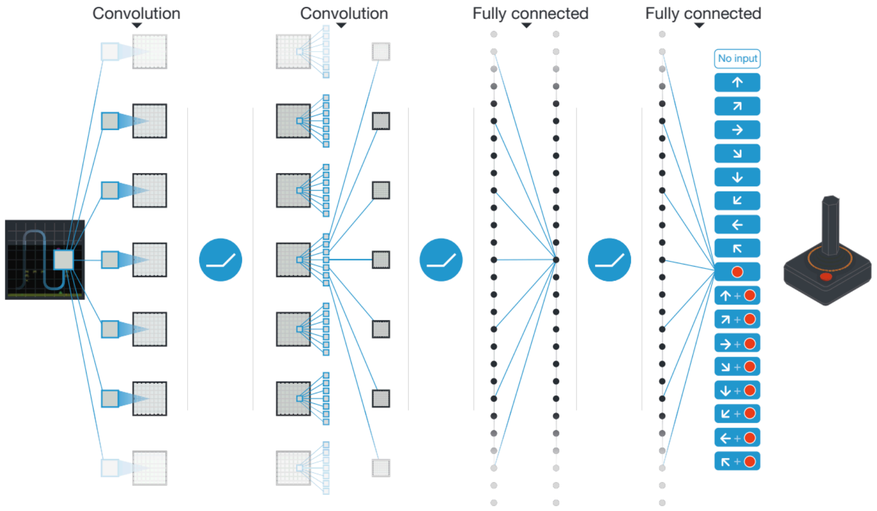
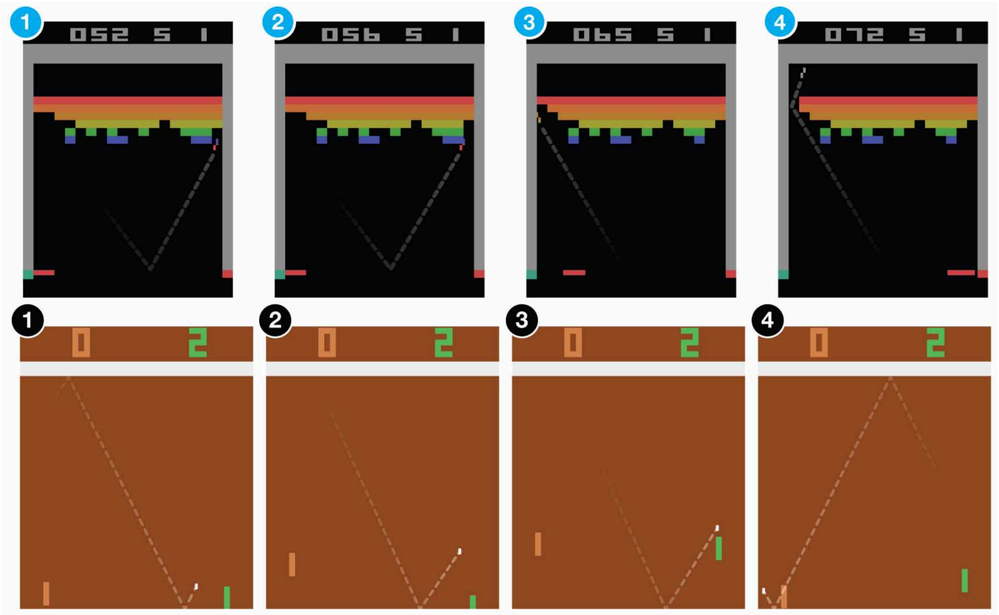
CS 182/282A: Designing, Visualizing, and Understanding Deep Neural Networks





Daniel Seita

Lecture 21

Deep Reinforcement Learning: Value Based Methods

Slides mostly adapted from past CS 294-112 lectures

Last Time: Markov Decision Processes

Definitions

Markov decision process $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$

$\mathcal{S}$ – state space states $s \in \mathcal{S}$ (discrete or continuous)

$\mathcal{A}$ – action space actions $a \in \mathcal{A}$ (discrete or continuous)

$\mathcal{T}$ – transition operator (now a tensor!)

r – reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

$r(s_t, a_t)$ – reward

$\tau = (s_0, a_0, s_1, a_1, \ldots, s_T, a_T)$ – trajectory – a sequence of states and actions

Andrey Markov

Richard Bellman

Last Time: Reinforcement Learning Overview

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

Expected one-step reward

$$\theta^\star = \arg\max_\theta E_{(\mathbf{s},\mathbf{a}) \sim p_\theta(\mathbf{s},\mathbf{a})} [r(\mathbf{s}, \mathbf{a})]$$

infinite horizon case

$$\theta^\star = \arg\max_\theta \sum_{t=1}^{T} E_{(\mathbf{s}_t,\mathbf{a}_t) \sim p_\theta(\mathbf{s}_t,\mathbf{a}_t)} [r(\mathbf{s}_t, \mathbf{a}_t)]$$

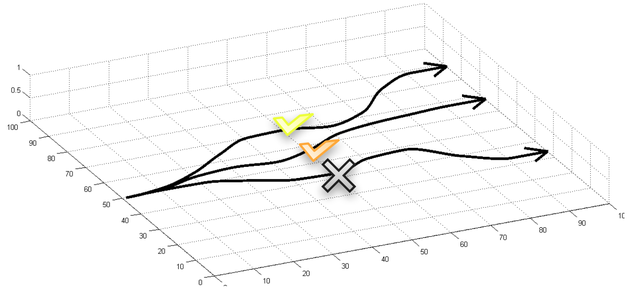
finite horizon case

Last Time: Policy Gradients

Optimizing the Model

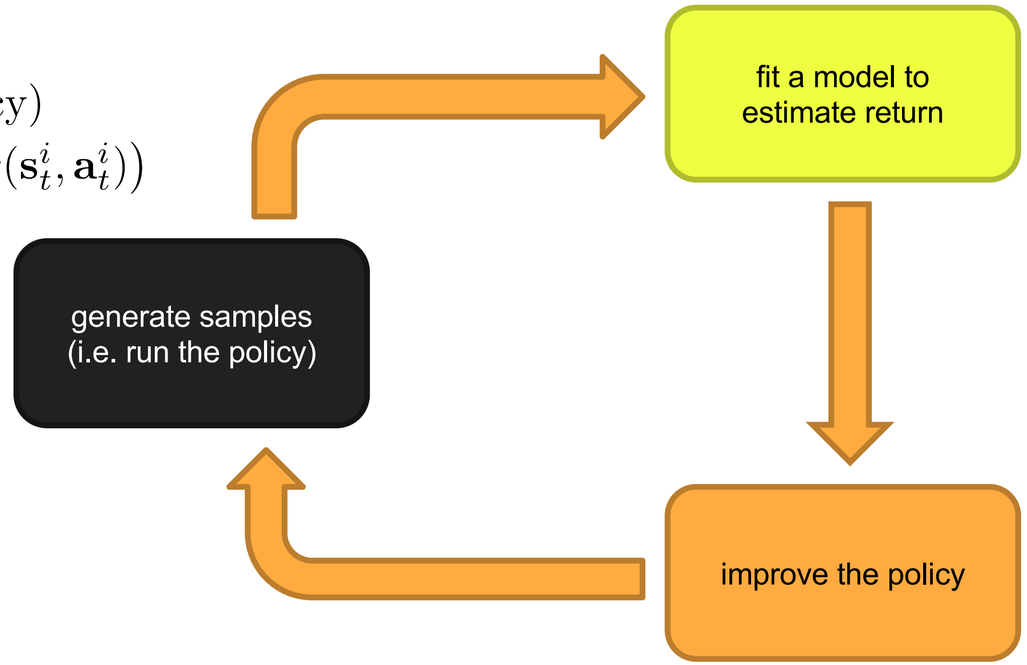
$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

To optimize, simply iterate: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$



REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i) \right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i) \right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$



- Also reviewed more advanced algorithms such as TRPO and PPO.
- Idea: take "small" gradient steps so that the policy doesn't change much.

