Computational Sensorimotor Learning (Spring'21)

Pulkit Agrawal

Lecture 4

Feb 25 2021

Value Iteration

$$V^{\pi^{k+1}}(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V^{\pi^k}(s') \right]$$

Q-Function

$$Q(s_{t_0}, a_{t_0}) = r(s_{t_0}, a_{t_0}) + \gamma E_\tau \left[\sum_{t=t_0+1} \gamma^{t-(t_0+1)} r_t \right]$$

Q-Value Iteration

$$Q^{\pi^{k+1}}(s,a) = R(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) \max_{a'} Q^{\pi^k}(s',a')$$

Value Iteration

$$V^{\pi^{k+1}}(s) = \max_{a \in A} \left[R(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) V^{\pi^k}(s') \right]$$

Q-Function

$$Q(s_{t_0}, a_{t_0}) = r(s_{t_0}, a_{t_0}) + \gamma E_\tau \left[\sum_{t=t_0+1} \gamma^{t-(t_0+1)} r_t \right]$$

Q-Value Iteration

$$Q^{\pi^{k+1}}(s,a) = R(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) \max_{a'} Q^{\pi^k}(s',a')$$

No dependence on Policy: Off-Policy Learning

Q-Value Iteration

$$Q^t(s,a) = R(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) \max_{a'} Q^{t-1}(s',a')$$

updated value

previous iteration

Optimum Policy

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

Q-Value Iteration

$$Q^t(s,a) = R(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) \max_{a'} Q^{t-1}(s',a')$$

What is the practical challenge
in using Q-value iteration?

Must estimate from **data**

$$\tau = (s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_{t-1}, a_{t-1}, r_{t-1}, s_t)$$

Q-Value Updates from data

$$Q^t(s,a) = R(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) \max_{a'} Q^{t-1}(s',a')$$

Transition Data: (s, a, r, s') $\quad max_{a'} Q^{t-1}(s,a')$

compute target for $Q^t(s,a)$: $\quad y_t = r + \max_{a'} Q^{t-1}(s',a')$

compute error $\quad e_t = y_t - Q^{t-1}(s,a)$

Update Q-value $\quad Q^t(s,a) \leftarrow Q^{t-1}(s,a) + \alpha e_t$

keep it small

Q-Value Updates from data

compute target for $Q^t(s,a)$: $y_t = r + \max_{a'} Q^{t-1}(s',a')$

compute error $e_t = y_t - Q^{t-1}(s,a)$

Update Q-value $Q^t(s,a) \leftarrow Q^{t-1}(s,a) + \alpha e_t$

$$Q^t(s,a) = Q^{t-1}(s,a) + \alpha\left(r + \max_{a'} Q^{t-1}(s',a') - Q^{t-1}(s,a)\right)$$

Q-Learning

how to initialize Q-values?

(large values)

$$\pi(s) = \arg\max_a Q^{t-1}(s,a)$$

(Greedy)

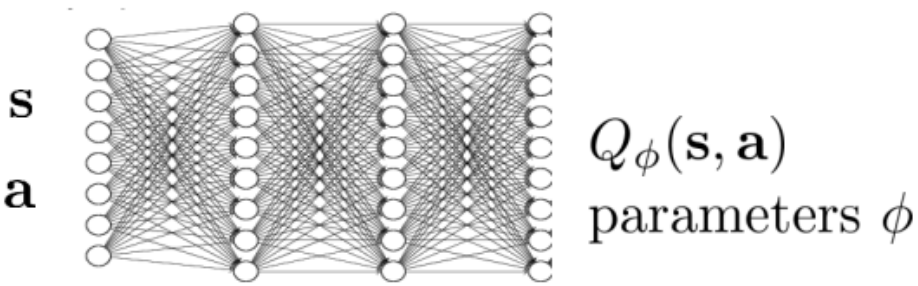
Moving to non-tabular case

Dataset: $\left\{ \left(s_i, a_i, r(s_i, a_i), s_i'\right) \;;\; i \in [1,N] \right\}$

(get targets)

(targets with improved Q)

(update parameters)



(slide adapted from Sergey Levine's Lectures)

Q-learning is off-policy

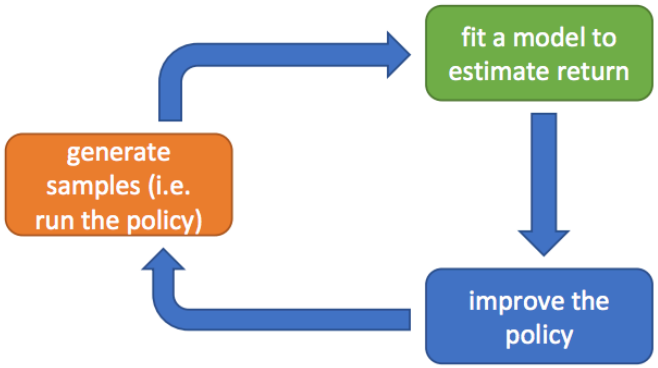
full fitted Q-iteration algorithm:

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

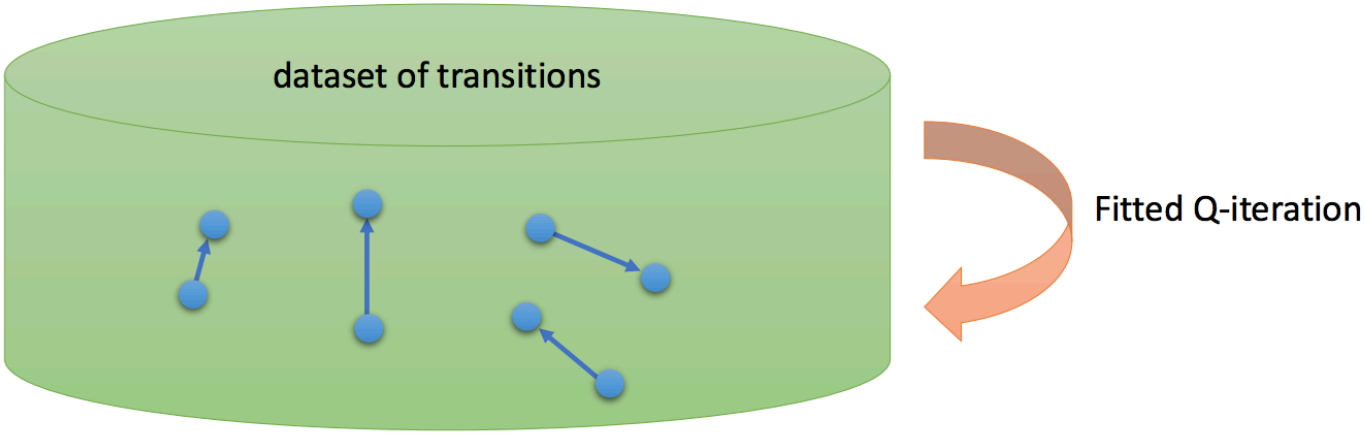
$K \times$

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

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



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



(slide adapted from Sergey Levine's Lectures)

Training (online v/s batch)

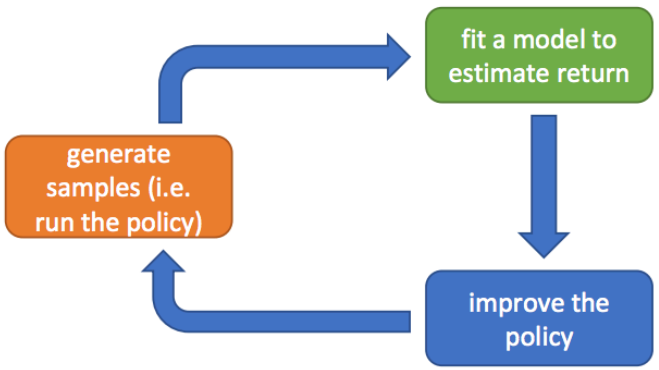
full fitted Q-iteration algorithm:

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

($K \times$)

impractical for a large dataset

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



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

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

use stochastic gradient descent!

(slide adapted from Sergey Levine's Lectures)

Is Q-learning gradient descent?

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

isn't this just gradient descent? that converges, right?

(slide adapted from Sergey Levine's Lectures)

Is Q-learning gradient descent?

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

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

(slide adapted from Sergey Levine's Lectures)

Policy for training and evaluation

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:

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

is this a good policy for training?

(slide adapted from Sergey Levine's Lectures)

Policy for training and evaluation

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:

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

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

“epsilon-greedy”

(slide adapted from Sergey Levine’s Lectures)

Policy for training and evaluation

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:

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

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

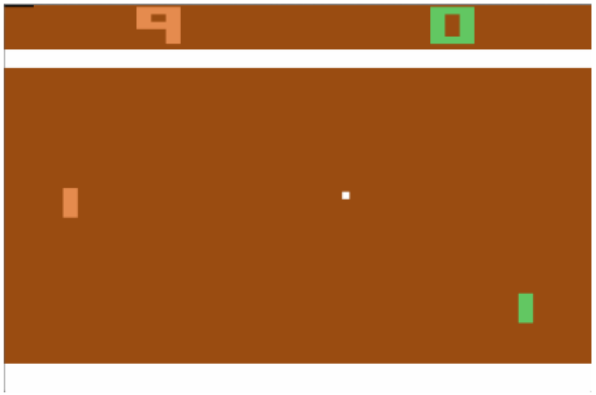
"epsilon-greedy"

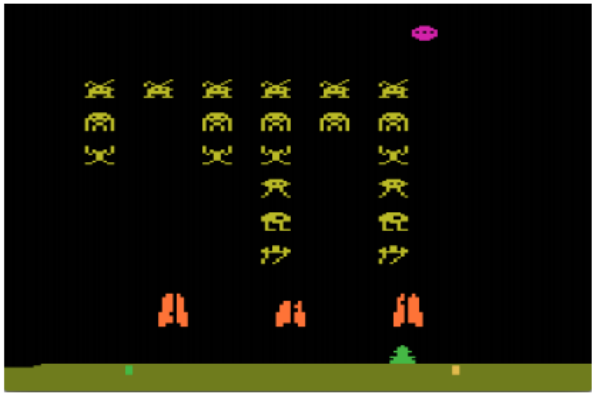
$$\pi(\mathbf{a}_t|\mathbf{s}_t) \propto \exp(Q_\phi(\mathbf{s}_t, \mathbf{a}_t))$$

"Boltzmann exploration"

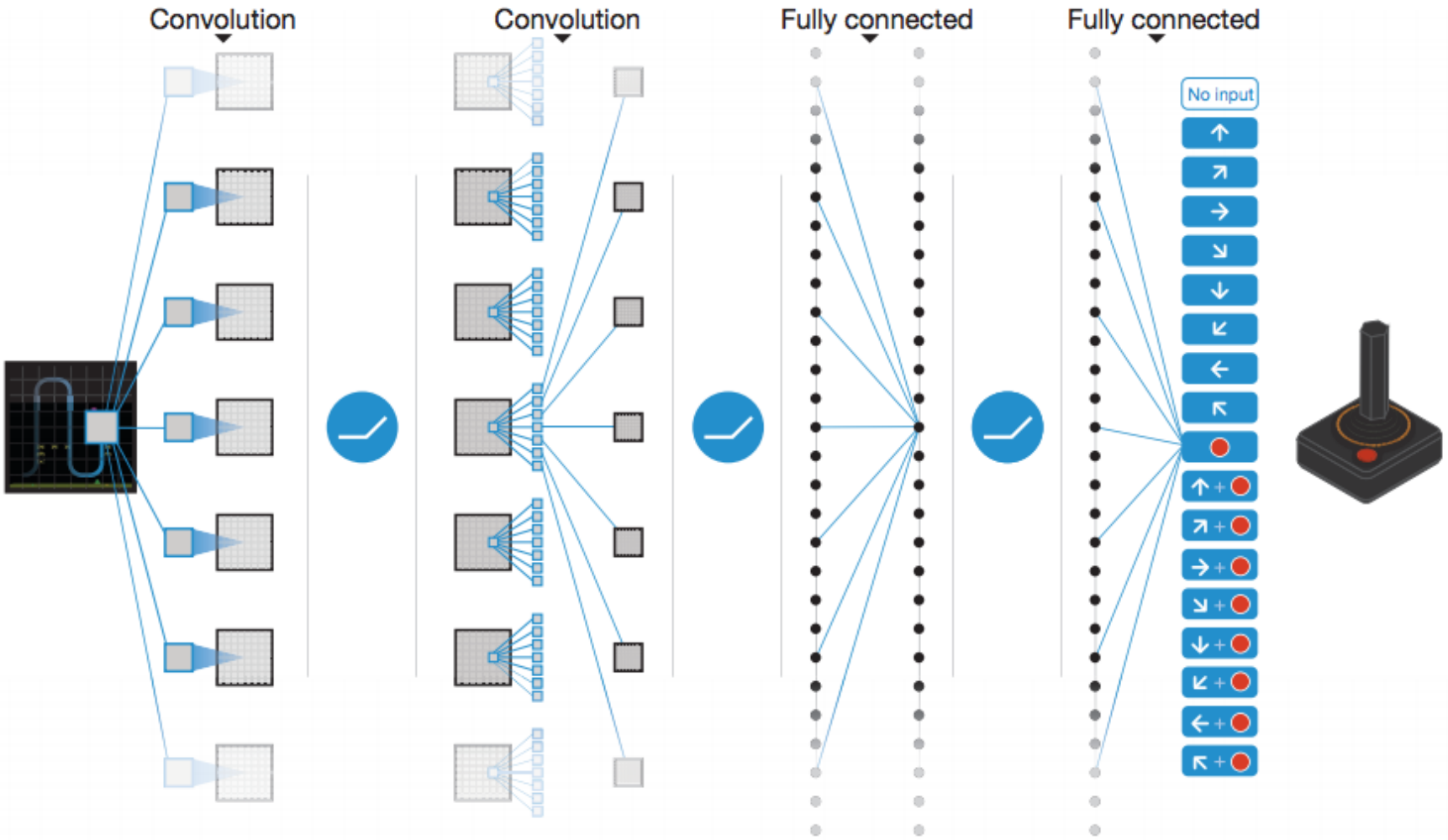
(slide adapted from Sergey Levine's Lectures)

