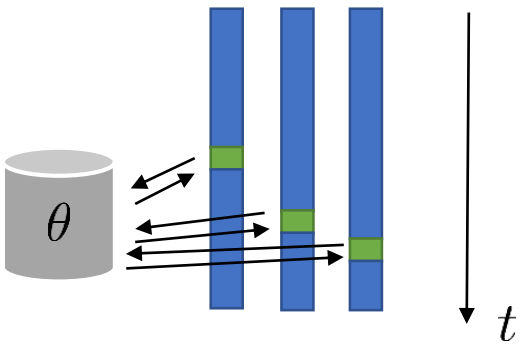
- sequential states are strongly correlated
- target value is always changing



synchronized parallel Q-learning




asynchronous parallel Q-learning




# Another solution: replay buffers

online Q iteration algorithm:

**special case with  $K = 1$ , and one gradient step**

- 
1. take some action  $\mathbf{a}_i$  and observe  $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
  2.  $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}')])$

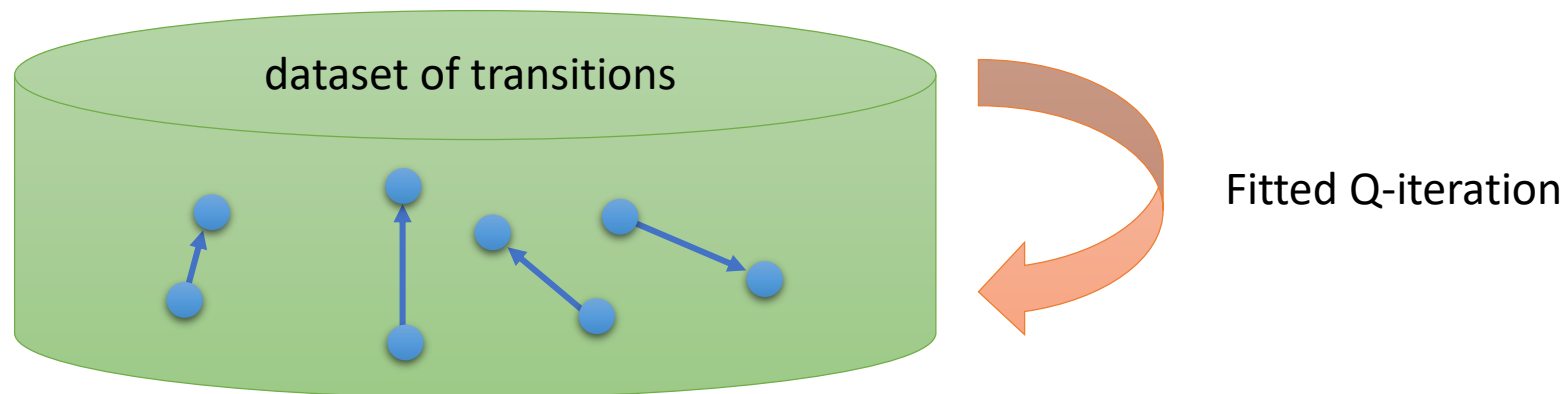
full fitted Q-iteration algorithm:

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

**any policy will work! (with broad support)**

**just load data from a buffer here**

**still use one gradient step**



# Another solution: replay buffers

Q-learning with a replay buffer:

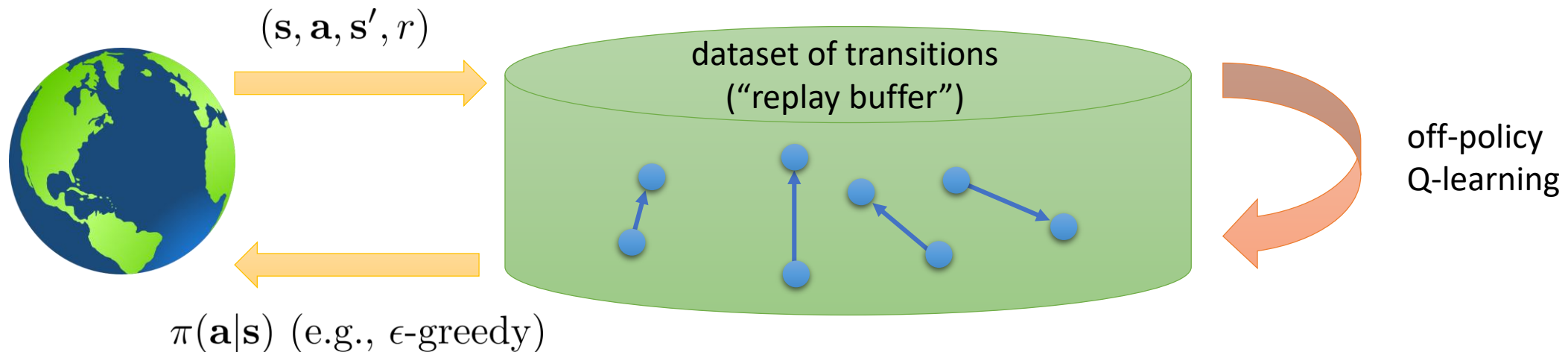
1. sample a batch  $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$  from  $\mathcal{B}$
2.  $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i) (Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$

+ 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...



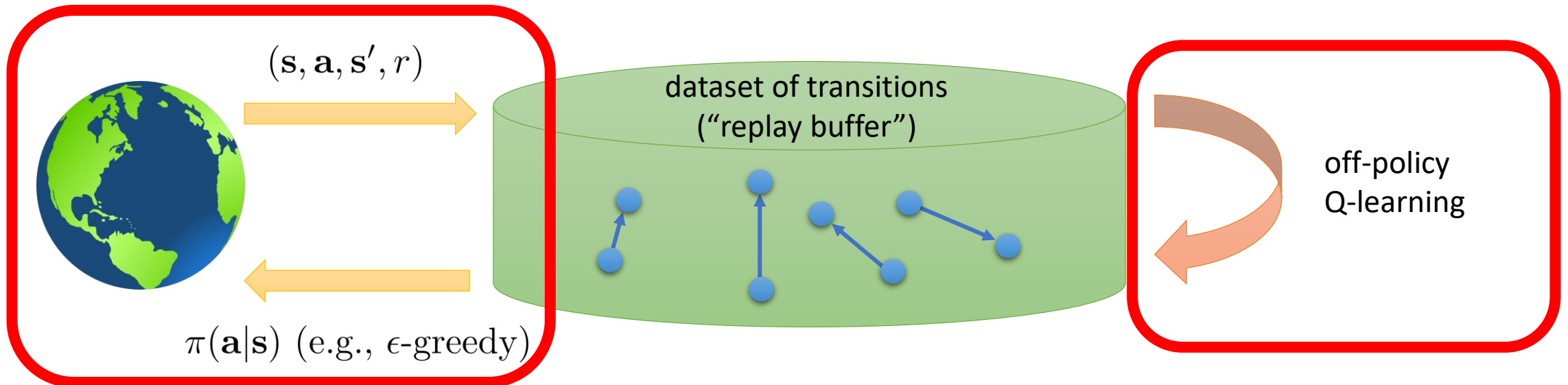


# Putting it together

full Q-learning with replay buffer:

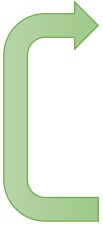
1. collect dataset  $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$  using some policy, add it to  $\mathcal{B}$
2. sample a batch  $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$  from  $\mathcal{B}$
3.  $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$

**K = 1 is common, though  
larger K more efficient**



# 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.  $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$
- ~~these are correlated!~~  
use replay buffer

Q-learning is *not* gradient descent!

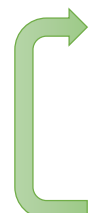
$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - (r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)))$$

no gradient through target value

**This is still a problem!**


# Q-Learning and Regression

full Q-learning with replay buffer:

- 
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.  $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i) (Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$

**one gradient step, moving target**

full fitted Q-iteration algorithm:

- 
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  $\phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

**perfectly well-defined, stable regression**

# Q-Learning with target networks

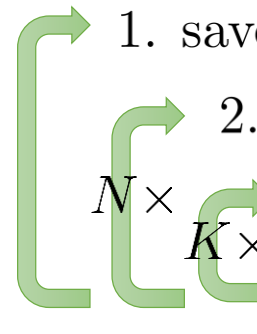
Q-learning with replay buffer and target network:

- 
1. save target network parameters:  $\phi' \leftarrow \phi$
  2. collect dataset  $\{(s_i, \mathbf{a}_i, s'_i, r_i)\}$  using some policy, add it to  $\mathcal{B}$
  - $N \times$  3. sample a batch  $(s_i, \mathbf{a}_i, s'_i, r_i)$  from  $\mathcal{B}$
  - $K \times$  4.  $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, \mathbf{a}_i)(Q_\phi(s_i, \mathbf{a}_i) - \underbrace{[r(s_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(s'_i, \mathbf{a}'_i)]}_{\text{targets don't change in inner loop!}})$

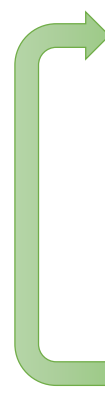
supervised regression

# “Classic” deep Q-learning algorithm (DQN)

Q-learning with replay buffer and target network:

- 
1. save target network parameters:  $\phi' \leftarrow \phi$
  2. collect dataset  $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$  using some policy, add it to  $\mathcal{B}$
  3. sample a batch  $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$  from  $\mathcal{B}$
  4.  $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$

“classic” deep Q-learning algorithm:

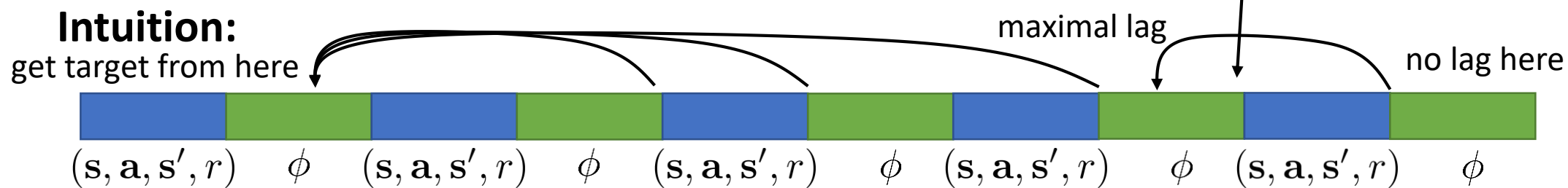
- 
1. take some action  $\mathbf{a}_i$  and observe  $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ , add it to  $\mathcal{B}$
  2. sample mini-batch  $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$  from  $\mathcal{B}$  uniformly
  3. compute  $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j,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

**You'll implement this in HW3!**

# Alternative target network

“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.  $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. update  $\phi'$



**Feels weirdly uneven, can we always have the same lag?**

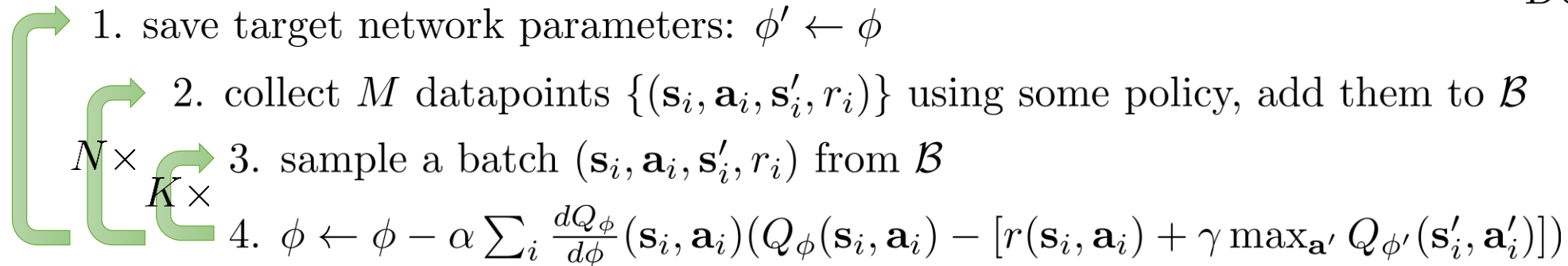
**Popular alternative (similar to Polyak averaging):**

5. update  $\phi'$ :  $\phi' \leftarrow \tau \phi' + (1 - \tau)\phi$   $\tau = 0.999$  works well