Logistics

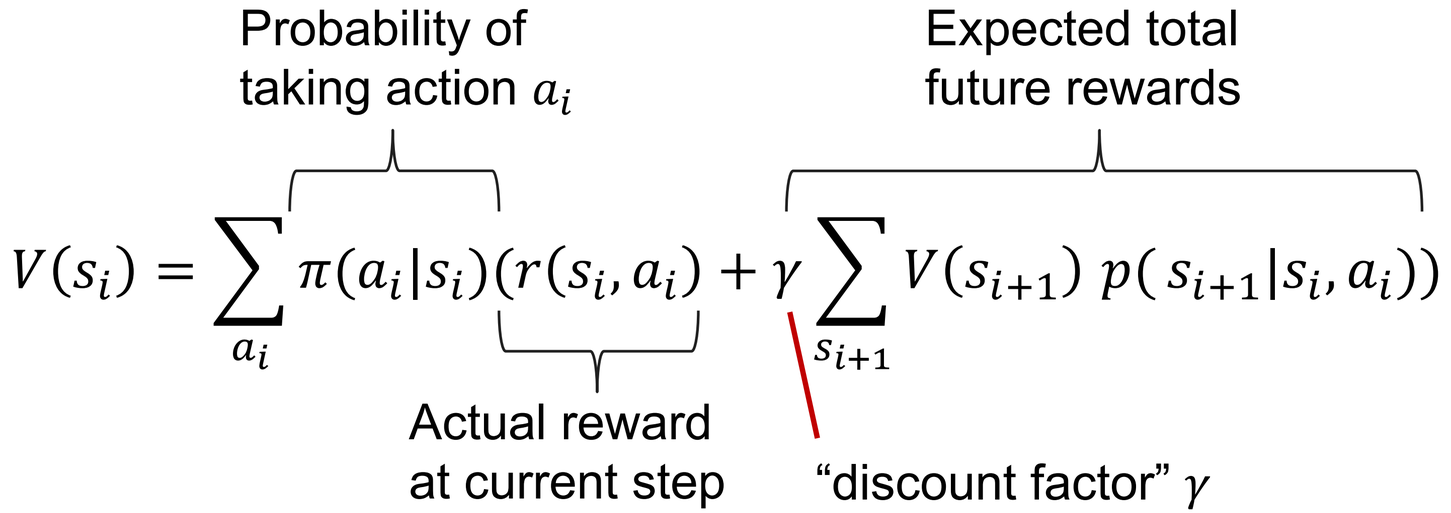
- Second project checkpoint, graded pass/fail. Please complete it.
- Poster session: Tuesday May 7, 2:00p to 4:00p, 310 Jacobs Hall.
- Project reports: due Monday May 13.
- HW 4 deadline extended to this Friday April 26, at 11:00pm.
- This lecture hopefully provides all info you need for completing Part 2.
  - Part 1 material was covered last week.

Today: Deep RL, Value-Based Methods

- Value Functions
- Actor-Critics and Q-Learning
- Deep Q-Network (DQN)
- Double DQN (DDQN)
- Practical Tips

Value Functions

We can define the value of a state s_i under a given policy π, $V(s_i)$ as the (discounted) *reward-to-go* from that state:



$0 < \lambda \leq 1$ is typically close to 1.

$\lambda < 1$ favors short-term rewards, and causes the recurrence to converge on all MDPs.

Several "Value Function" Options

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \mid s_0 = s, a_0 = a \right]$$

Called Q-function or state-action-value function

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi \left[r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \mid s_0 = s \right] \\ &= \mathbb{E}_{a \sim \pi} \left[Q^\pi(s,a) \right] \end{aligned}$$

Called state-value function

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

Called advantage function

Recall that TRPO and PPO use Advantage

TRPO optimizes: $L(\theta') = E\left[\frac{\pi_{\theta'}(a|s)}{\pi_\theta(a|s)} A(s,a)\right]$ with a KL-divergence regularization loss to avoid large changes in π_θ.

PPO combines the main and regularization loss into a single formula. First define

$$r_t(\theta) = \frac{\pi_{\theta'}(a|s)}{\pi_\theta(a|s)}$$

the PPO objective is

$$L(\theta') = E[\min(r_t A_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon) A_t)]$$

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

Actor-Critic Introduction

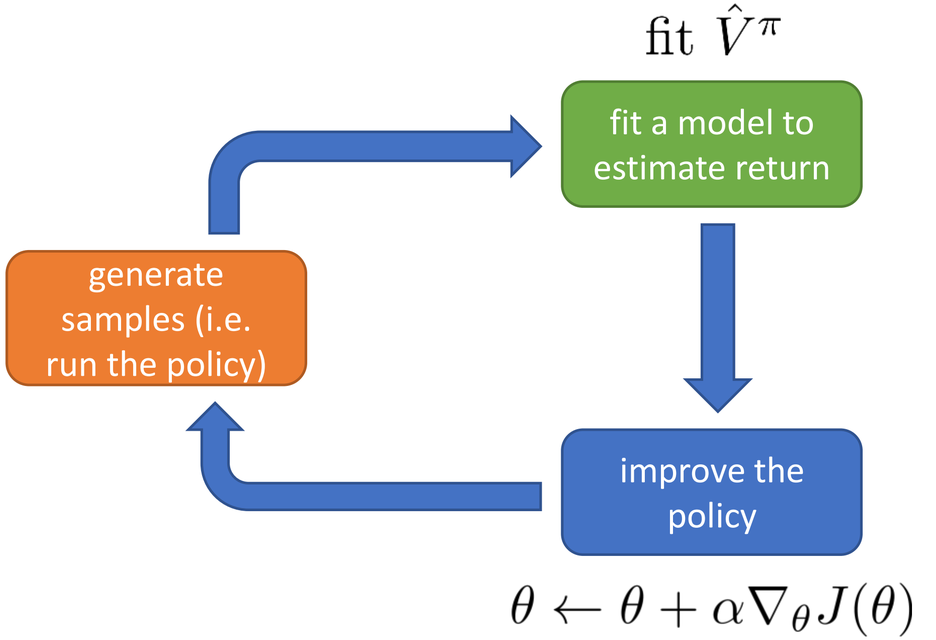
- TRPO and PPO mainly viewed as "policy gradient" methods, but there is another (not entirely disjoint) category of methods known as "actor critic":
  - Actor: the policy.
  - Critic: the value estimator.
- For policy gradient actor critic methods, the "critic" is usually viewed as the learned state-dependent baseline which reduces variance (see [1]).
- We will discuss "critic-only" algorithms (e.g., DQN and DDQN) soon.

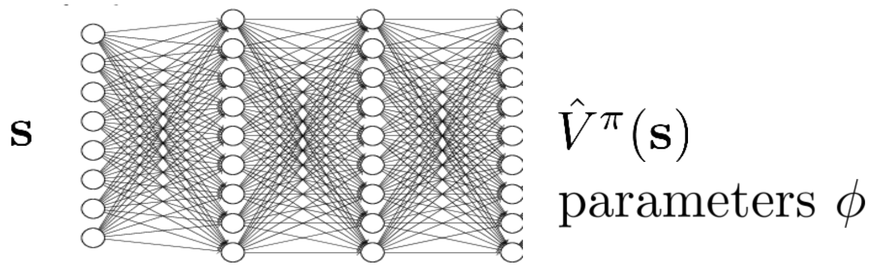
[1]: Mnih et al., *Asynchronous Methods for Deep Reinforcement Learning*, ICML 2016.

Actor-critic algorithms

batch actor-critic algorithm:

1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it on the robot)
2. fit $\hat{V}^\pi_\phi(\mathbf{s})$ to sampled reward sums
3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}^\pi_\phi(\mathbf{s}'_i) - \hat{V}^\pi_\phi(\mathbf{s}_i)$
4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

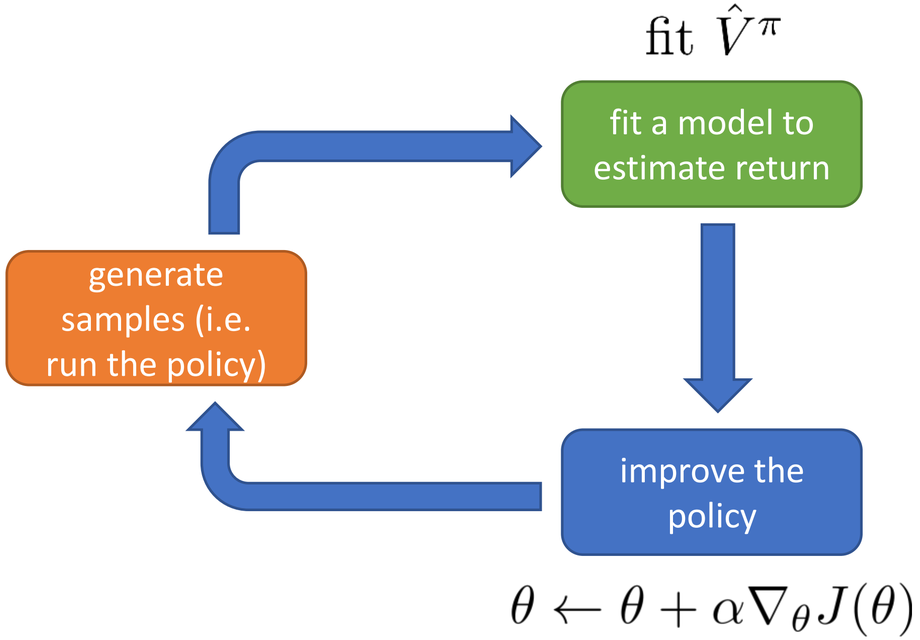




Actor-critic algorithms

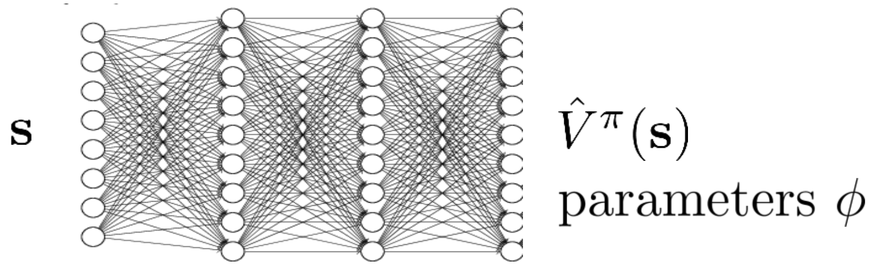
batch actor-critic algorithm:

1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it on the robot)
2. fit $\hat{V}^\pi_\phi(\mathbf{s})$ to sampled reward sums
3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}^\pi_\phi(\mathbf{s}'_i) - \hat{V}^\pi_\phi(\mathbf{s}_i)$
4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$



online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update $\hat{V}^\pi_\phi$ using target $r + \gamma \hat{V}^\pi_\phi(\mathbf{s}')$
3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}^\pi_\phi(\mathbf{s}') - \hat{V}^\pi_\phi(\mathbf{s})$
4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^\pi(\mathbf{s}, \mathbf{a})$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$



Can we omit policy gradient completely?

$A^\pi(\mathbf{s}_t, \mathbf{a}_t)$: how much better is $\mathbf{a}_t$ than the average action according to π

$\arg\max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t)$: best action from $\mathbf{s}_t$, if we then follow π

at *least* as good as any $\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t)$

regardless of what $\pi(\mathbf{a}_t|\mathbf{s}_t)$ is!

Can we omit policy gradient completely?

$A^\pi(\mathbf{s}_t, \mathbf{a}_t)$: how much better is $\mathbf{a}_t$ than the average action according to π

$\arg\max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t)$: best action from $\mathbf{s}_t$, if we then follow π

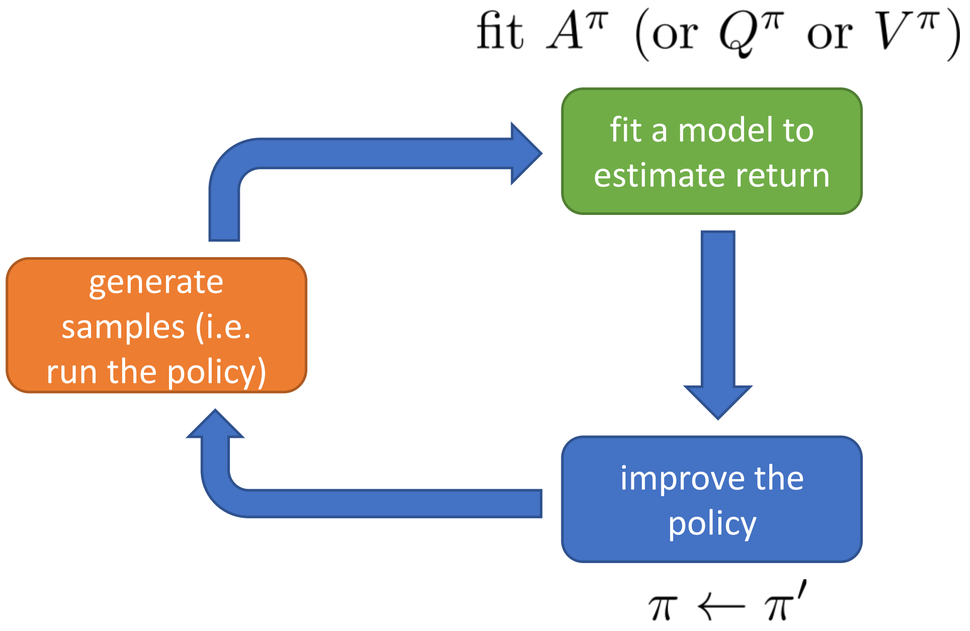
at *least* as good as any $\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t)$

regardless of what $\pi(\mathbf{a}_t|\mathbf{s}_t)$ is!

forget policies, let's just do this!

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

as good as π
(probably better)



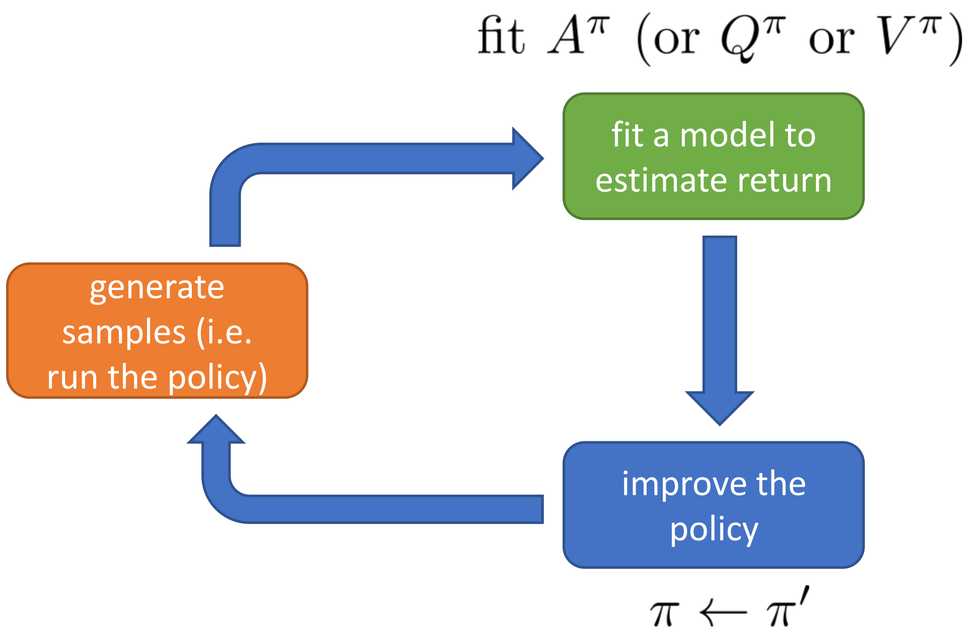
Policy iteration

High level idea:

policy iteration algorithm:

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

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



Policy iteration

High level idea:

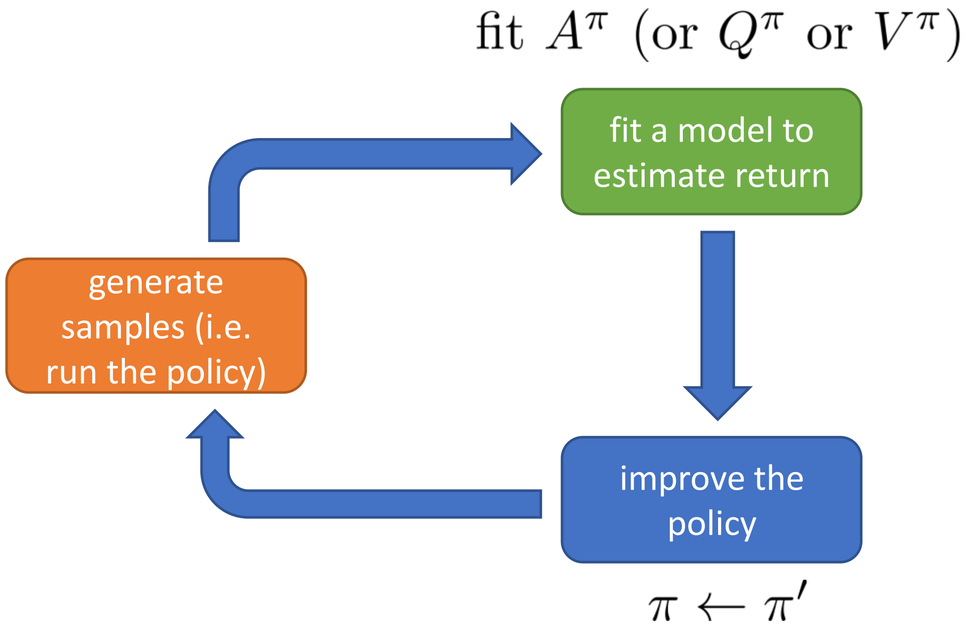
policy iteration algorithm:

1. evaluate $A^\pi(\mathbf{s}, \mathbf{a})$ ⟵ how to do this?
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: $A^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma E[V^\pi(\mathbf{s}')] - V^\pi(\mathbf{s})$

let's evaluate $V^\pi(\mathbf{s})$!