Q-Learning applied to ATARI Games

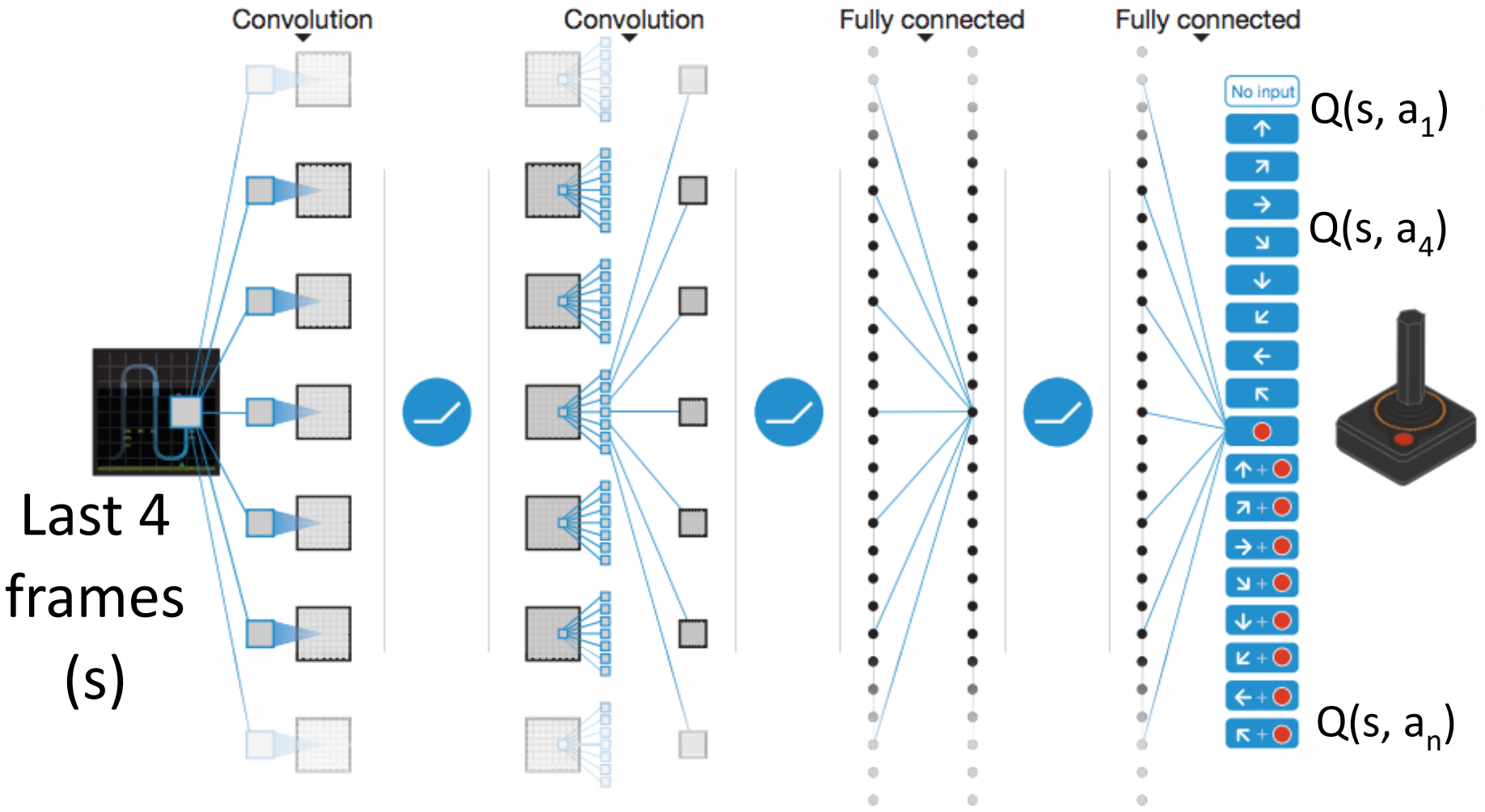




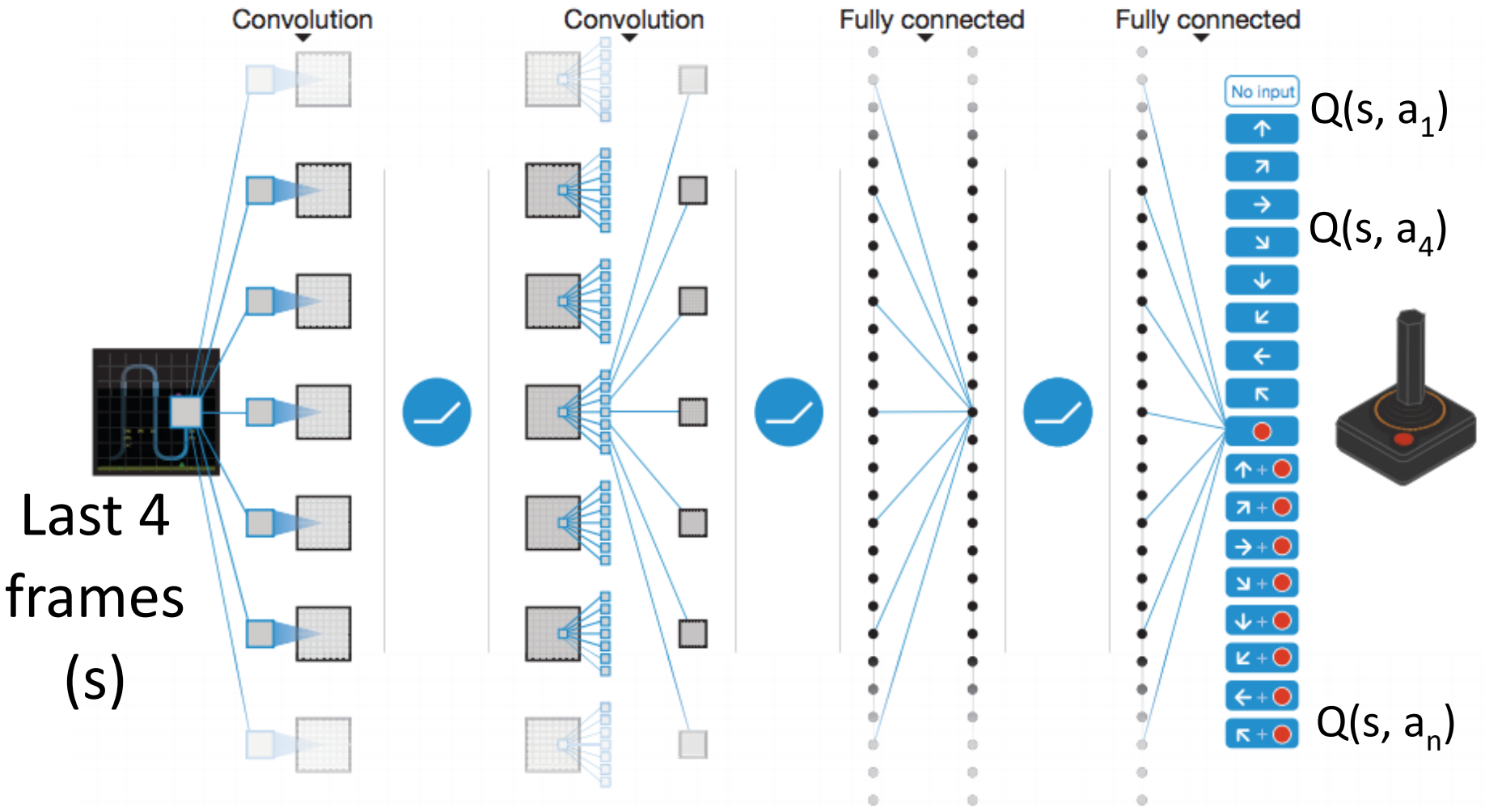
Learning Q-Values



Learning Q-Values

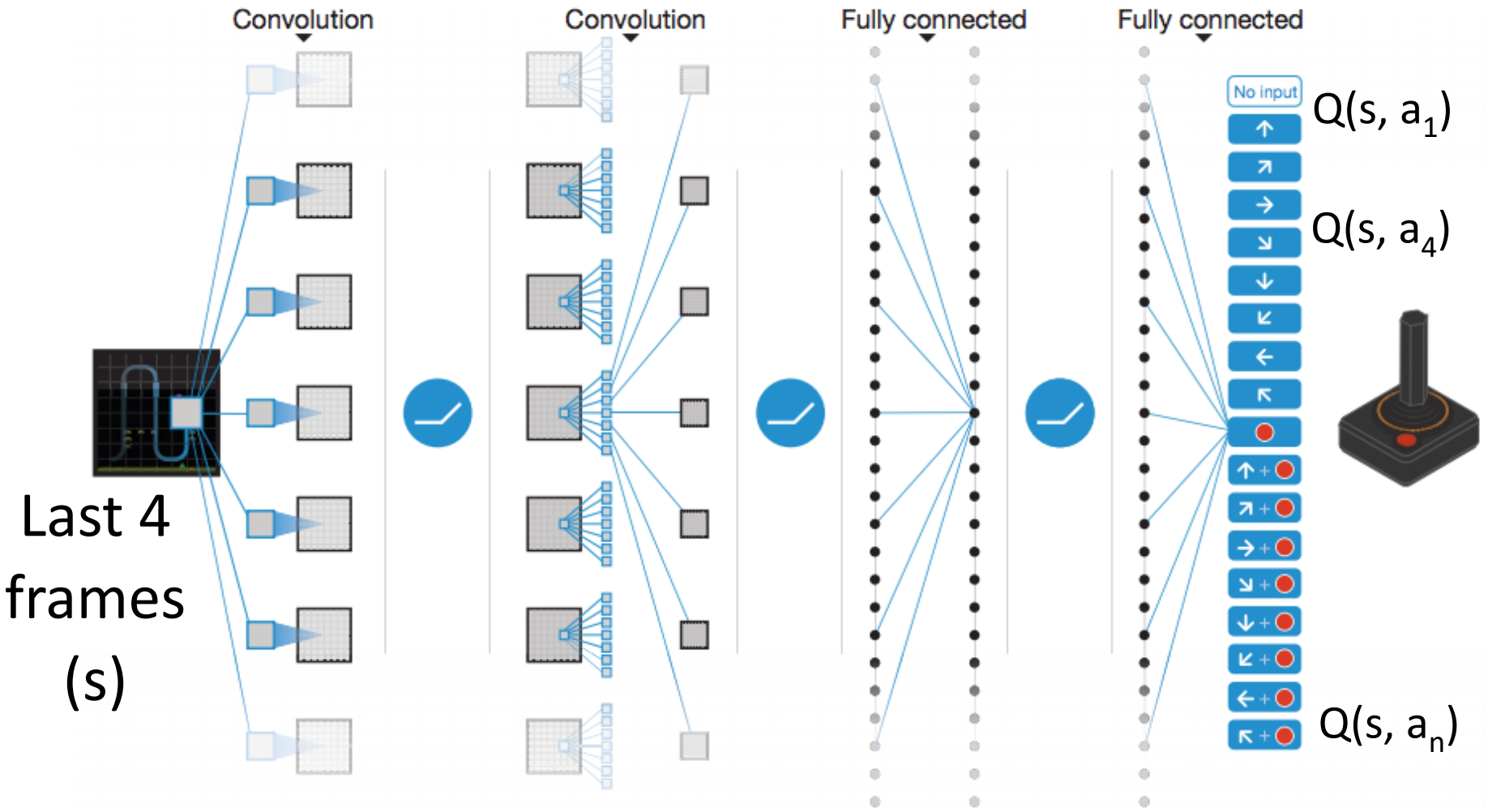


Learning Q-Values



With probability $(1 - \varepsilon)$ → execute $\max_a Q(s,a)$

Learning Q-Values

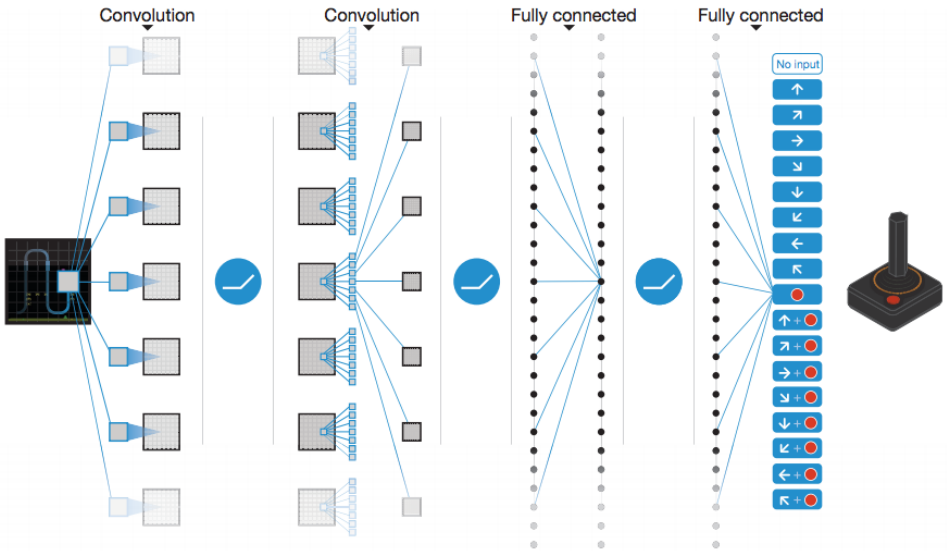


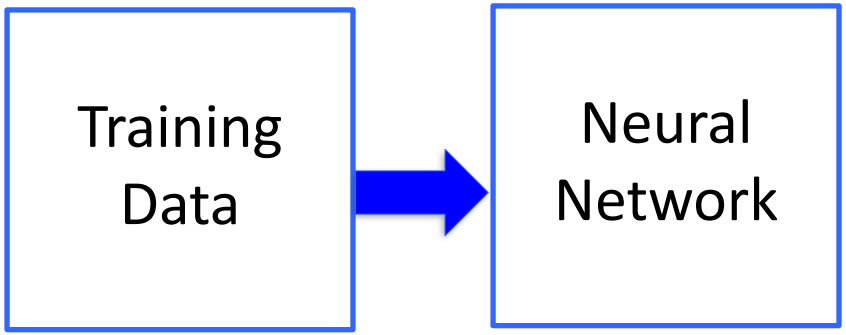
With probability $(1 - \varepsilon) \rightarrow$ execute $\max_a Q(s,a)$

With probability $\varepsilon \rightarrow$ execute random action

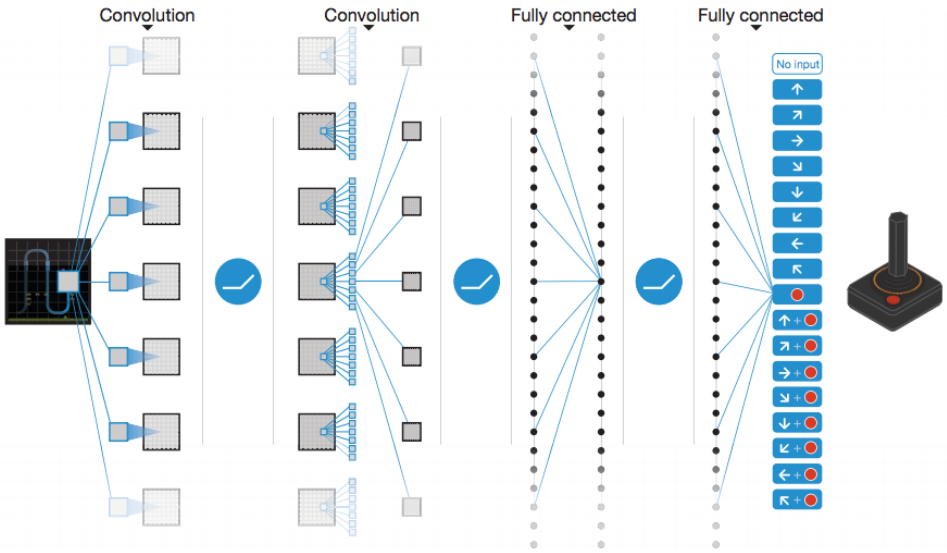
Straightforward application of Q-Learning doesnot work

Why?