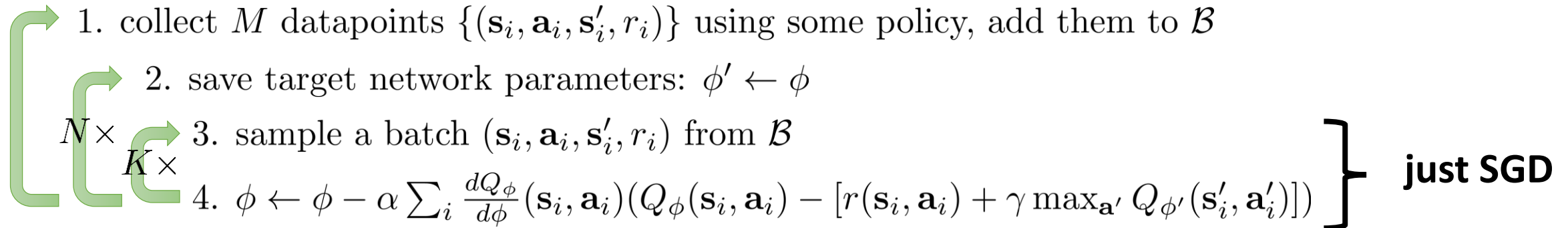
# Fitted Q-iteration and Q-learning

Q-learning with replay buffer and target network:

DQN:  $N = 1, K = 1$

- 
1. save target network parameters:  $\phi' \leftarrow \phi$
  2. collect  $M$  datapoints  $\{(s_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$  using some policy, add them to  $\mathcal{B}$
  - $N \times$  3. sample a batch  $(s_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$  from  $\mathcal{B}$
  - $K \times$  4.  $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, \mathbf{a}_i)(Q_\phi(s_i, \mathbf{a}_i) - [r(s_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$

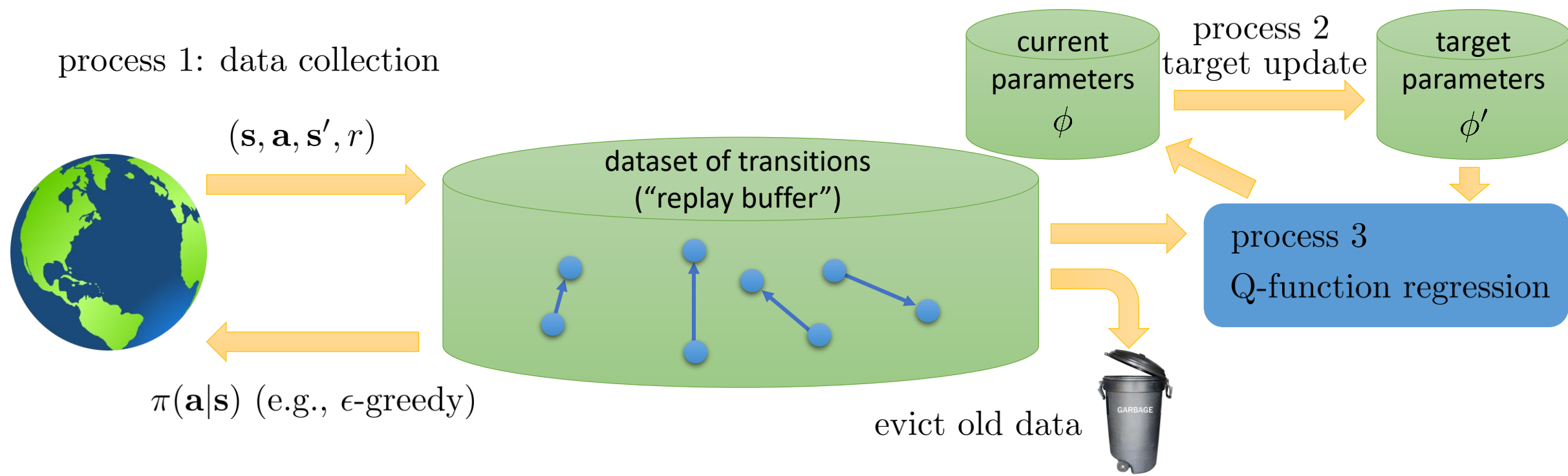
Fitted Q-learning (written similarly as above):

- 
1. collect  $M$  datapoints  $\{(s_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$  using some policy, add them to  $\mathcal{B}$
  2. save target network parameters:  $\phi' \leftarrow \phi$
  - $N \times$  3. sample a batch  $(s_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$  from  $\mathcal{B}$
  - $K \times$  4.  $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, \mathbf{a}_i)(Q_\phi(s_i, \mathbf{a}_i) - [r(s_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$
- } just SGD

# A more general view

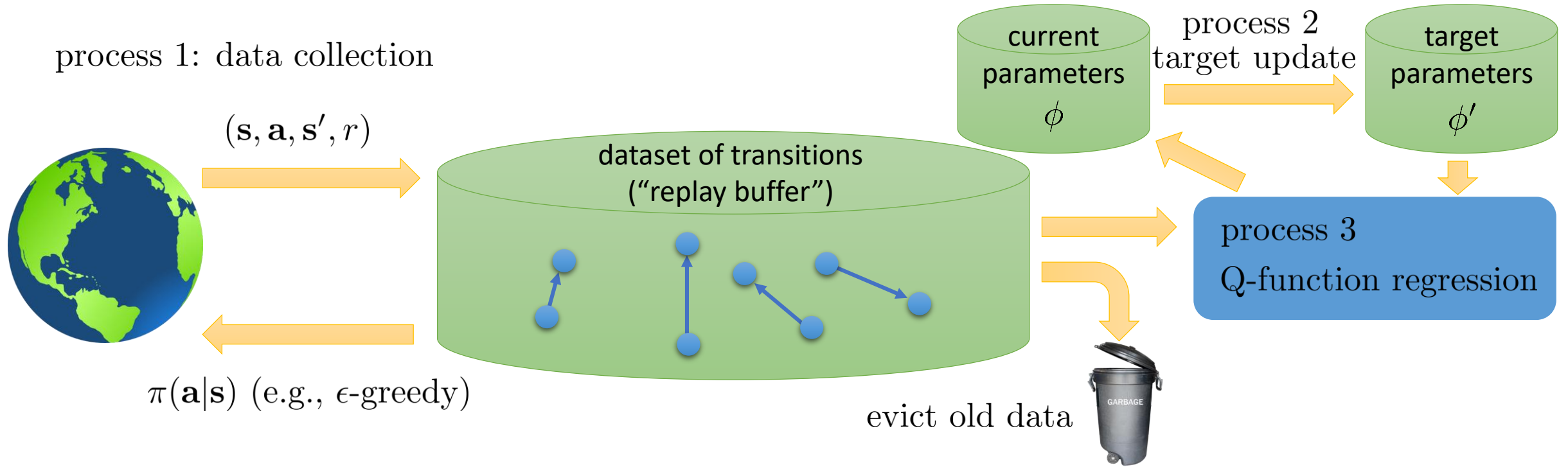
Q-learning with replay buffer and target network:

1. save target network parameters:  $\phi' \leftarrow \phi$
2. collect  $M$  datapoints  $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$  using some policy, add them to  $\mathcal{B}$
3. sample a batch  $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$  from  $\mathcal{B}$
4.  $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i) (Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$





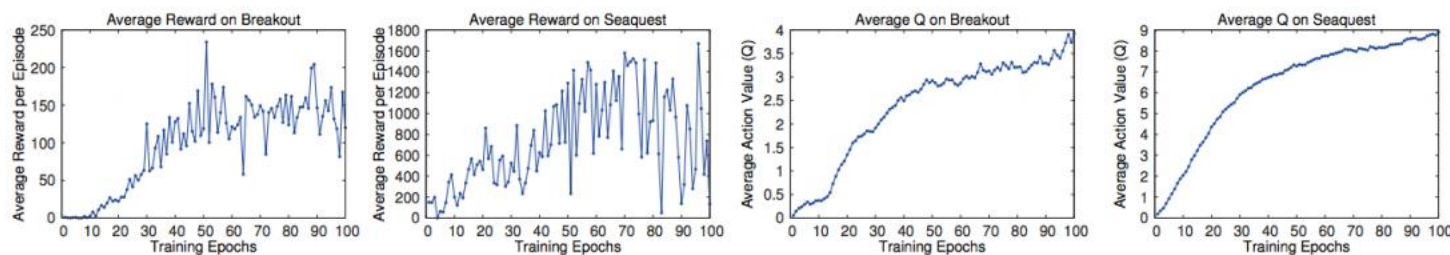
# A more general view



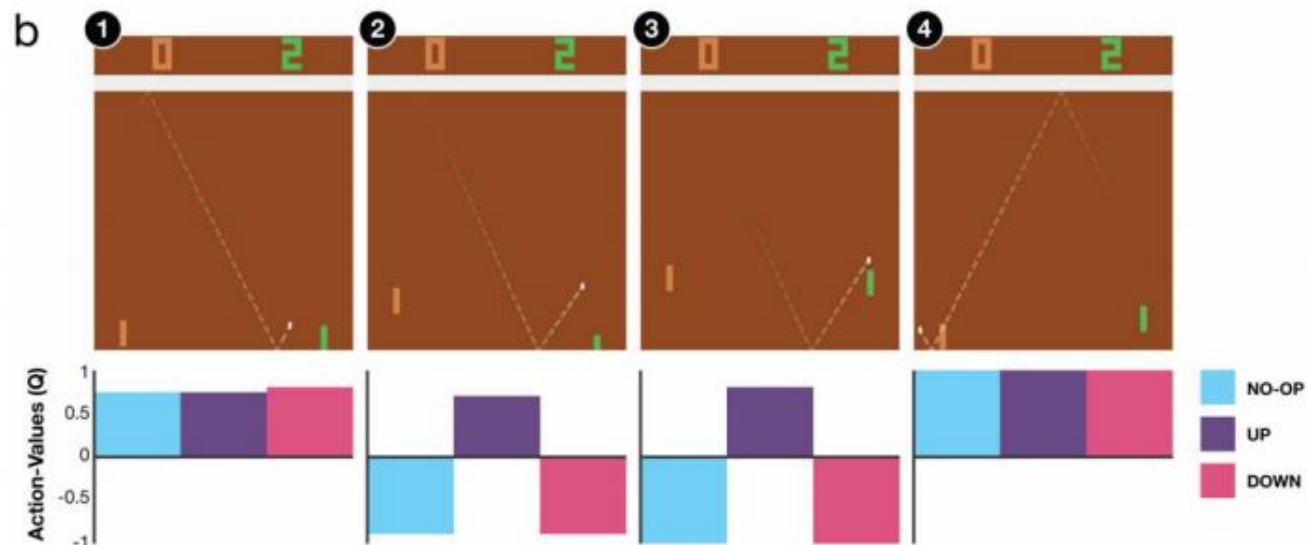
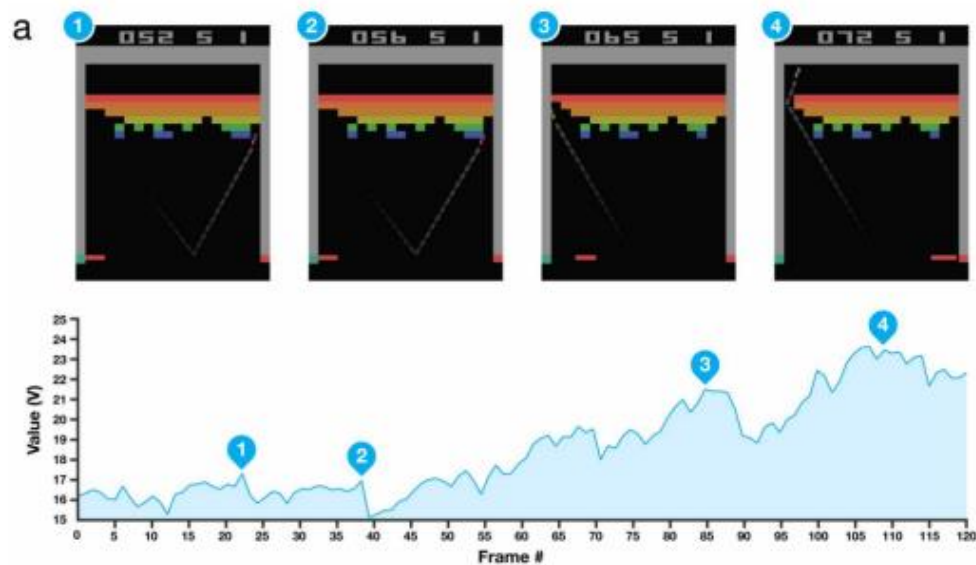
- Online Q-learning (last lecture): evict immediately, process 1, process 2, and process 3 all run at the same speed
- DQN: process 1 and process 3 run at the same speed, process 2 is slow
- Fitted Q-iteration: process 3 in the inner loop of process 2, which is in the inner loop of process 1