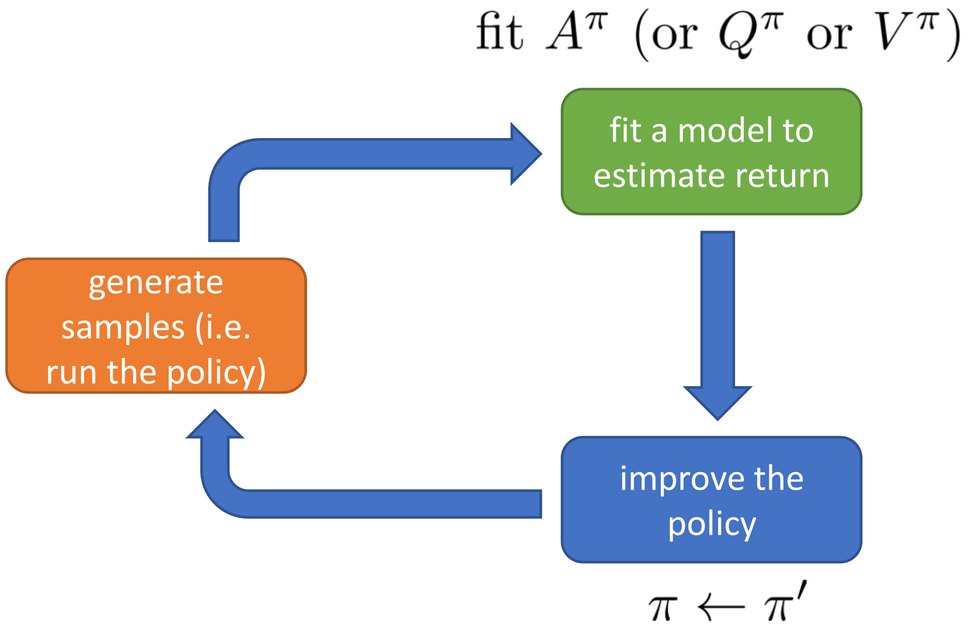
Policy iteration

High level idea:

policy iteration algorithm:

1. evaluate $A^\pi(\mathbf{s}, \mathbf{a})$ ⟵ how to do this?
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: $A^\pi(\mathbf{s}, \mathbf{a}) = \underbrace{r(\mathbf{s}, \mathbf{a}) + \gamma E[V^\pi(\mathbf{s}')]}_{Q^\pi(\mathbf{s}, \mathbf{a})} - V^\pi(\mathbf{s})$ $\Longrightarrow$ $\arg\max_{\mathbf{a}_t} A^\pi(\mathbf{s}_t, \mathbf{a}_t) = \arg\max_{\mathbf{a}_t} Q^\pi(\mathbf{s}_t, \mathbf{a}_t)$

Fitted Q-iteration

full fitted Q-iteration algorithm:

parameters

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

dataset size N, collection policy

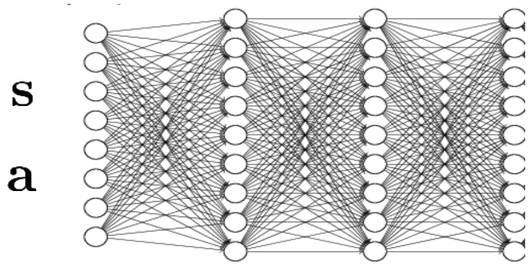
$K\times$

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

iterations K

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

gradient steps S



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

Why is this algorithm off-policy?

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

Why is this algorithm off-policy?

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

given $\mathbf{s}$ *and* $\mathbf{a}$, transition is independent of π

this approximates the value of π' at $\mathbf{s}'_i$

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

Why is this algorithm off-policy?

full fitted Q-iteration algorithm:

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

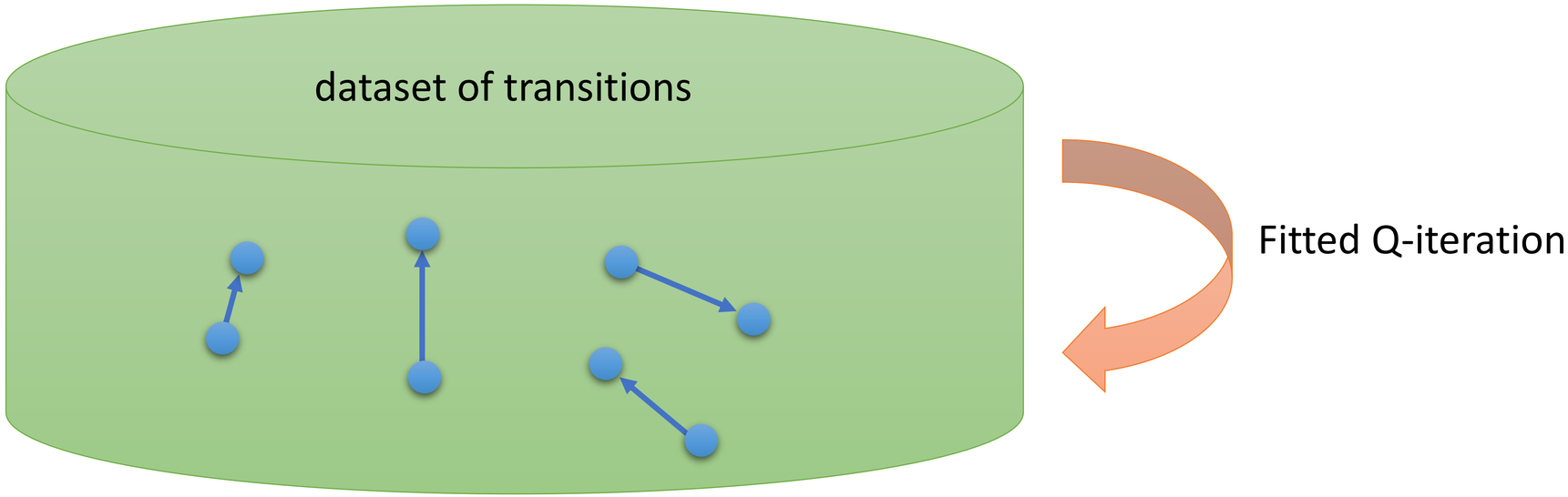
$K\times$

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

given $\mathbf{s}$ *and* $\mathbf{a}$, transition is independent of π

this approximates the value of π' at $\mathbf{s}'_i$

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



What is fitted Q-iteration optimizing?

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 $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$ ← this max improves the policy (tabular case)
3. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

↑ error $\mathcal{E}$

$$\mathcal{E} = \frac{1}{2} E_{(\mathbf{s},\mathbf{a})\sim\beta} \left[Q_\phi(\mathbf{s}, \mathbf{a}) - [r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}', \mathbf{a}')] \right]$$

What is fitted Q-iteration optimizing?

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 $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$ ← this max improves the policy (tabular case)
3. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$

↑ error $\mathcal{E}$

$$\mathcal{E} = \frac{1}{2} E_{(\mathbf{s},\mathbf{a})\sim\beta} \left[Q_\phi(\mathbf{s}, \mathbf{a}) - [r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}', \mathbf{a}')] \right]$$

if $\mathcal{E} = 0$, then $Q_\phi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}', \mathbf{a}')$

this is an *optimal* Q-function, corresponding to optimal policy π':

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

maximizes reward

sometimes written $Q^\star$ and $\pi^\star$

most guarantees are lost when we leave the tabular case (e.g., when we use neural network function approximation)

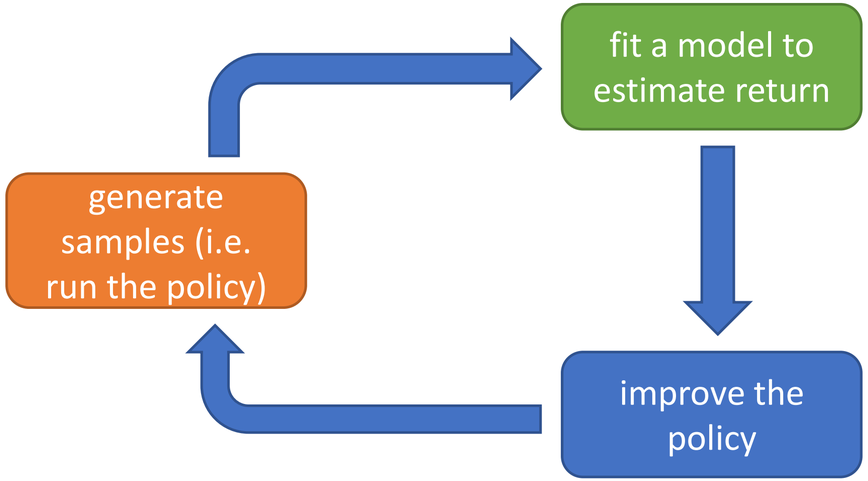
Online Q-learning algorithms

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

full fitted Q-iteration algorithm:

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

($K\times$)



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

off policy, so many choices here!