Supervised Learning

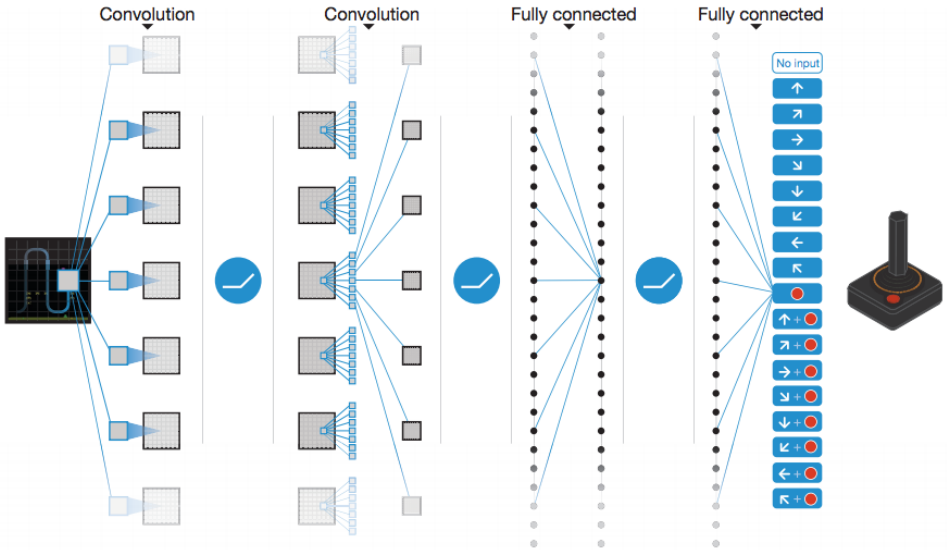


Training Data → Neural Network

Supervised Learning

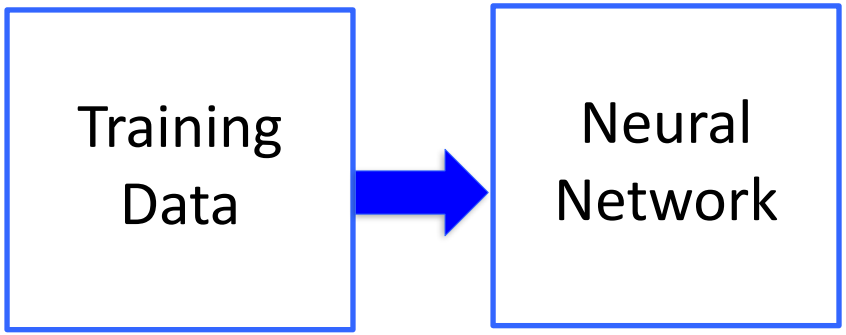
Training Data → Neural Network

Reinforcement Learning

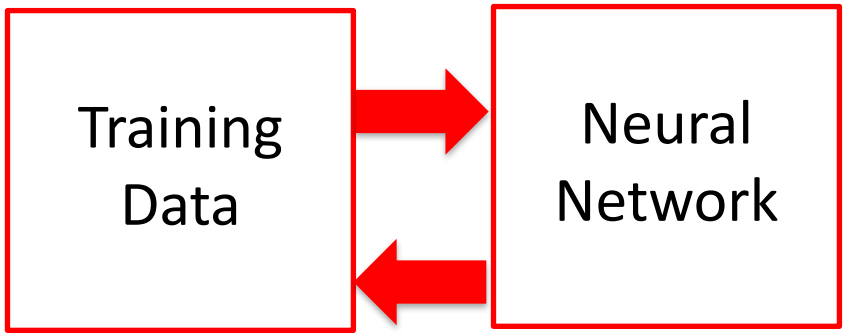


Stuck in bad local minima → Training data collected near this minima

Vicious Cycle ☹



Supervised Learning



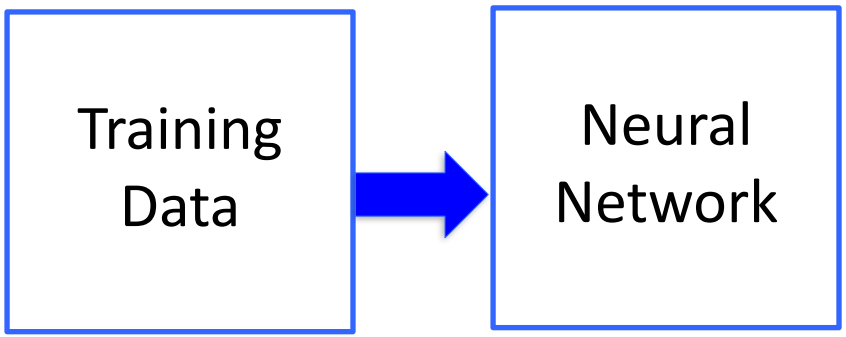
Reinforcement Learning

How to Overcome this problem?

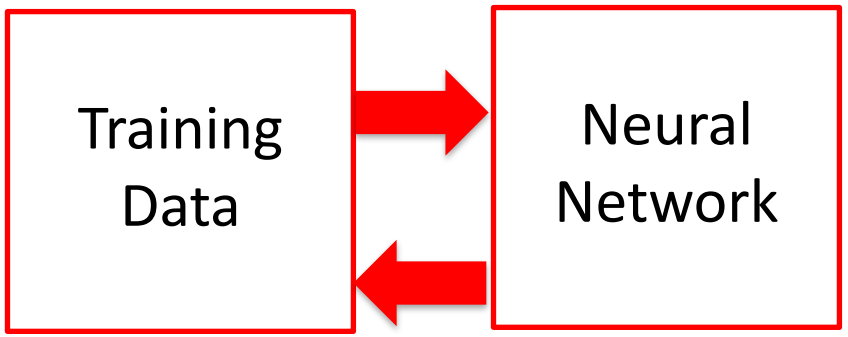
Maintain data-diversity!

Stuck in bad local minima → Training data collected near this minima

Vicious Cycle ☹



Supervised Learning



Reinforcement Learning

One Solution

Memory of Past Experience

s_1, a_1, r_1, s_2

s_2, a_2, r_2, s_3

.

.

.

s_T, a_T, r_T, s_{T+1}

(some near local minima, some possibly away)

One Solution

Memory of Past Experience

s_1, a_1, r_1, s_2

s_2, a_2, r_2, s_3

.

.

.

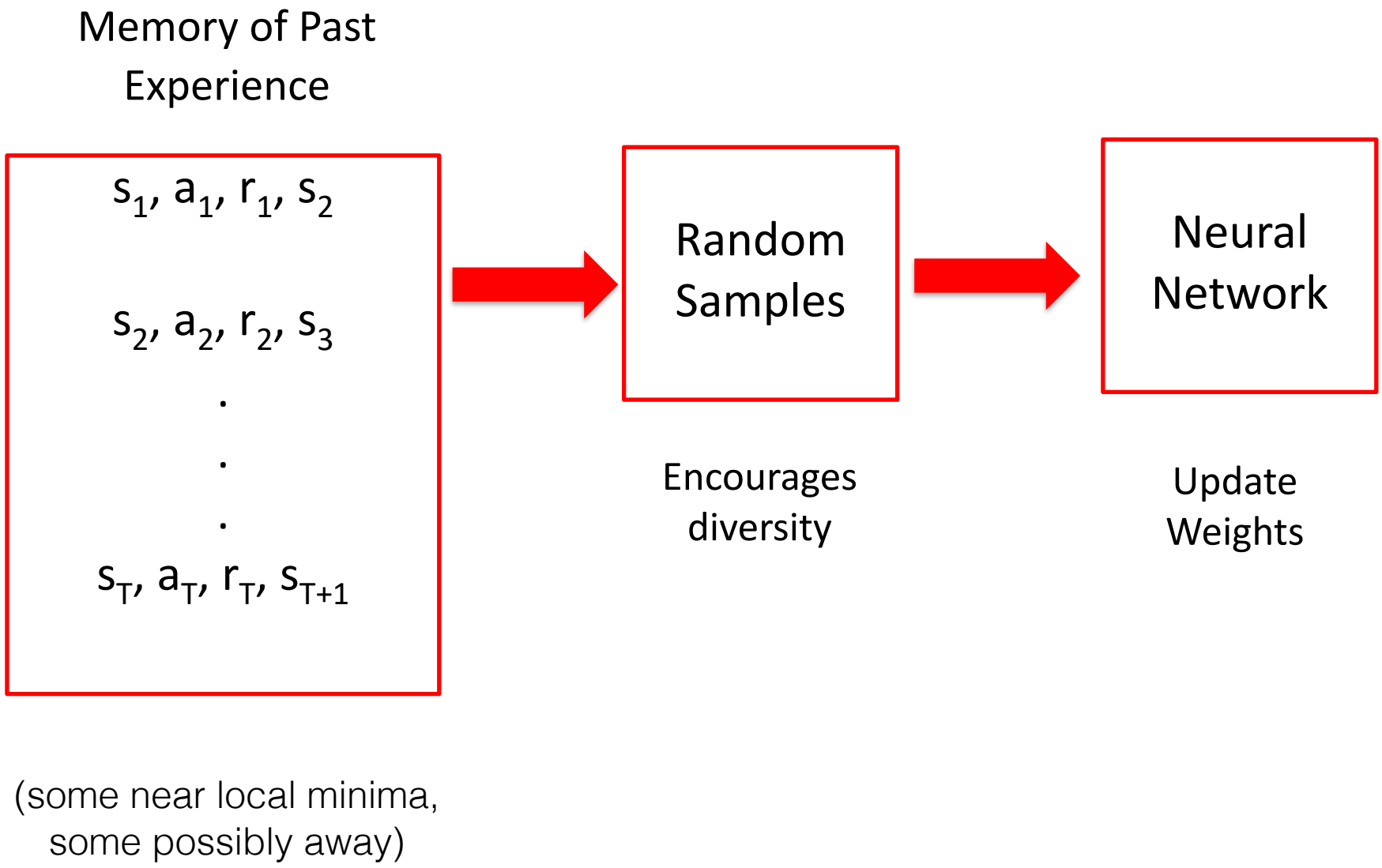
s_T, a_T, r_T, s_{T+1}

Random Samples

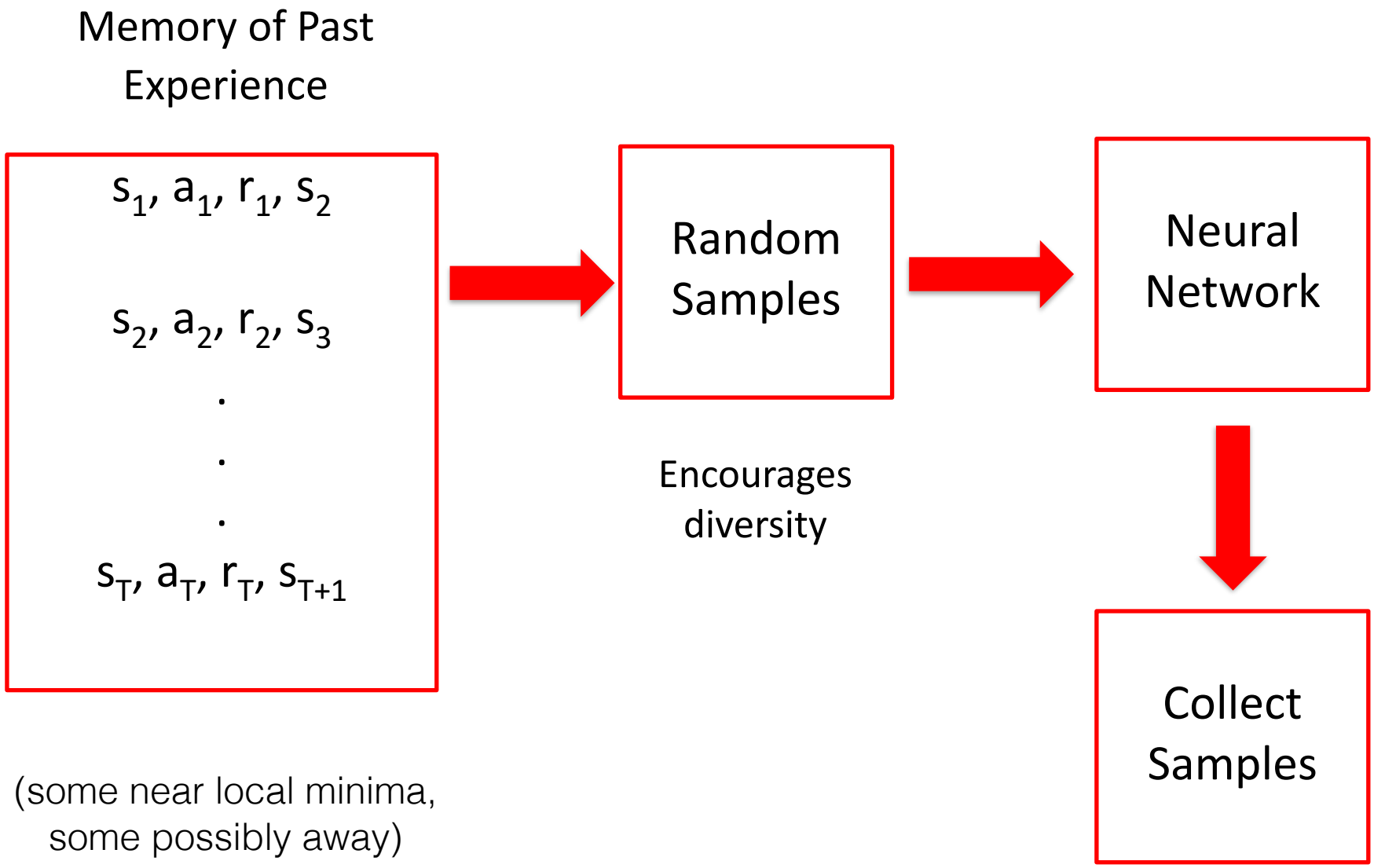
Encourages diversity

(some near local minima, some possibly away)