Break

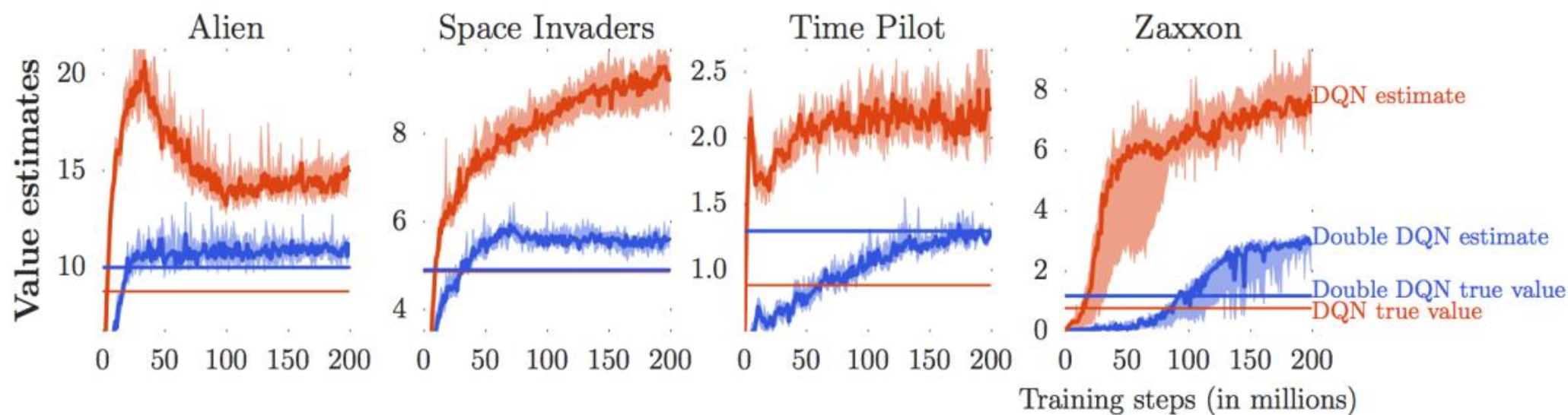
# Are the Q-values accurate?



**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  $\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}') = \underline{Q_{\phi'}(\mathbf{s}', \arg \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}'))}$


value *also* comes from  $Q_{\phi'}$     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  $\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}') = \underline{Q_{\phi'}(\mathbf{s}', \arg \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}'))}$

value *also* comes from  $Q_{\phi'}$     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?

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

$N$ -step return estimator



# Q-learning with N-step returns

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

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

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

+ less biased target values when Q-values are inaccurate

+ typically faster learning, especially early on

- only actually correct when learning on-policy

why?

we need transitions  $\mathbf{s}_{j,t'}, \mathbf{a}_{j,t'}, \mathbf{s}_{j,t'+1}$  to come from  $\pi$  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

# Q-learning with continuous actions

What's the problem with continuous actions?

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

**this max**

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

**this max  
particularly problematic (inner loop of training)**

How do we perform the max?

Option 1: optimization

- gradient based optimization (e.g., SGD) a bit slow in the inner loop
- action space typically low-dimensional – what about stochastic optimization?

# Q-learning with stochastic optimization

Simple solution:

$$\max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}) \approx \max \{Q(\mathbf{s}, \mathbf{a}_1), \dots, Q(\mathbf{s}, \mathbf{a}_N)\}$$

$(\mathbf{a}_1, \dots, \mathbf{a}_N)$  sampled from some distribution (e.g., uniform)

+ dead simple

+ efficiently parallelizable

- not very accurate

**but... do we care? How good does the target need to be anyway?**

More accurate solution:

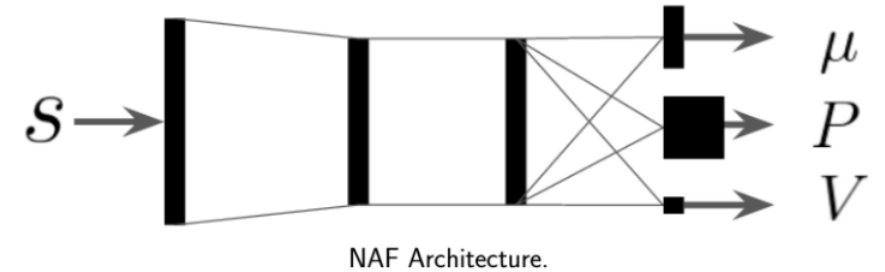
- cross-entropy method (CEM)
  - simple iterative stochastic optimization
- CMA-ES
  - substantially less simple iterative stochastic optimization

**works OK, for up to about 40 dimensions**

# Easily maximizable Q-functions

Option 2: use function class that is easy to optimize

$$Q_{\phi}(\mathbf{s}, \mathbf{a}) = -\frac{1}{2}(\mathbf{a} - \mu_{\phi}(\mathbf{s}))^T P_{\phi}(\mathbf{s})(\mathbf{a} - \mu_{\phi}(\mathbf{s})) + V_{\phi}(\mathbf{s})$$



## NAF: Normalized Advantage Functions

$$\arg \max_{\mathbf{a}} Q_{\phi}(\mathbf{s}, \mathbf{a}) = \mu_{\phi}(\mathbf{s}) \quad \max_{\mathbf{a}} Q_{\phi}(\mathbf{s}, \mathbf{a}) = V_{\phi}(\mathbf{s})$$

- + no change to algorithm
- + just as efficient as Q-learning
- loses representational power

# Q-learning with continuous actions

Option 3: learn an approximate maximizer

DDPG (Lillicrap et al., ICLR 2016)

“deterministic” actor-critic  
(really approximate Q-learning)

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

idea: train another network  $\mu_{\theta}(\mathbf{s})$  such that  $\mu_{\theta}(\mathbf{s}) \approx \arg \max_{\mathbf{a}} Q_{\phi}(\mathbf{s}, \mathbf{a})$

how? just solve  $\theta \leftarrow \arg \max_{\theta} Q_{\phi}(\mathbf{s}, \mu_{\theta}(\mathbf{s}))$