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

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

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

final 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 is this a bad idea for step 1?

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

final 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 is this a bad idea for step 1?

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

“epsilon-greedy”

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

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

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

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

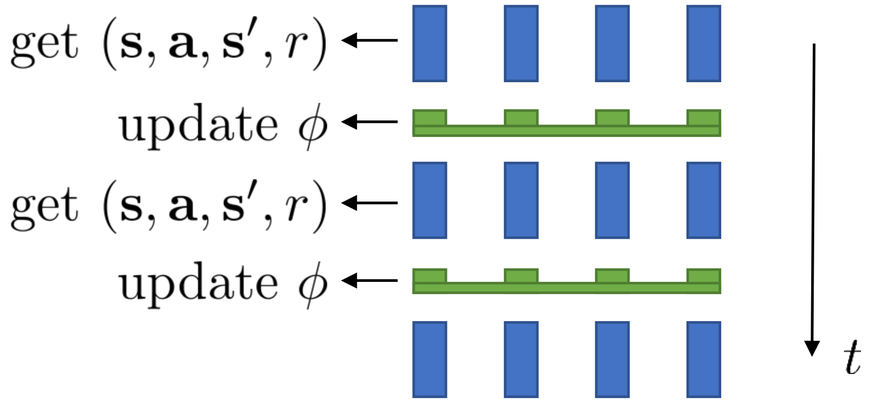
Correlated samples in online Q-learning

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



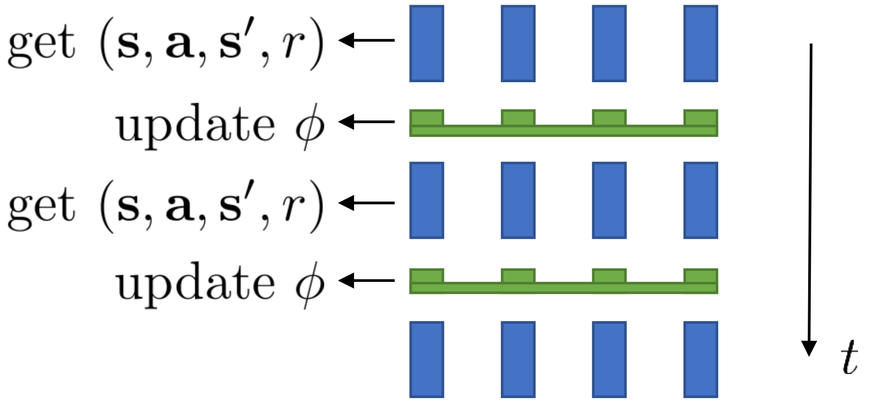
Correlated samples in online Q-learning

online Q iteration algorithm:

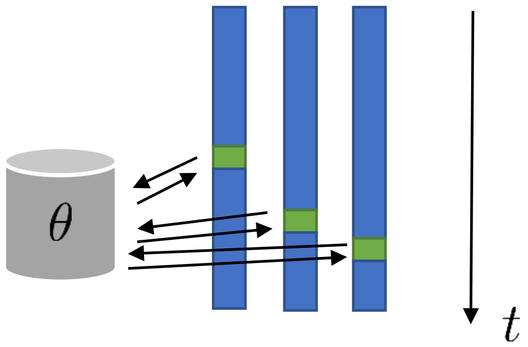
1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$

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

synchronized parallel Q-learning



asynchronous parallel Q-learning



Note: can apply same idea for policy gradients + actor critics. Mnih et al., *Asynchronous Methods for Deep RL*, 2016.

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

special case with K = 1, and one gradient step

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

$K\times$

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

special case with K = 1, and one gradient step

full fitted Q-iteration algorithm:

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

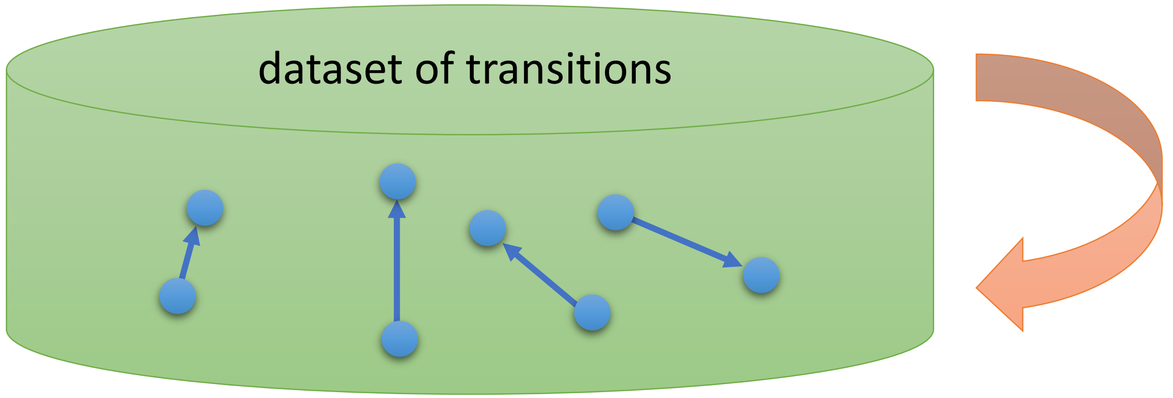
$K\times$

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



Fitted Q-iteration

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)

Another solution: replay buffers

Q-learning with a replay buffer:

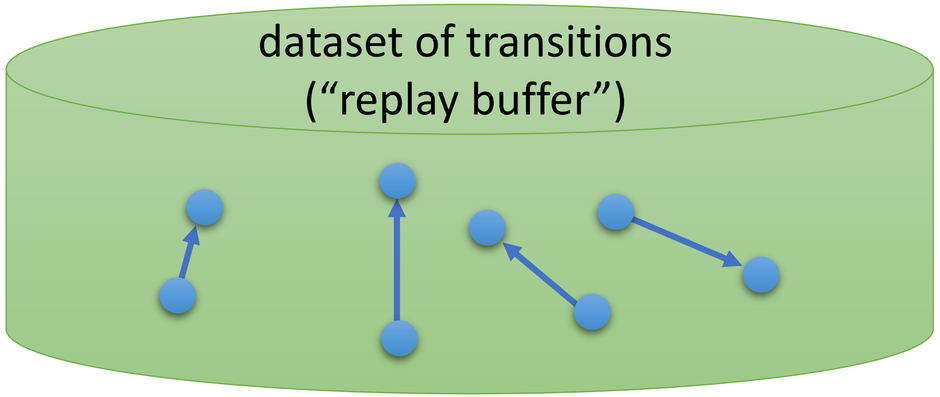
1. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
2. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$

+ 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…



Another solution: replay buffers

Q-learning with a replay buffer:

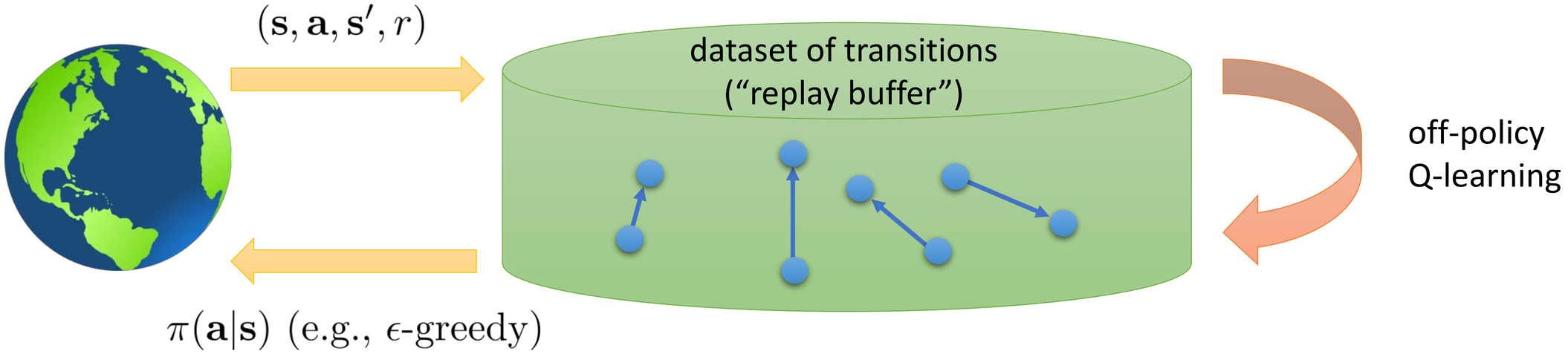
1. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
2. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$

+ samples are no longer correlated

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

but where does the data come from?