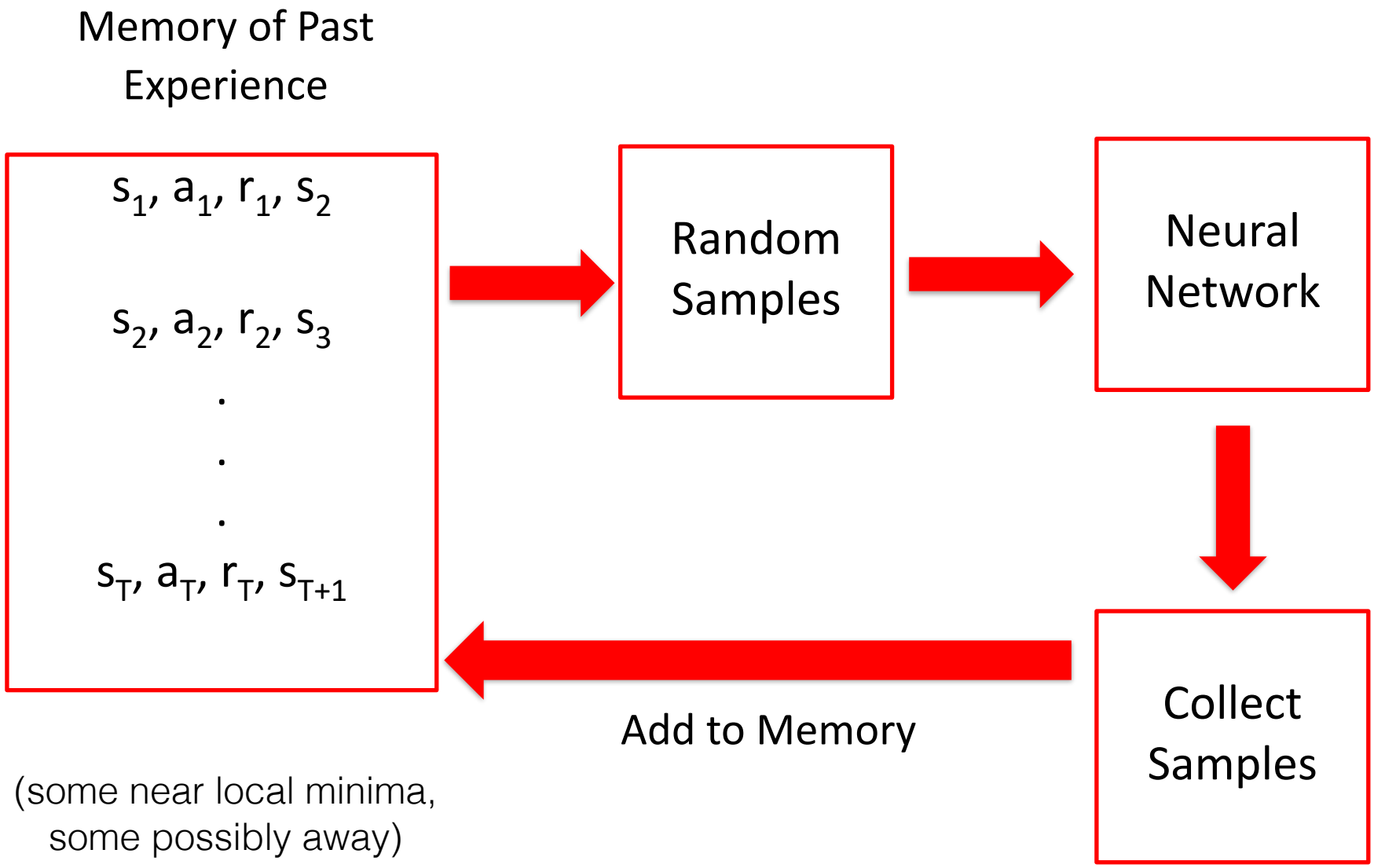
One Solution



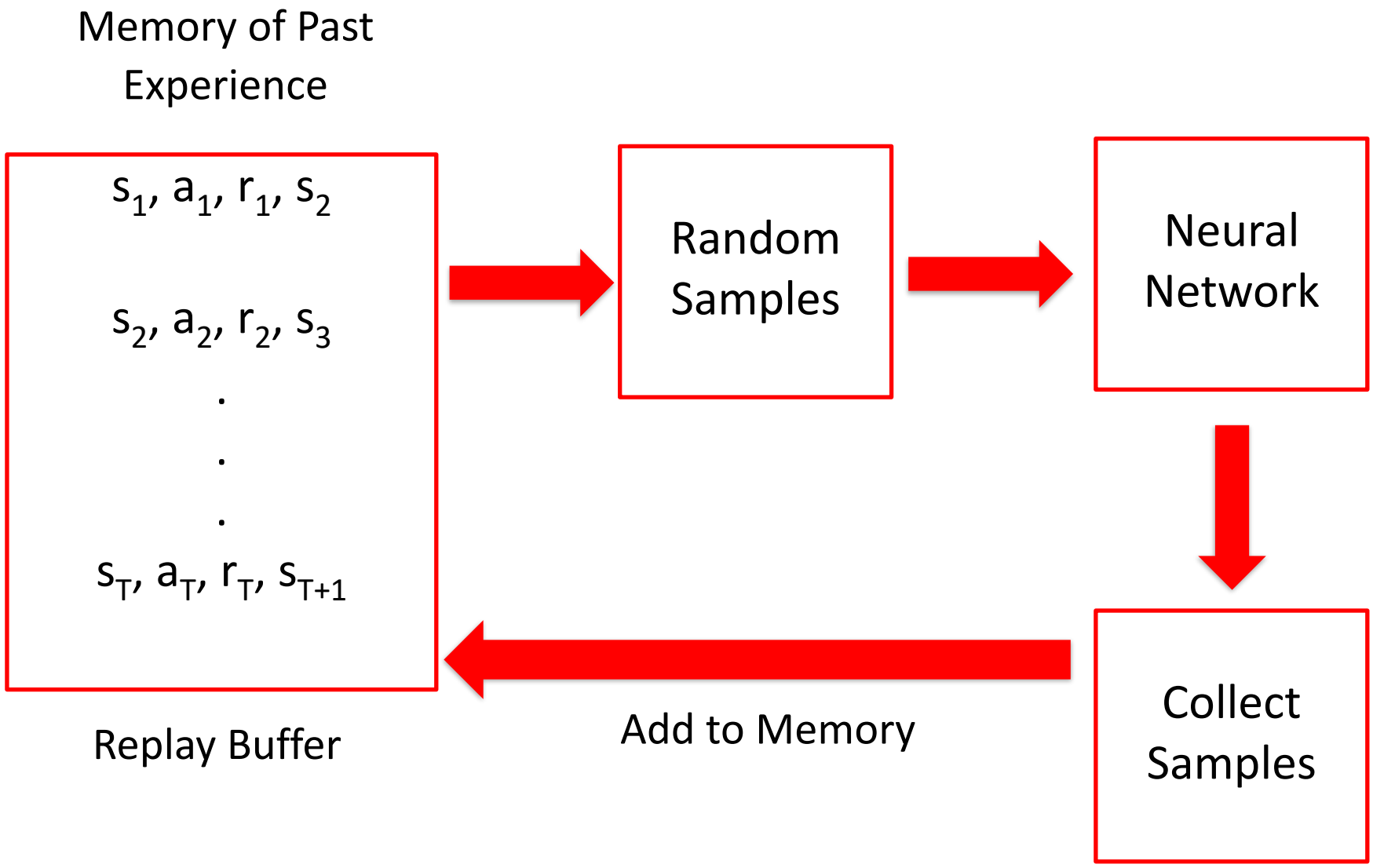
One Solution



One Solution



One Solution



Using Replay buffer still poses challenges

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

(slide adapted from Sergey Levine's Lectures)

Using Replay buffer still poses challenges

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

one gradient step, moving target

If (s, a, s') is sampled in "two" batches target Q-values will be different!

(slide adapted from Sergey Levine's Lectures)

Using Replay buffer still poses challenges

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

one gradient step, moving target

If (s, a, s') is sampled in "two" batches target Q-values will be different!

fitted Q-iteration

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

No batches here!

perfectly well-defined, stable regression

(slide adapted from Sergey Levine's Lectures)

Idea of target networks

$Q_\phi(s, a)$

Q-Network

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

Target Q-Network

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

$N\times$ $K\times$

supervised regression

targets don't change in inner loop!

(slide adapted from Sergey Levine's Lectures)

Idea of target networks

$Q_\phi(s,a)$

Q-Network

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

Target Q-Network

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

$N\times$ $K\times$

targets don't change in inner loop!

supervised regression

(slide adapted from Sergey Levine's Lectures)

Idea of target networks

$Q_\phi(s, a)$

Q-Network

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

Target Q-Network

Q-learning with replay buffer and target network:

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

$N\times$ $K\times$

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

supervised regression

targets don't change in inner loop!

Target Q-Network is just an **old** version of Q-Network

(slide adapted from Sergey Levine's Lectures)

Idea of target networks

$Q_\phi(s,a)$

Q-Network

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

Target Q-Network

Q-learning with replay buffer and target network:

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

$N\times$ $K\times$

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

targets don't change in inner loop!

supervised regression

Target Q-Network is just an **old** version of Q-Network

Or slow updates

$$\phi' \leftarrow \tau\phi + (1-\tau)\phi' \qquad \tau = 0.001$$

(slide adapted from Sergey Levine's Lectures)

Idea of target networks

$Q_\phi(s, a)$

Q-Network

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

Target Q-Network

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

supervised regression

targets don't change in inner loop!

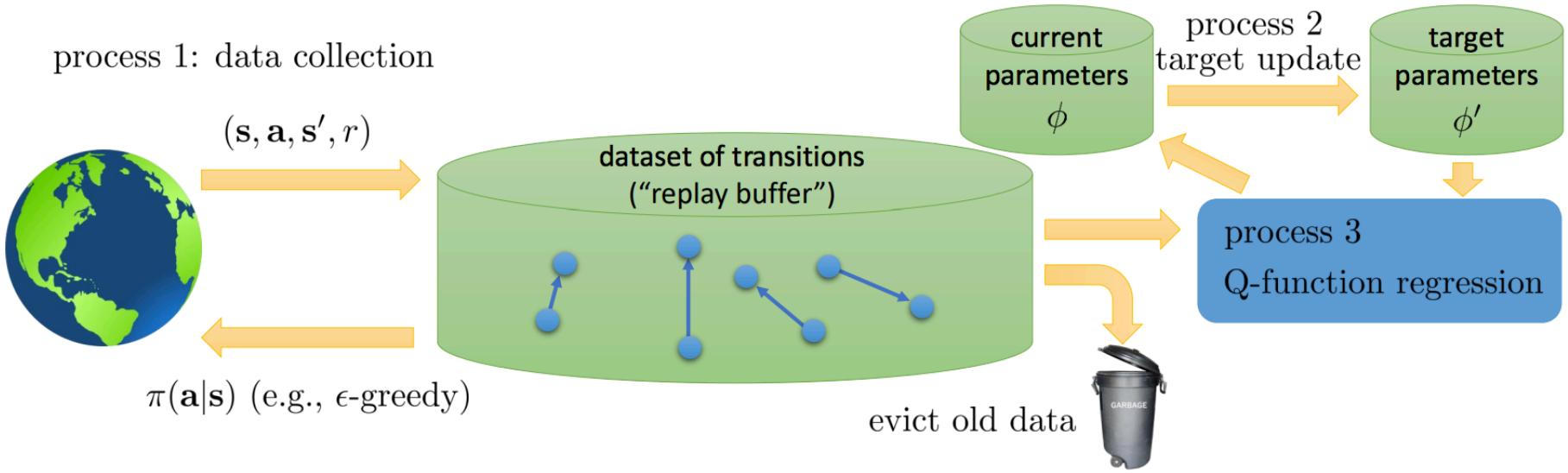
Target Q-Network is just an **old** version of Q-Network