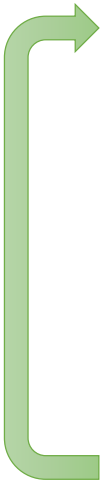
$$\frac{dQ_{\phi}}{d\theta} = \frac{d\mathbf{a}}{d\theta} \frac{dQ_{\phi}}{d\mathbf{a}}$$

new target  $y_j = r_j + \gamma Q_{\phi'}(\mathbf{s}'_j, \mu_{\theta}(\mathbf{s}'_j)) \approx r_j + \gamma Q_{\phi'}(\mathbf{s}'_j, \arg \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j))$

# Q-learning with continuous actions

Option 3: learn an approximate maximizer

DDPG:

- 
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  $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mu_{\theta'}(\mathbf{s}'_j))$  using *target* nets  $Q_{\phi'}$  and  $\mu_{\theta'}$
  4.  $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
  5.  $\theta \leftarrow \theta + \beta \sum_j \frac{d\mu}{d\theta}(\mathbf{s}_j) \frac{dQ_\phi}{d\mathbf{a}}(\mathbf{s}_j, \mathbf{a})$
  6. update  $\phi'$  and  $\theta'$  (e.g., Polyak averaging)

# 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

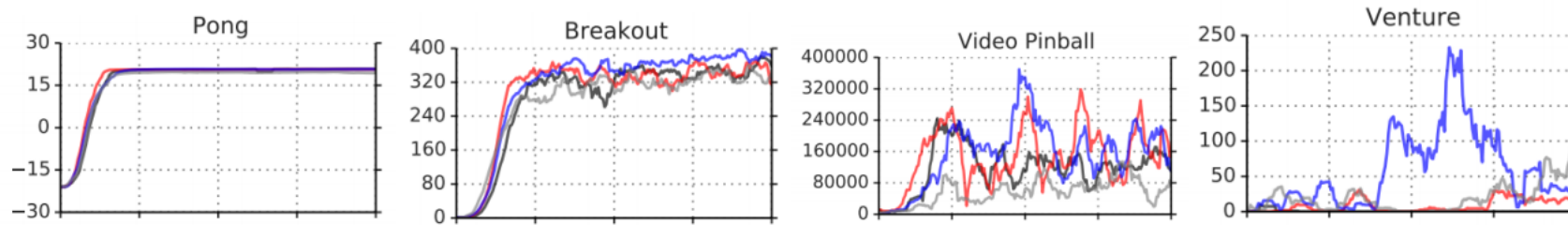


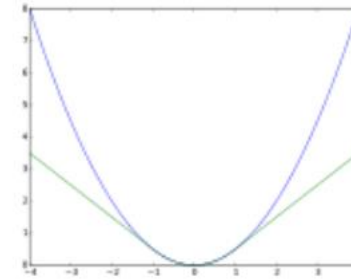
Figure: From T. Schaul, J. Quan, I. Antonoglou, and D. Silver. “Prioritized experience replay”. *arXiv preprint arXiv:1511.05952* (2015), Figure 7

- 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

# Advanced tips for Q-learning

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

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

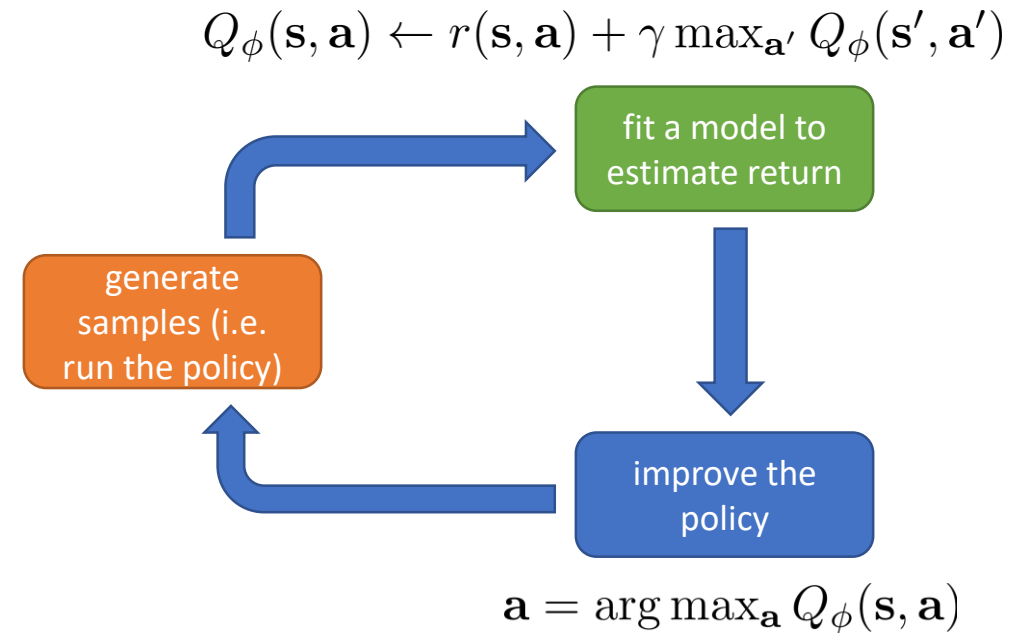


- Double Q-learning helps *a lot* in practice, simple and no downsides
- N-step returns also help a lot, but have some downsides
- Schedule exploration (high to low) and learning rates (high to low), Adam optimizer can help too
- Run multiple random seeds, it's very inconsistent between runs