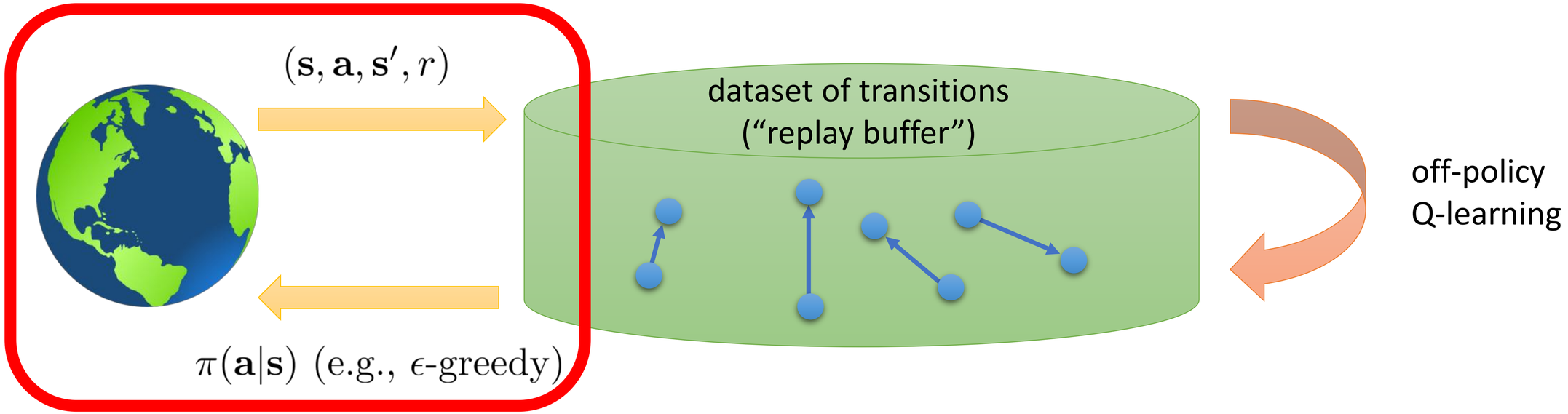
need to periodically feed the replay buffer…



Putting it together

full Q-learning with replay buffer:

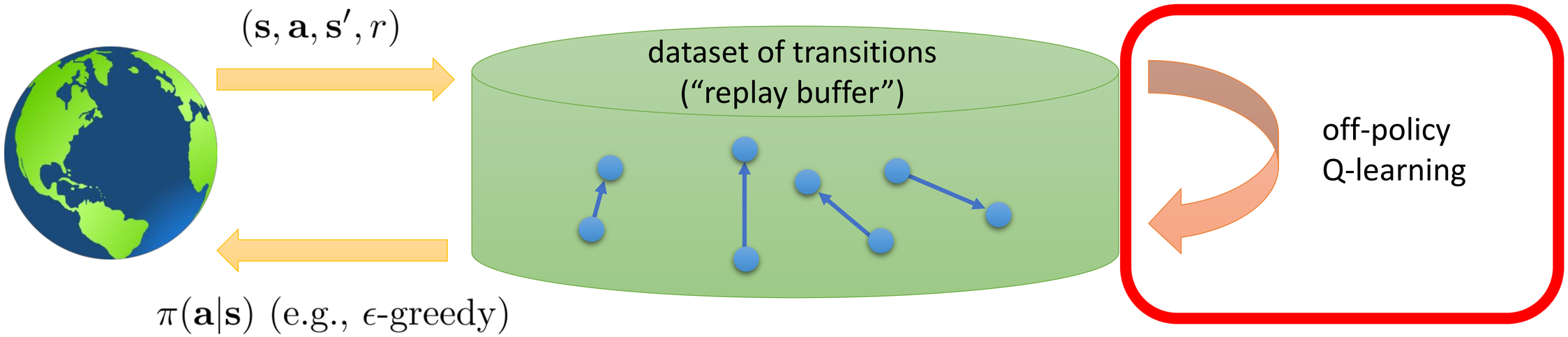
1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{B}$



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 $\mathcal{B}$



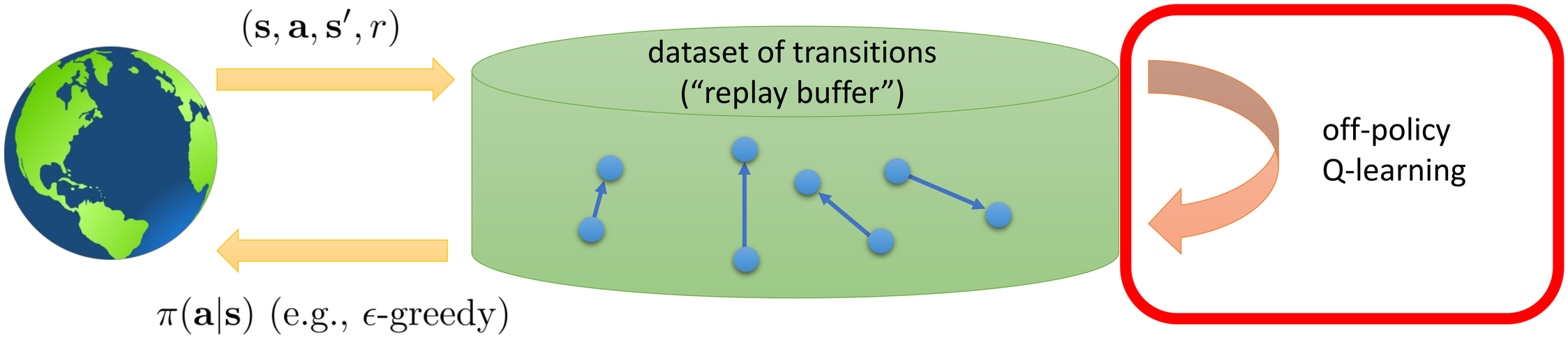
Putting it together

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 $\mathcal{B}$

$K\times$

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



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!

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

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!

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

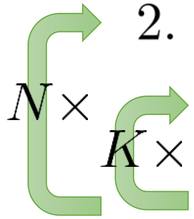
no gradient through target value

This is a problem!

Q-Learning with target networks

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 $\mathcal{B}$

$N\times$

3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$

$K\times$

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

Q-Learning with target networks

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

$N\times$ (steps 1–4) $K\times$ (steps 3–4)

targets don't change in inner loop!

supervised regression

Classic DQN Algorithm You'll Implement

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For t = 1,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$ in D

    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

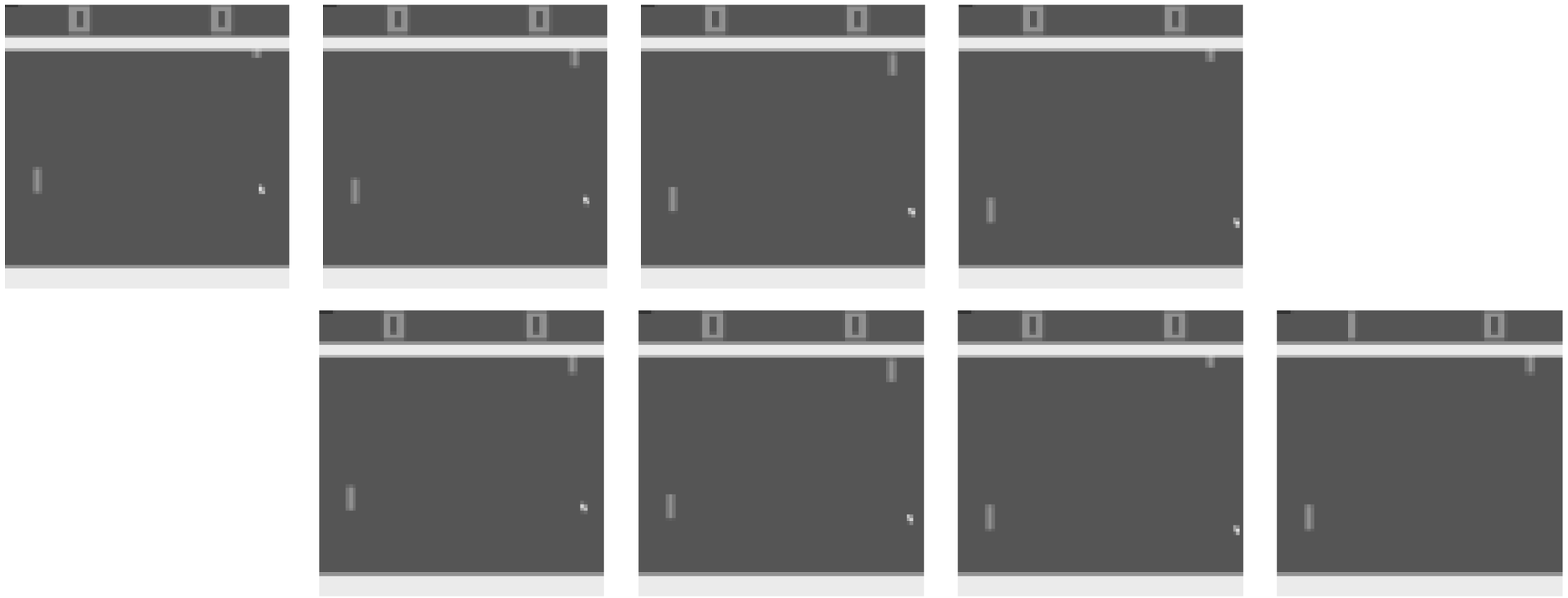
  End For

End For

- From Mnih et al., 2015*.
- Notation: theta now represents Q-network parameters, and phi represents processed (84 x 84 x 4) states.