Or slow updates

$$\phi' \leftarrow \tau\phi + (1-\tau)\phi' \qquad \tau = 0.001$$

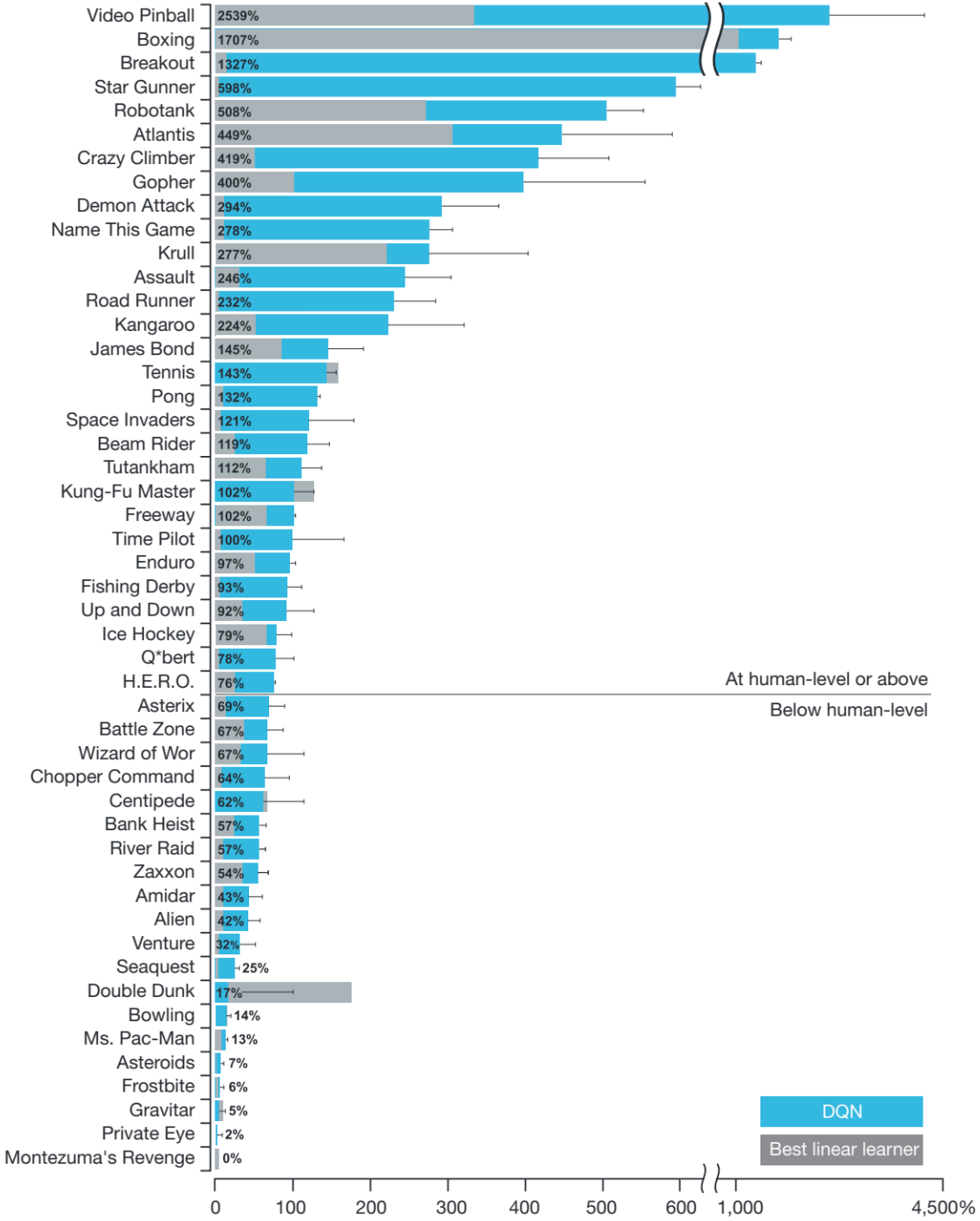
(slide adapted from Sergey Levine's Lectures)

General View of Q-learning

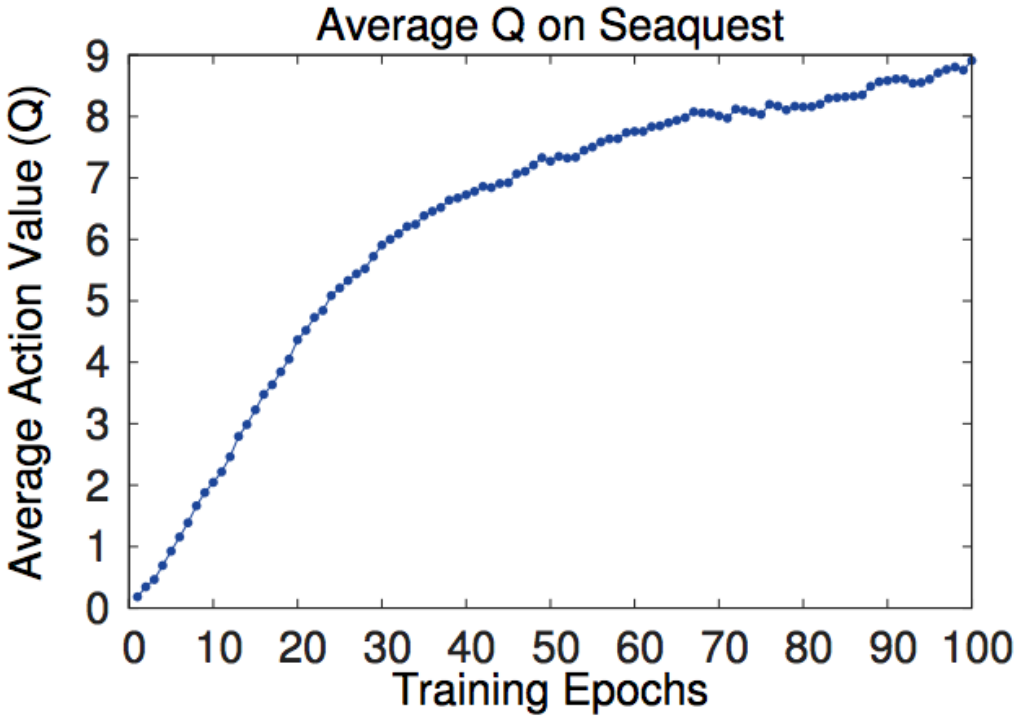


(slide adapted from Sergey Levine's Lectures)

Super-Human Performance



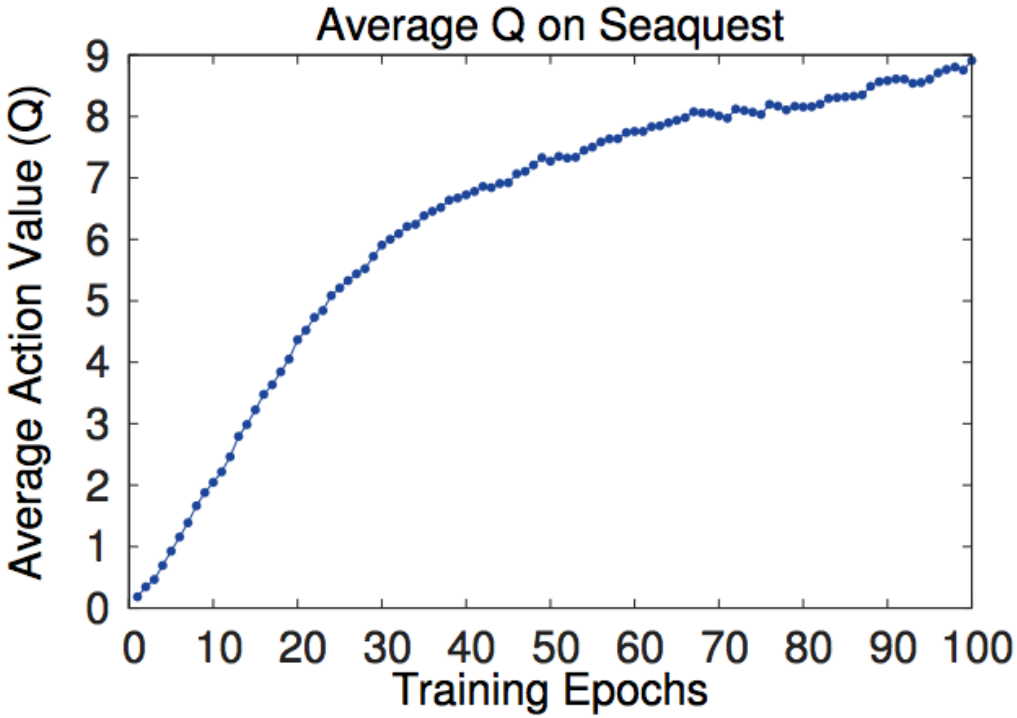
Superhuman performance is great, BUT



Training for ~10M frames

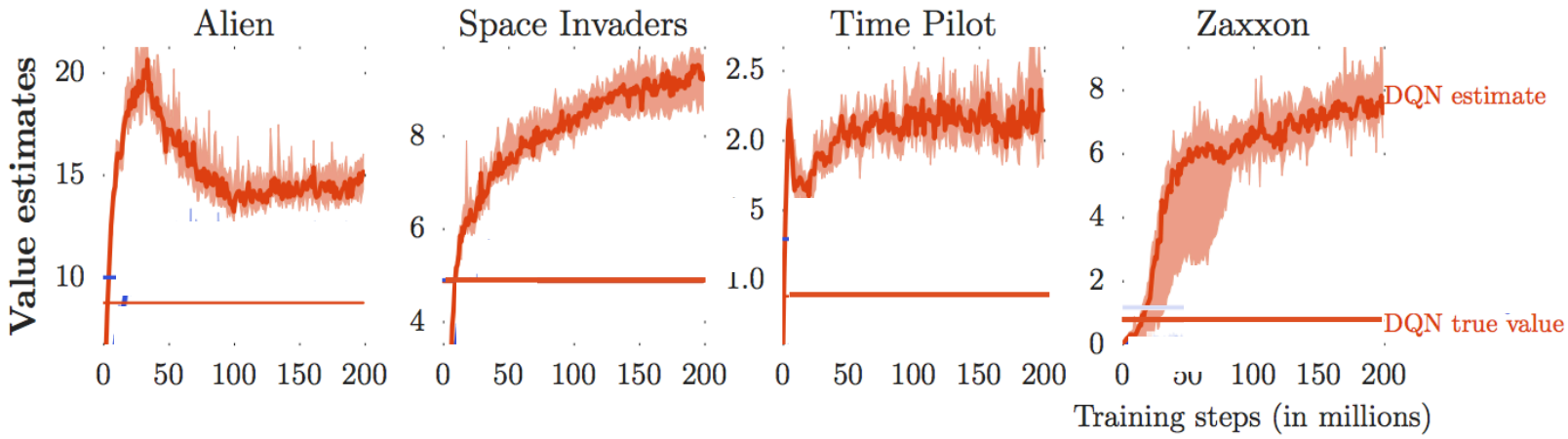
10M interactions on a real robot ….??

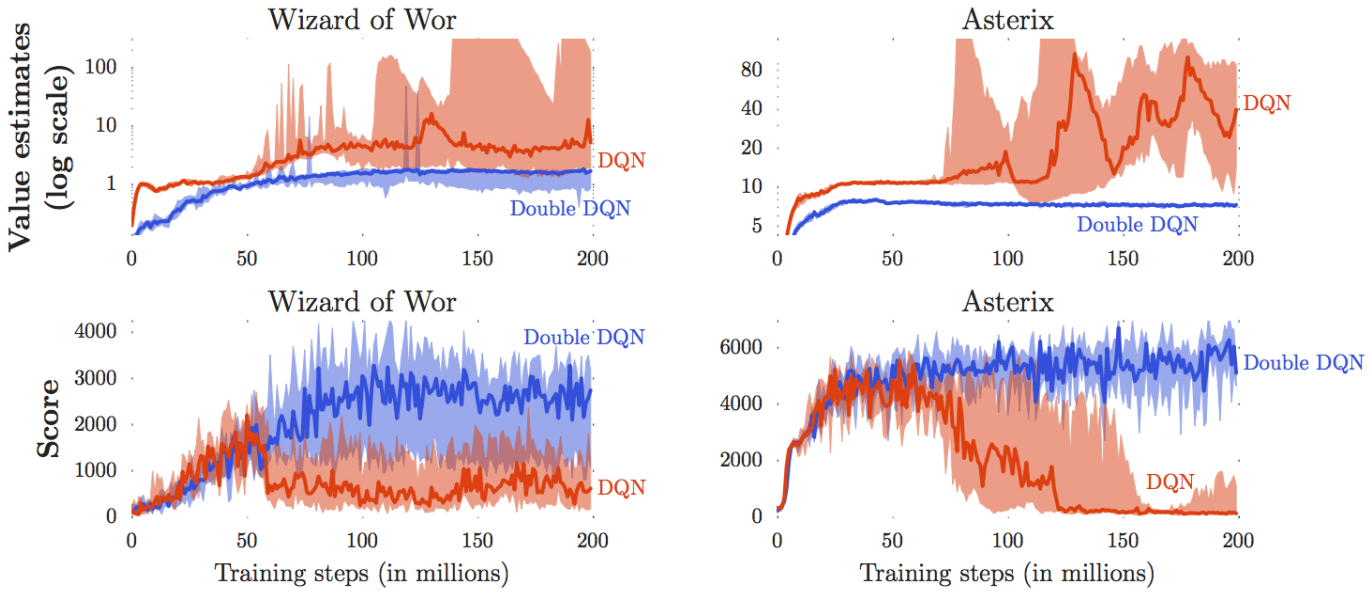
Superhuman performance is great, BUT



Sample inefficient!

How good are the estimated Q-Values?



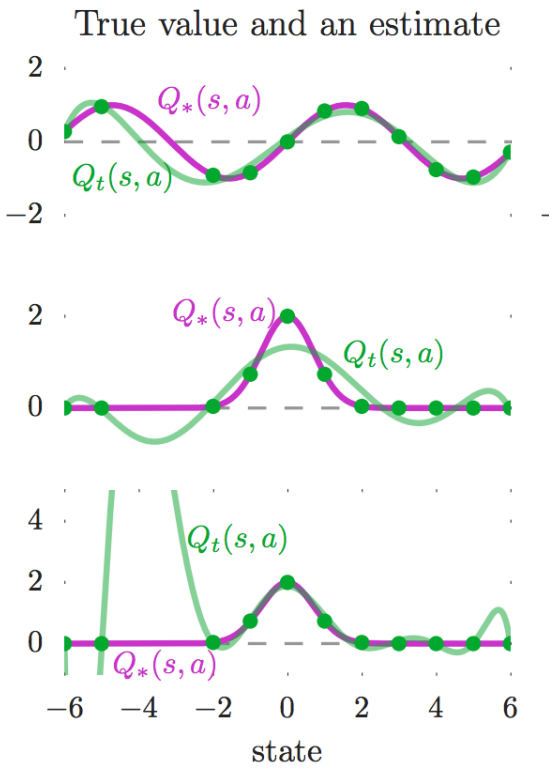


Over-estimation leads to worse Performance

also note the variance

Deep Reinforcement Learning with Double Q-Learning, Hasselt et al., 2015

Why is this the case?