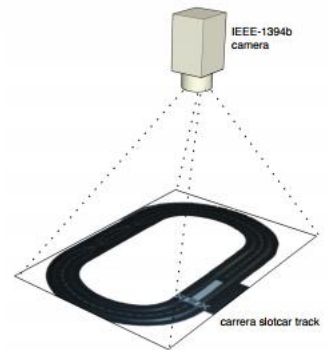
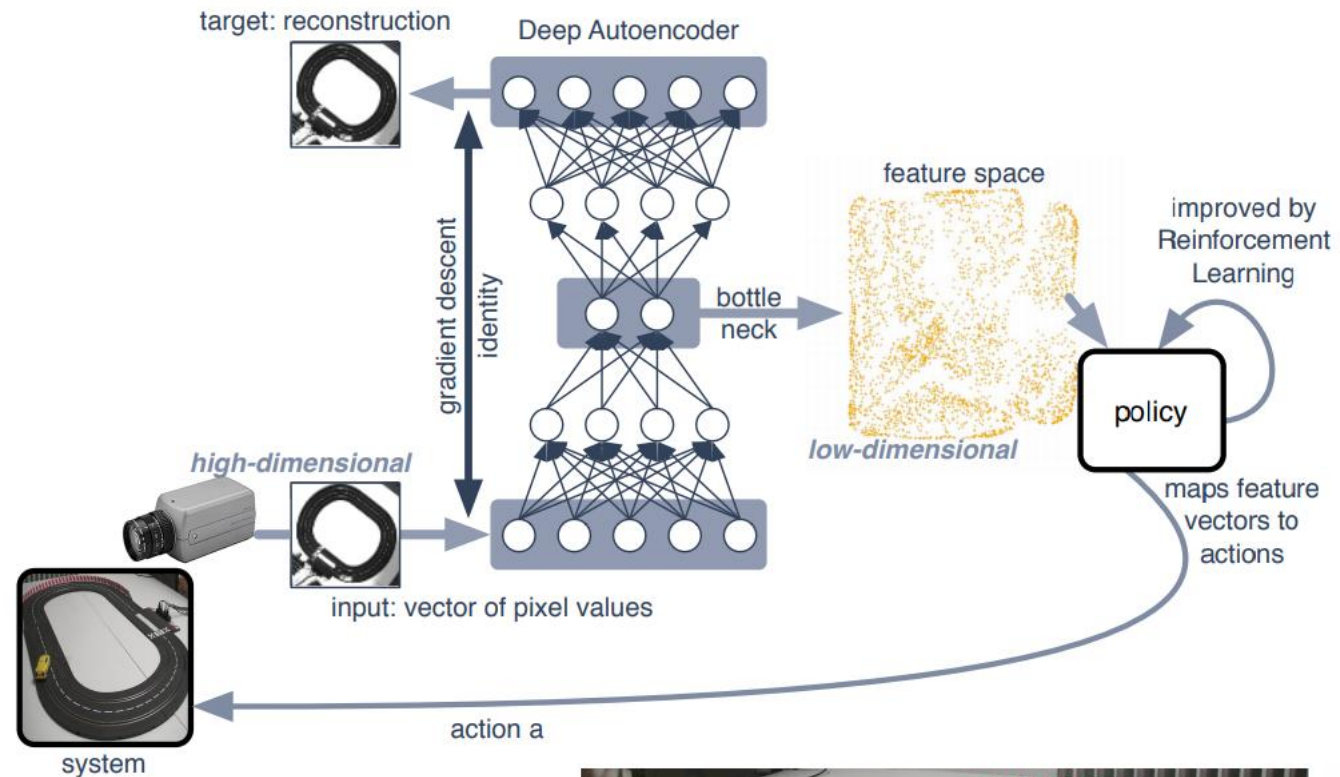
# Review

- Q-learning in practice
  - Replay buffers
  - Target networks
- Generalized fitted Q-iteration
- Double Q-learning
- Multi-step Q-learning
- Q-learning with continuous actions
  - Random sampling
  - Analytic optimization
  - Second “actor” network



# Fitted Q-iteration in a latent space

- “Autonomous reinforcement learning from raw visual data,” Lange & Riedmiller ‘12
- Q-learning on top of latent space learned with autoencoder
- Uses fitted Q-iteration
- Extra random trees for function approximation (but neural net for embedding)



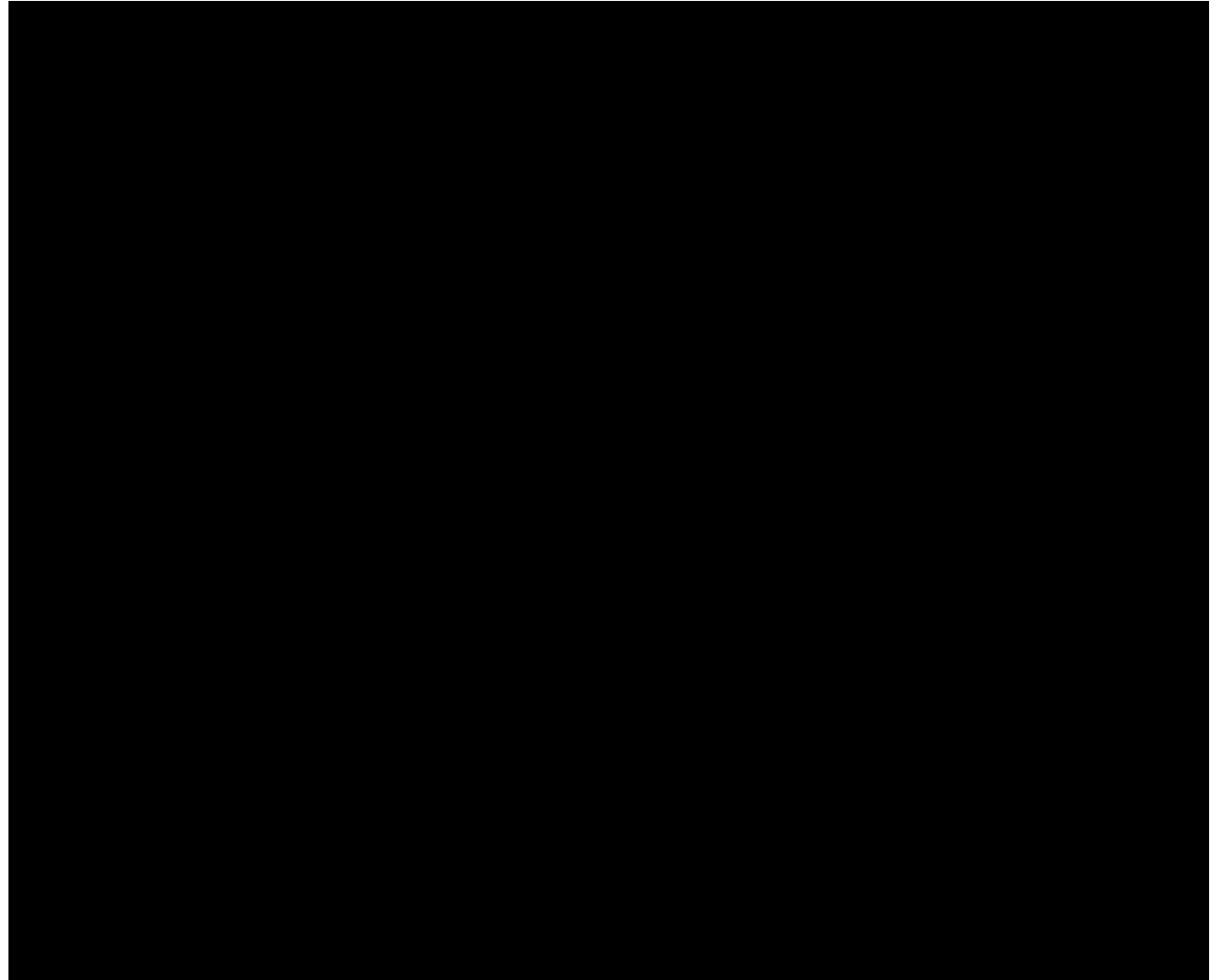
# Q-learning with convolutional networks