*You can also look at their earlier 2013 workshop paper.

Wait, Why is There a "4" Here?

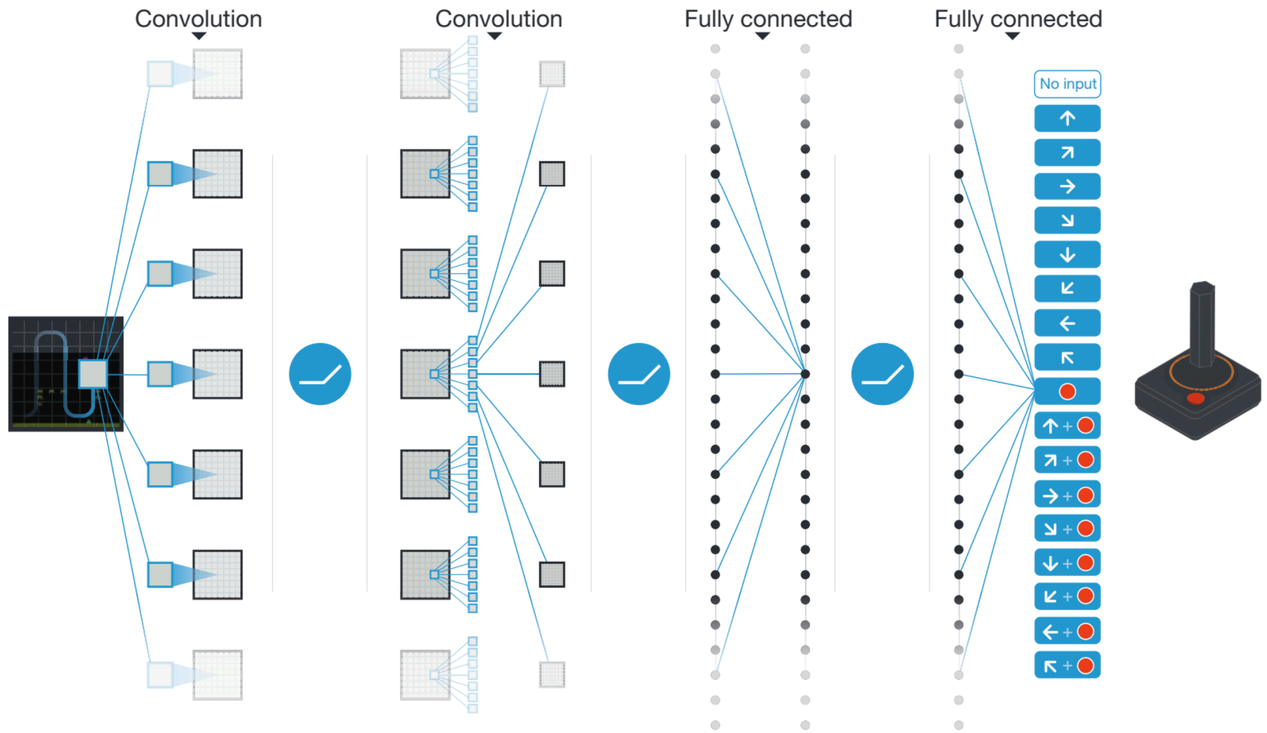


(84 x 84 x 4)
processed state
at time t

(84 x 84 x 4)
processed state
at time t+1

- To know what action to take at time t, helps to know about previous time steps.
- This is known as a *frame history length* parameter.
  - Not to be confused with *frame skip*, or how many frames an action is applied in the emulator…

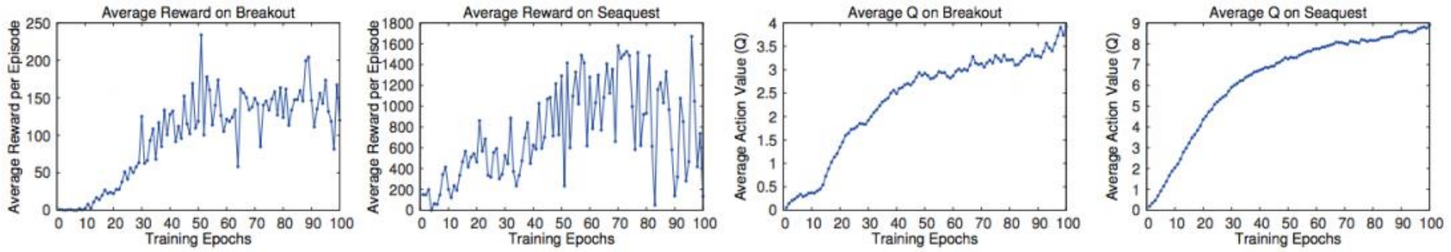
The Q-Network Architecture



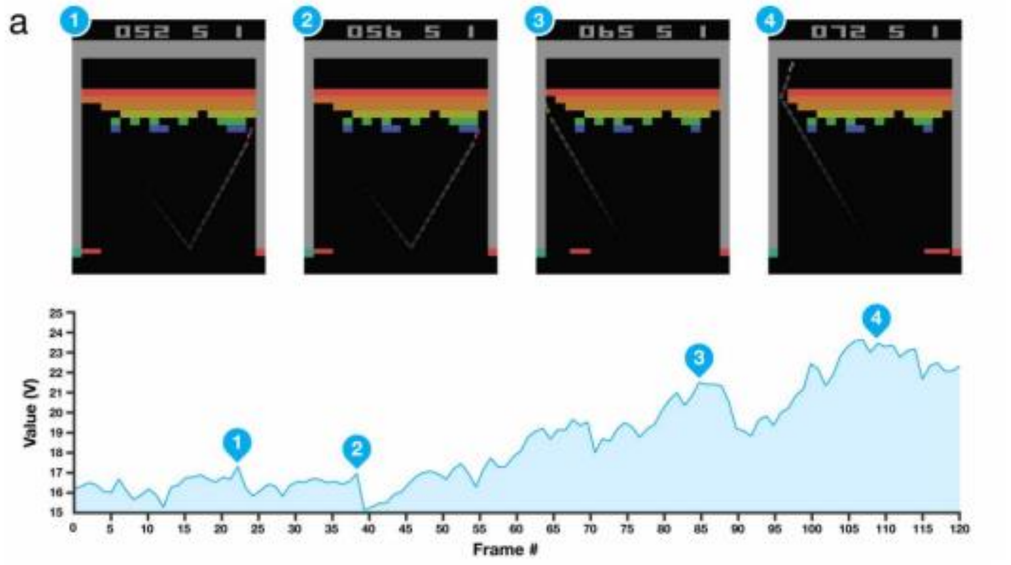
- Q-network maps a data block of size (B x 84 x 84 x 4) to the batch of Q(s,a) values for all possible actions.
  - Means just one forward pass is needed to get Q-values for all actions.
  - Then policy chooses whatever action index maximizes Q-value (unless using random action).