$Q_t(s,a)$: estimated

$Q_*(s,a)$: true value

Deep Reinforcement Learning with Double Q-Learning, Hasselt et al., 2015

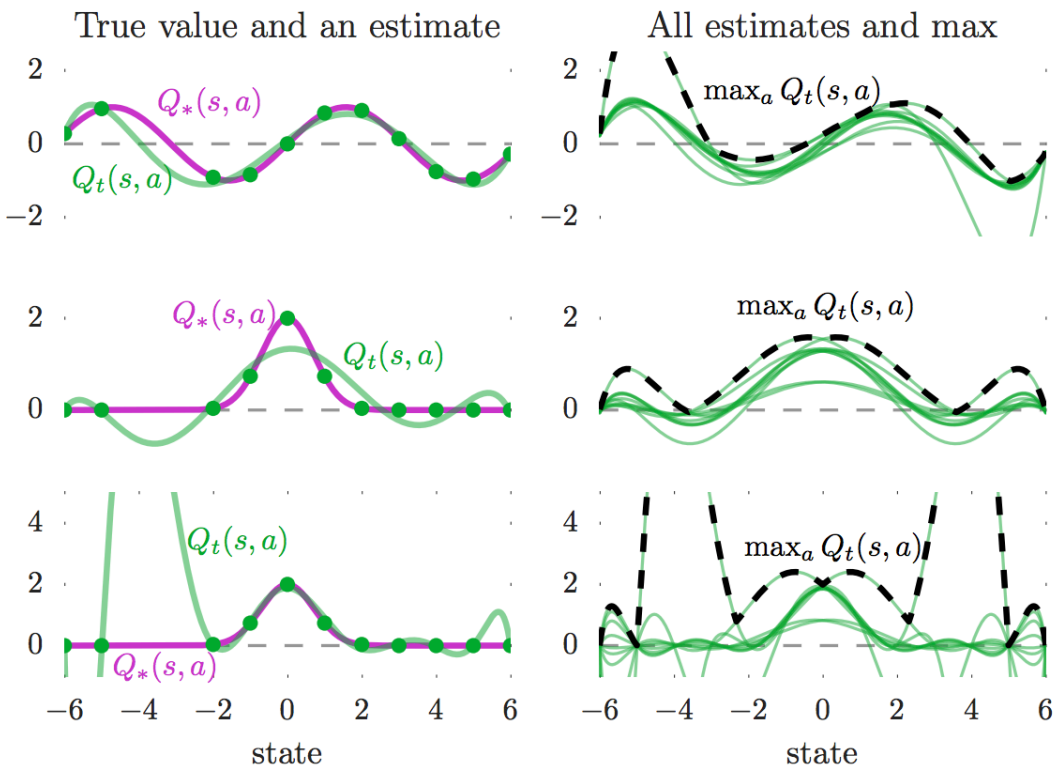
Why is this the case?

$$\hat{Q}(s, a_1) = Q(s, a_1) + \epsilon_1(s)$$

$$\hat{Q}(s, a_2) = Q(s, a_2) + \epsilon_2(s)$$

$$\vdots$$

$$\hat{Q}(s, a_N) = Q(s, a_N) + \epsilon_N(s)$$



$$\max_a \hat{Q}(s, a) = \max_i \left(Q(s, a_i) + \epsilon_i(s) \right)$$

assume $V(s) = Q(s, a_i)\ \forall i$

$$\max_a \hat{Q}(s, a) = V(s) + \max_i(\epsilon_i(s))$$

$$\max_a \hat{Q}(s, a) \geq V(s)$$

$Q_t(s,a)$: estimated

$Q_*(s,a)$: true value

10 actions

each line: $Q(s,a_i)$

Deep Reinforcement Learning with Double Q-Learning, Hasselt et al., 2015

Why is this the case?

$$\max_a \hat{Q}(s,a) = \max_i \left(Q(s,a_i) + \epsilon_i(s) \right)$$

assume $V(s) = Q(s,a_i)\ \forall i$

$$\max_a \hat{Q}(s,a) = V(s) + \max_i(\epsilon_i(s))$$

$$\max_a \hat{Q}(s,a) \geq V(s)$$

if $\Sigma_a \left(\hat{Q}(s,a) - V(s) \right) = 0$ unbiased estimator

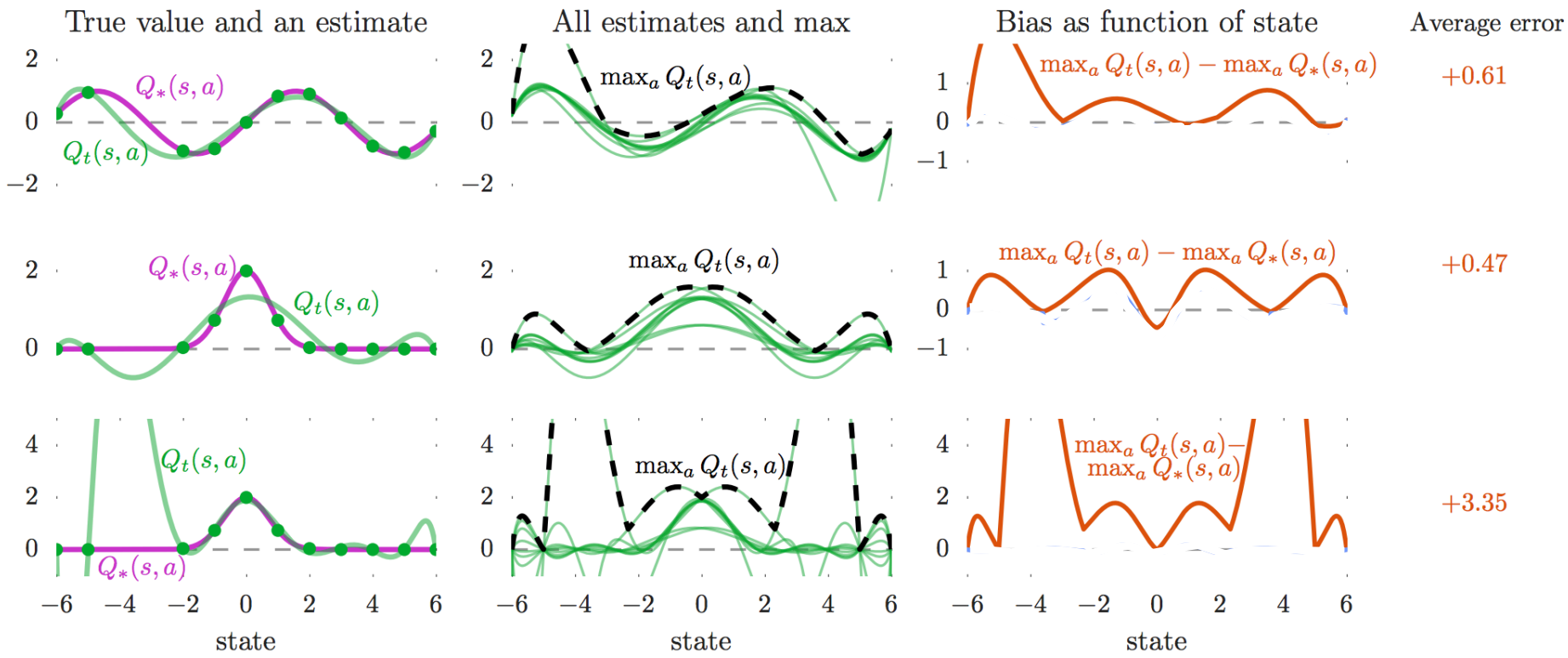
but, $\Sigma_a \left(\hat{Q}(s,a) - V(s) \right)^2 = C$ estimation error

then,

$$\max \hat{Q}(s,a) \geq V(s) + \sqrt{\frac{C}{m-1}}$$ (m: number of actions)

Deep Reinforcement Learning with Double Q-Learning, Hasselt et al., 2015

Why is this the case?



$Q_t(s,a)$: estimated

$Q_*(s,a)$: true value

10 actions

each line: $Q(s,a_i)$

High Errors!!

Deep Reinforcement Learning with Double Q-Learning, Hasselt et al., 2015

Why is this the case?

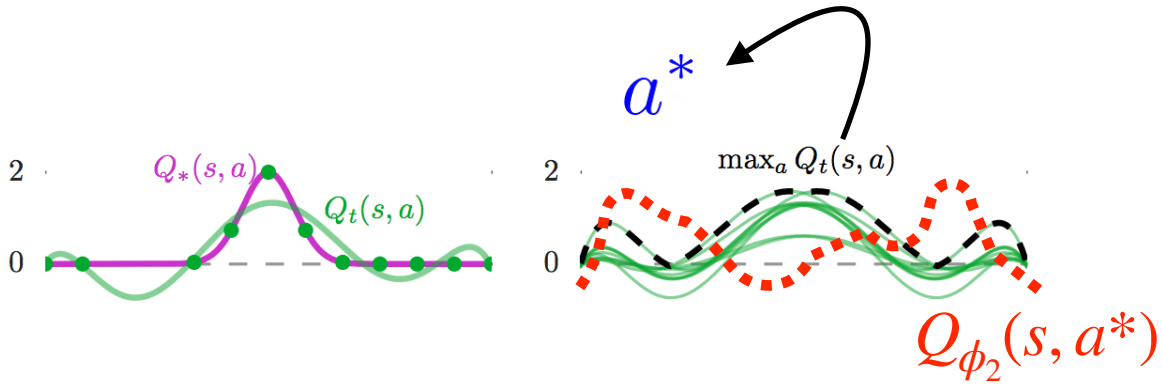
$$\min_{\phi_1} \|y - Q_{\phi_1}(s,a)\|^2$$

$$y = r + \max_a Q_{\phi_1}(s,a)$$

$$Q_{\phi_1}\left(s, \arg\max_a Q_{\phi_1}(s,a)\right)$$

$$Q_{\phi_1}\left(s, a^*_{\phi_1}\right)$$

reinforces overestimation



Two Seperate Predictors: ϕ_1, ϕ_2

$$Q_{\phi_2}\left(s, a^*_{\phi_1}\right)$$

reduces overestimation

$Q_t(s,a)$: estimated

$Q_*(s,a)$: true value

10 actions

each line: $Q(s,a_i)$

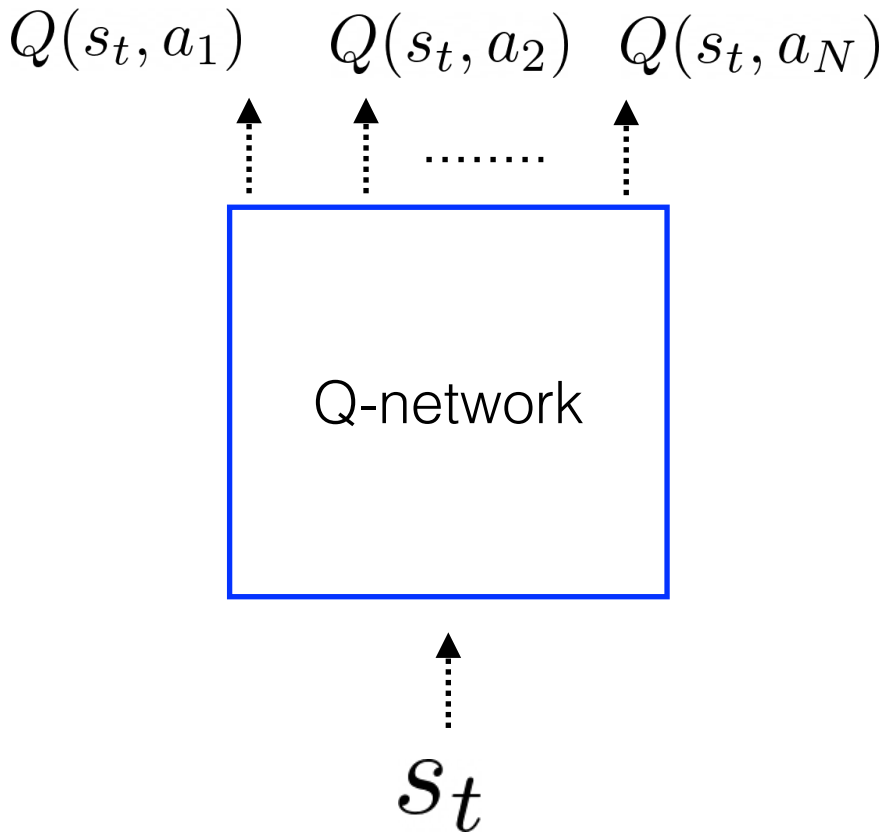
Double Q-Learning (DDQN)

Deep Reinforcement Learning with Double Q-Learning, Hasselt et al., 2015

As the number of actions increase..

most
actions have the same
effect

distinguishing between
actions is important



Learning Q-function for each action separately is wasteful!

Dueling Network Architectures for Deep Reinforcement Learning, Wang et al., 2016

Improving Learning by Better Neural Net Design

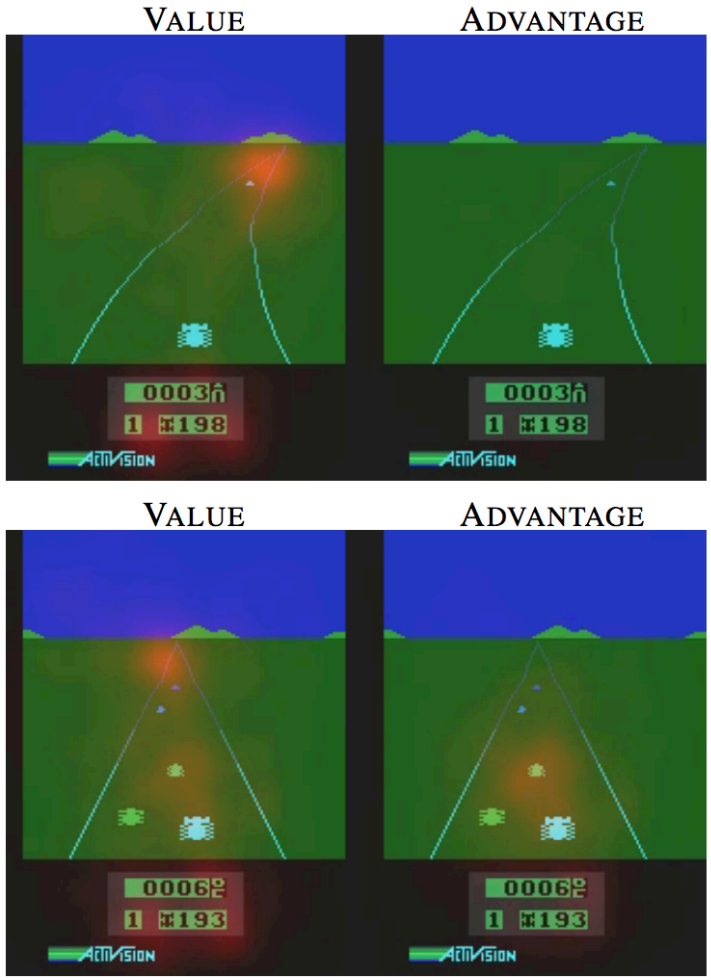


Figure 2. See, attend and drive: Value and advantage saliency maps (red-tinted overlay) on the Atari game Enduro, for a trained dueling architecture. The value stream learns to pay attention to the road. The advantage stream learns to pay attention only when there are cars immediately in front, so as to avoid collisions.