- “Human-level control through deep reinforcement learning,” Mnih et al. ‘13
- Q-learning with convolutional networks
- Uses replay buffer and target network
- One-step backup
- One gradient step
- Can be improved a lot with double Q-learning (and other tricks)



# Q-learning with continuous actions

- “Continuous control with deep reinforcement learning,” Lillicrap et al. ‘15
- Continuous actions with maximizer network
- Uses replay buffer and target network (with Polyak averaging)
- One-step backup
- One gradient step per simulator step

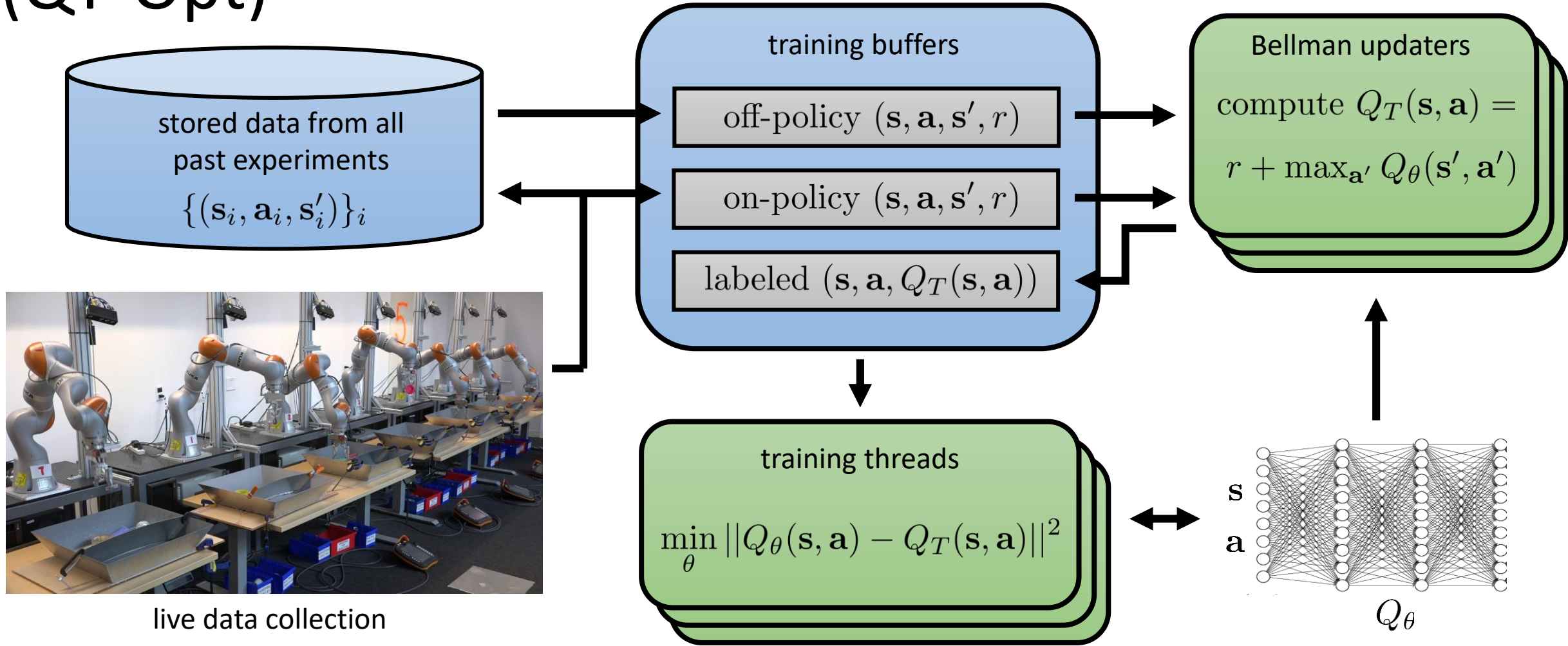


# Q-learning on a real robot

- “Robotic manipulation with deep reinforcement learning and ...,” Gu\*, Holly\*, et al. ‘17
- Continuous actions with NAF (quadratic in actions)
- Uses replay buffer and target network
- One-step backup
- Four gradient steps per simulator step for efficiency
- Parallelized across multiple robots



# Large-scale Q-learning with continuous actions (QT-Opt)



live data collection

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



# Q-learning suggested readings

- Classic papers
  - Watkins. (1989). Learning from delayed rewards: introduces Q-learning
  - Riedmiller. (2005). Neural fitted Q-iteration: batch-mode Q-learning with neural networks
- Deep reinforcement learning Q-learning papers
  - Lange, Riedmiller. (2010). Deep auto-encoder neural networks in reinforcement learning: early image-based Q-learning method using autoencoders to construct embeddings
  - Mnih et al. (2013). Human-level control through deep reinforcement learning: Q-learning with convolutional networks for playing Atari.
  - Van Hasselt, Guez, Silver. (2015). Deep reinforcement learning with double Q-learning: a very effective trick to improve performance of deep Q-learning.
  - Lillicrap et al. (2016). Continuous control with deep reinforcement learning: continuous Q-learning with actor network for approximate maximization.
  - Gu, Lillicrap, Stuskever, L. (2016). Continuous deep Q-learning with model-based acceleration: continuous Q-learning with action-quadratic value functions.
  - Wang, Schaul, Hessel, van Hasselt, Lanctot, de Freitas (2016). Dueling network architectures for deep reinforcement learning: separates value and advantage estimation in Q-function.