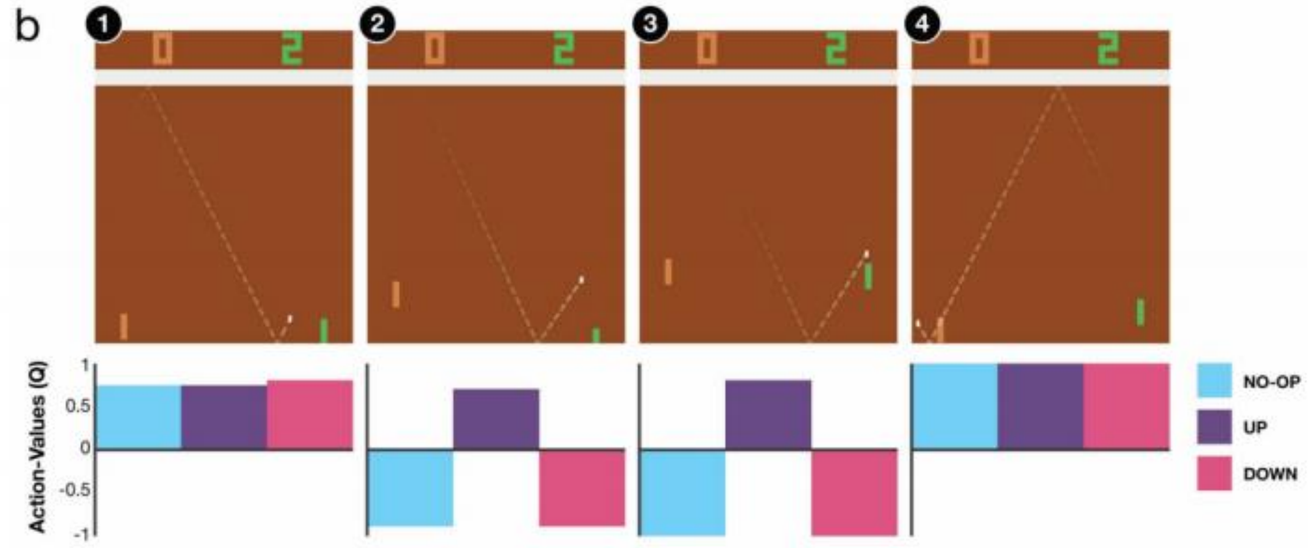
Unfortunately misleading: there are actually three convolution layers (not two).

Are the Q-values accurate?

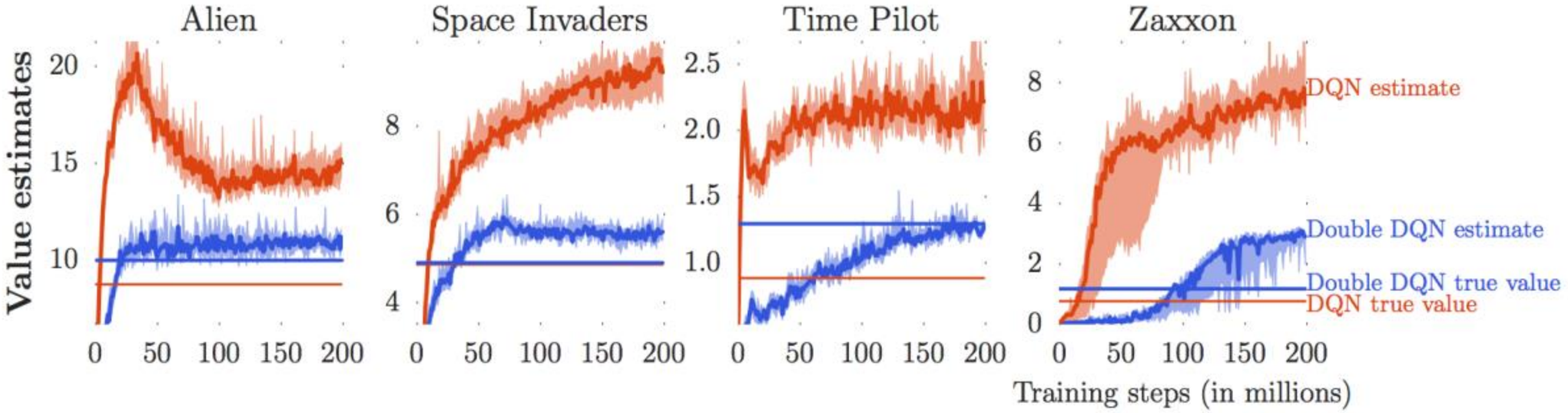


As predicted Q increases, so does the return





Are the Q-values accurate?



- (Ignore the blue curves for now, just look at the "DQN" stuff.)
- Straight horizontal orange curves show — *after training* — average actual discounted return at each state. If no bias, they would match the learning curves at the right side of the plots.

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

value *also* comes from $Q_{\phi'}$ action selected according to $Q_{\phi'}$

(Follows from Jensen's inequality)

Double Q-learning

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

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

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:

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

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

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!

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

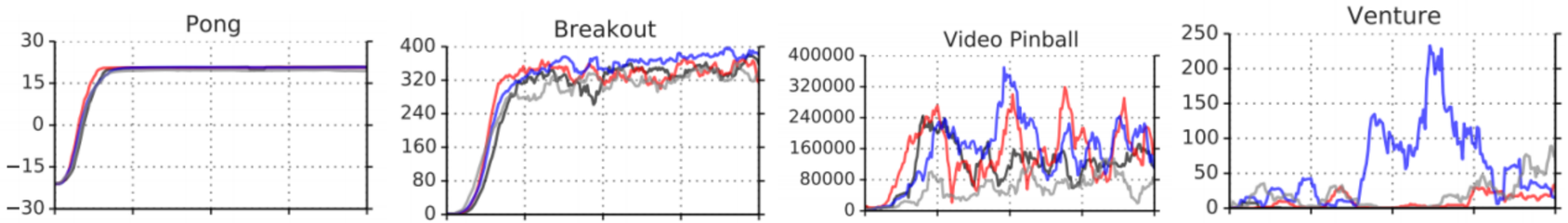


Figure: From T. Schaul, J. Quan, I. Antonoglou, and D. Silver. "Prioritized experience replay". *arXiv preprint arXiv:1511.05952* (2015), Figure 7

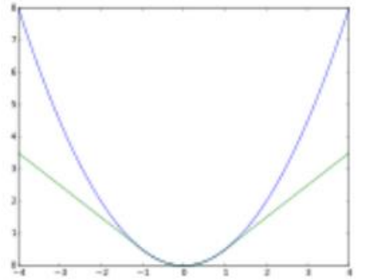
- Large replay buffers help improve stability
  - Looks more like fitted Q-iteration
- It takes time, be patient – might be no better than random for a while
- Start with high exploration (epsilon) and gradually reduce

Slide partly borrowed from J. Schulman

Advanced tips for Q-learning

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

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

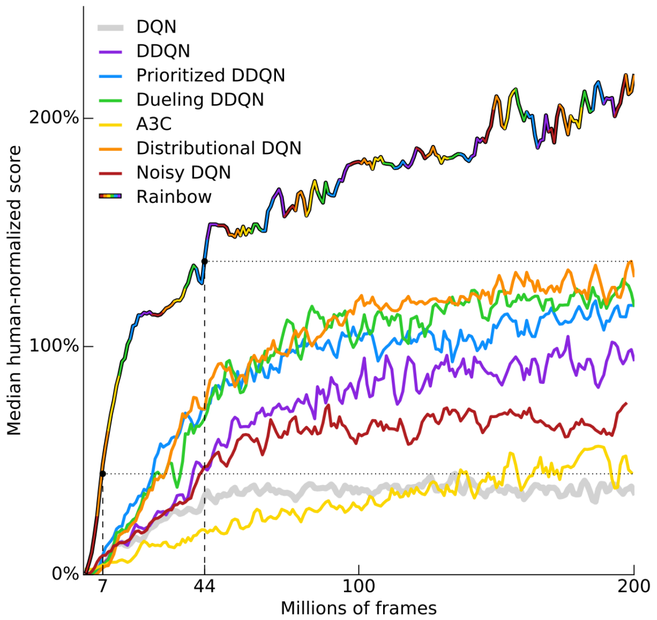


- Double Q-learning helps *a lot* in practice, simple and no downsides
- N-step returns also help a lot, but have some downsides
- Schedule exploration (high to low) and learning rates (high to low), Adam optimizer can help too
- Run multiple random seeds, it's very inconsistent between runs

Slide partly borrowed from J. Schulman

Upgrades to (D)DQN

- Prioritized Experience Replay [1]
  - Rather than sample uniformly at random, sample according to TD error!
- Dueling Architectures [2]
  - Separate state value and advantage value streams.
- Rainbow [3]
  - Combine a bunch of improvements together!



[1]: Schaul et al., *Prioritized Experience Replay*, ICLR 2016.
[2]: Wang et al., *Dueling Network Architectures for Deep Reinforcement Learning*, ICML 2016.
[3]: Hessel et al., *Rainbow: Combining Improvements in Deep Reinforcement Learning*, AAAI 2018.

Summary

- Value Functions
- Actor-Critic and Q-Learning
- Deep Q-Network (DQN)
- Double DQN (DDQN)
- Practical Tips

Next time: more advanced exploration methods!