In case of redundant actions

no need to learn Q(s,a)

for each action separately

Separately learn
V(s) and A(s,a) (advantage)

$$Q(s,a) = V(s) + A(s,a)$$

Dueling Network Architectures for Deep Reinforcement Learning, Wang et al., 2016

Dueling Network

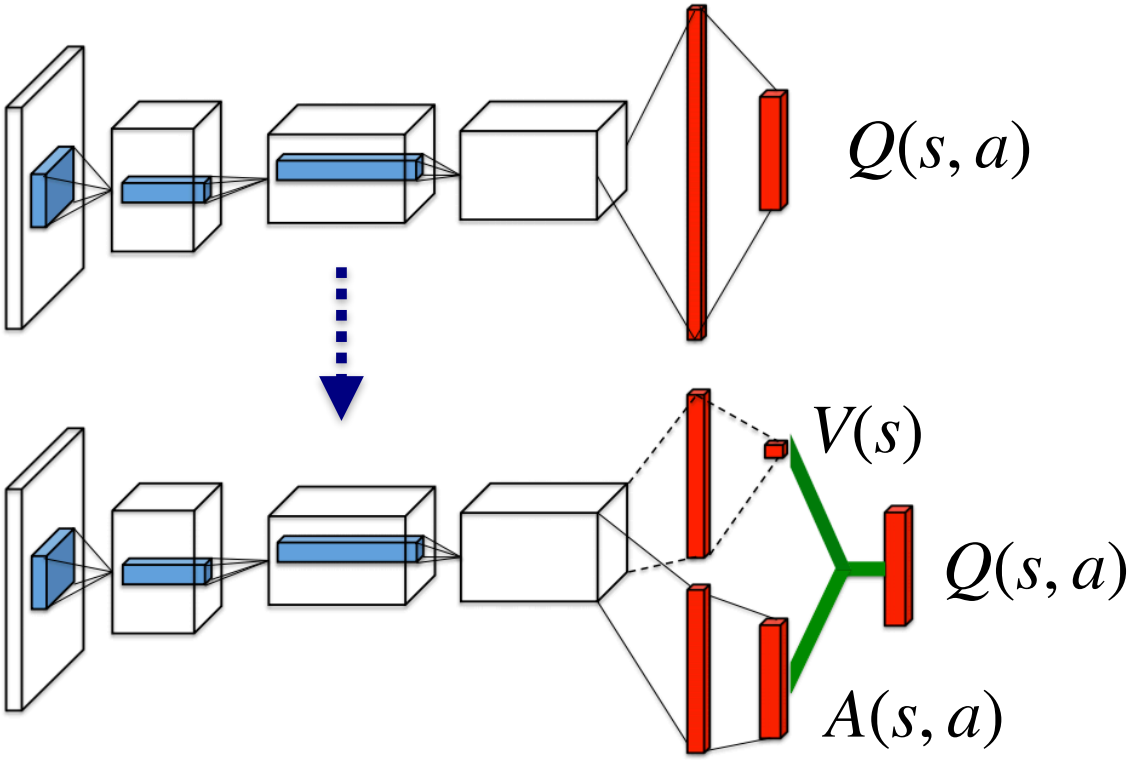
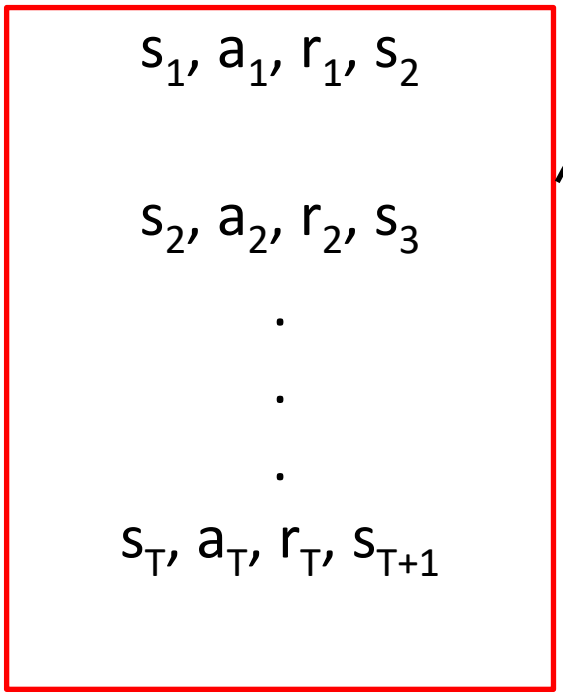


Figure 1. A popular single stream Q-network (**top**) and the dueling Q-network (**bottom**). The dueling network has two streams to separately estimate (scalar) state-value and the advantages for each action; the green output module implements equation (9) to combine them. Both networks output Q-values for each action.

Separately learn V(s) and A(s,a)

Dueling Network Architectures for Deep Reinforcement Learning, Wang et al., 2016

Let's go back to the Q-learning updates



Replay Buffer

sample uniformly!

can we do better?

$$y_t = r_t + \gamma \max_a Q(s_{t+1}, a)$$

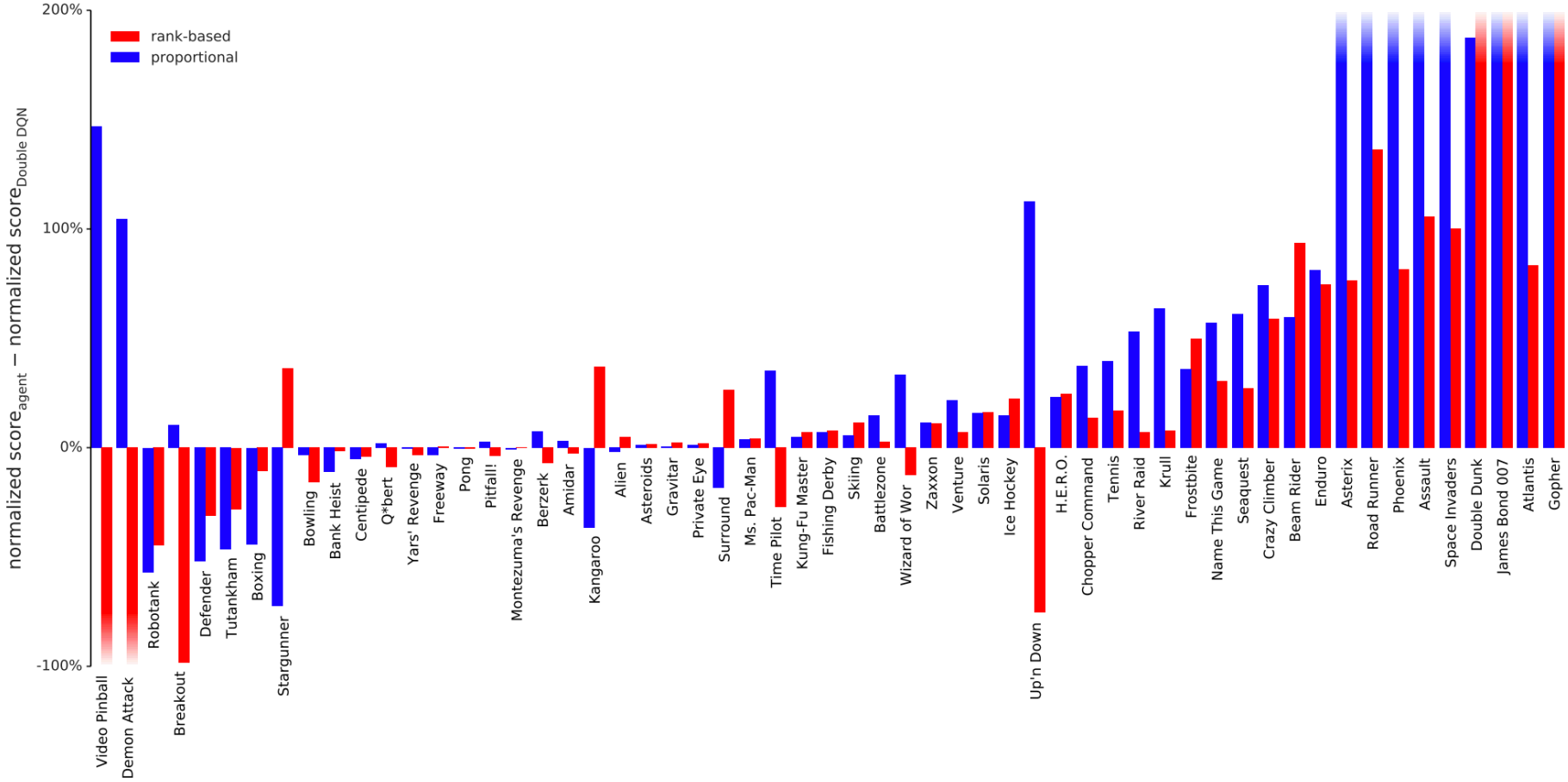
$$\min \|y_t - Q(s_t, a_t)\|^2$$

prioritized sampling!

$$p_t \propto \left| R_{t+1} + \gamma_{t+1} \max_{a'} q_{\bar{\theta}}(S_{t+1}, a') - q_\theta(S_t, A_t) \right|$$

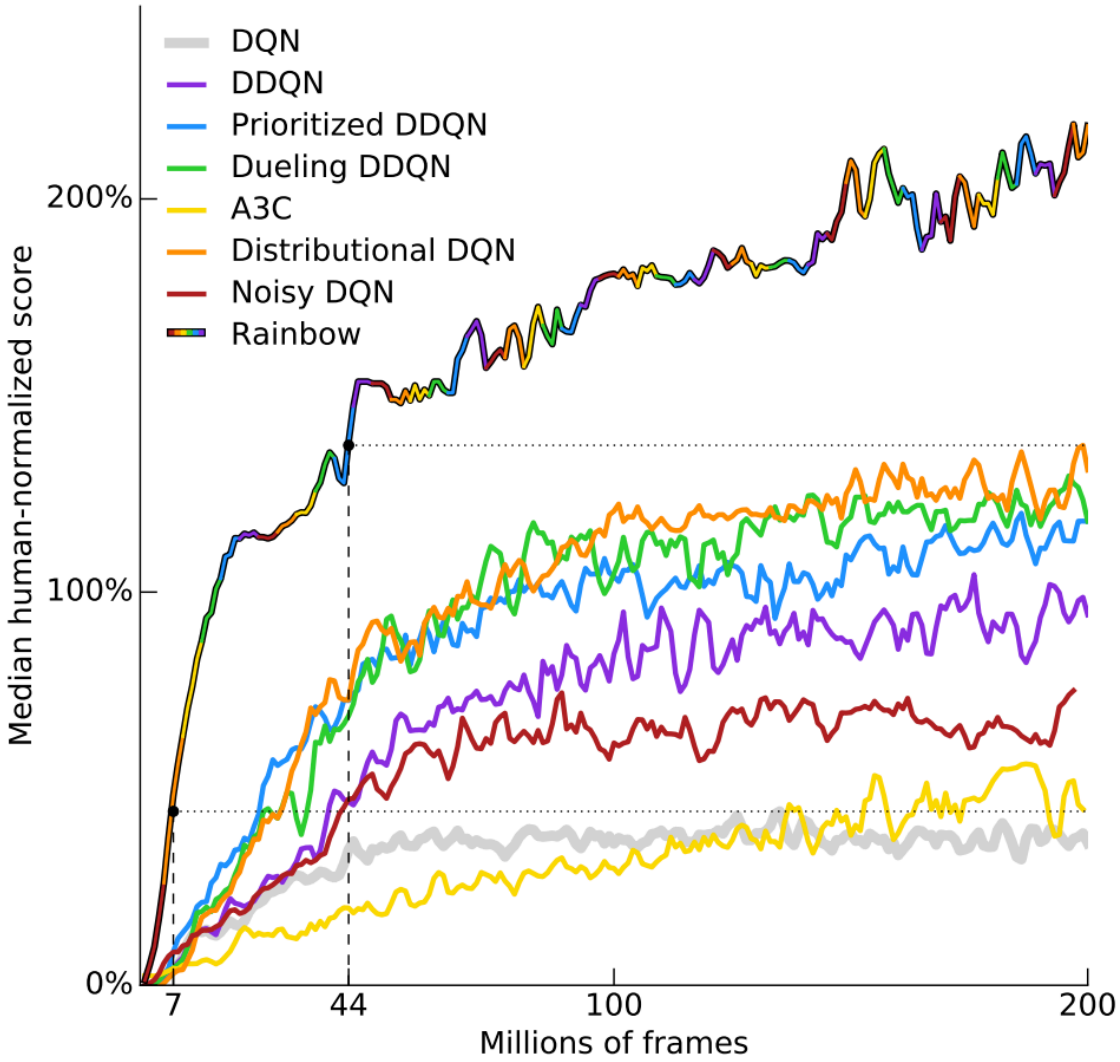
(similar to hard negative mining in ML)

Prioritized Experience Replay



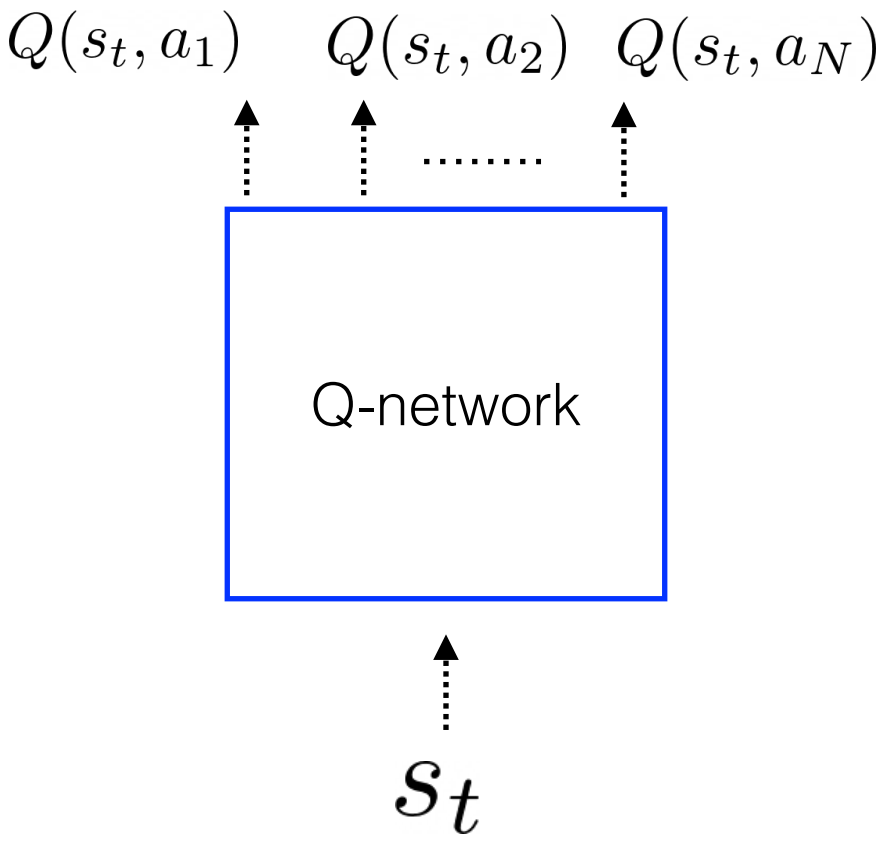
Prioritized Experience Replay, Schaul et al., 2015

Rainbow: Combining multiple of these improvements

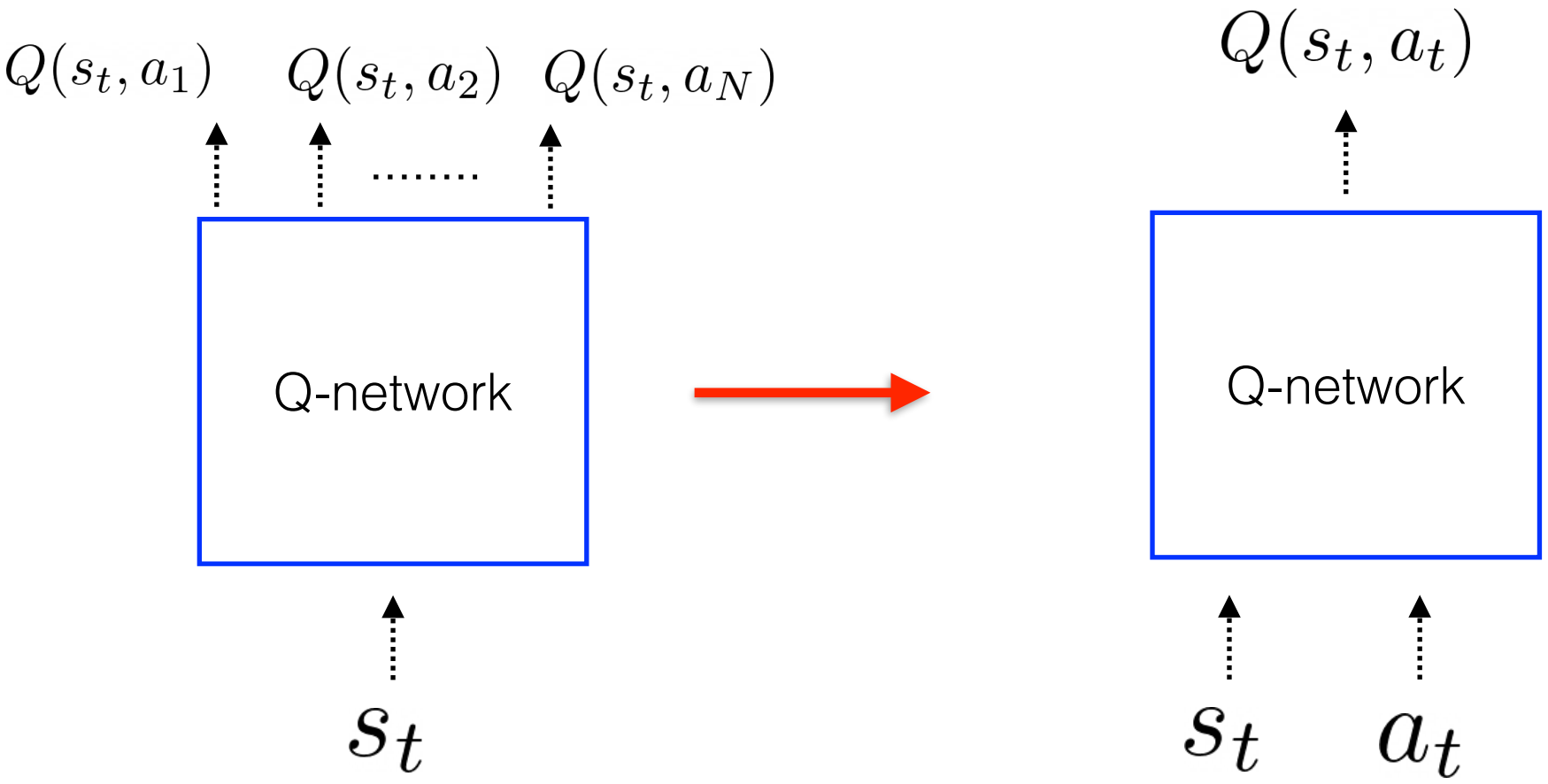


Rainbow: Combining Improvements in Deep Reinforcement Learning, Hessel et al., 2017

How do we deal with continuous actions?

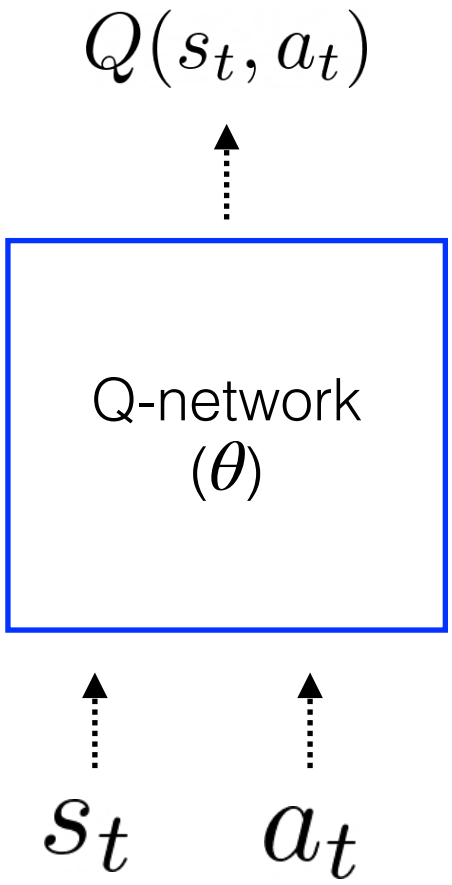


How do we deal with continuous actions?



One can use the same formalism for continuous action in Contextual Bandits

How do we deal with continuous actions?



How to find the actions?

gradient free optimization
(e.g. cross-entropy method)

gradient based methods
(backprop)

$$\max_a Q_\theta(s_t, a)$$

$$a^k \leftarrow a^{init} \; ; \; k = 0$$

repeat until convergence:

$$a^k \leftarrow a^{k-1} + \eta \nabla_a Q(s, a^{k-1})$$

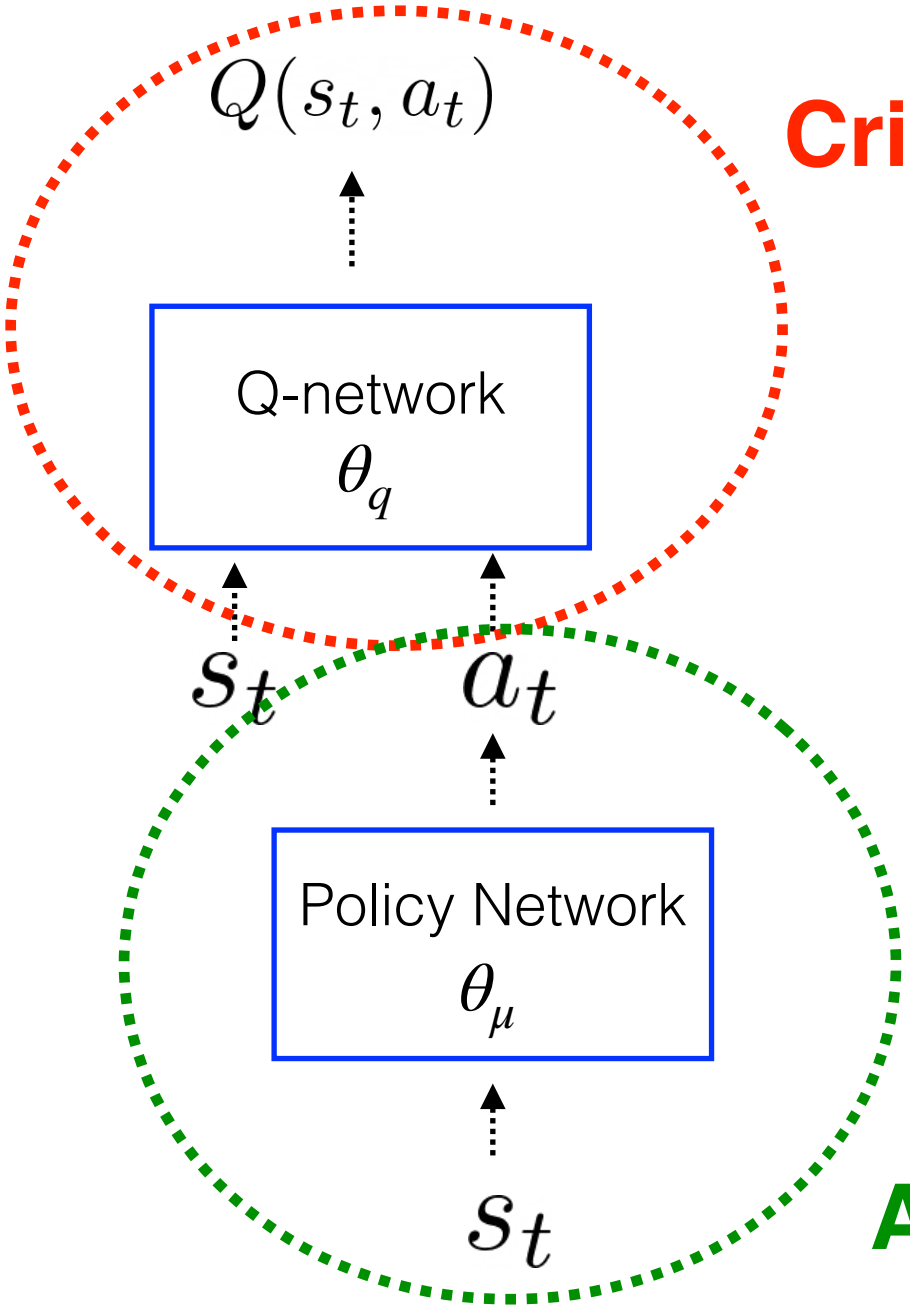
if $|a_k - a_{k-1}| < \delta$:

return

slow

Easily converge to local minima

Directly predict a?



Actor-Critic Methods

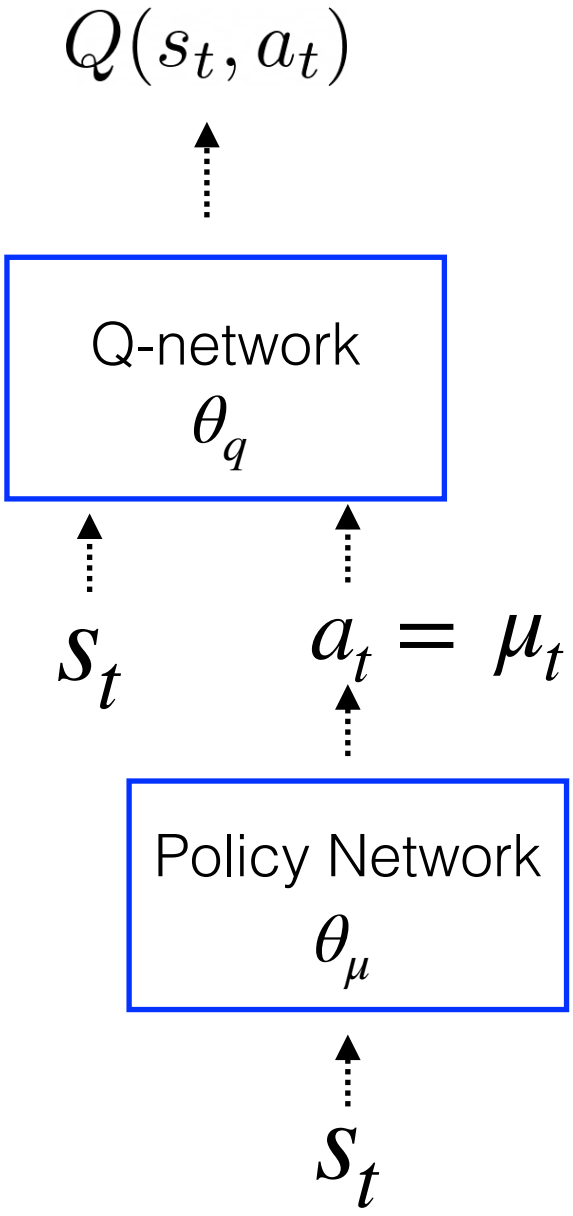
Want to take gradient with reward

but reward is non-differentiable

critic approximates the reward

provides gradient to the actor

Deep Deterministic Gradients (DDPG)



optimize the critic

$$\min_{\theta_q} \left(Q_{\theta_q}(s_t, a_t) - r_t + \gamma Q_{\theta'_q}(s_t, a_t) \right)$$

(Temporal Differencing (TD) Error)

optimize the actor

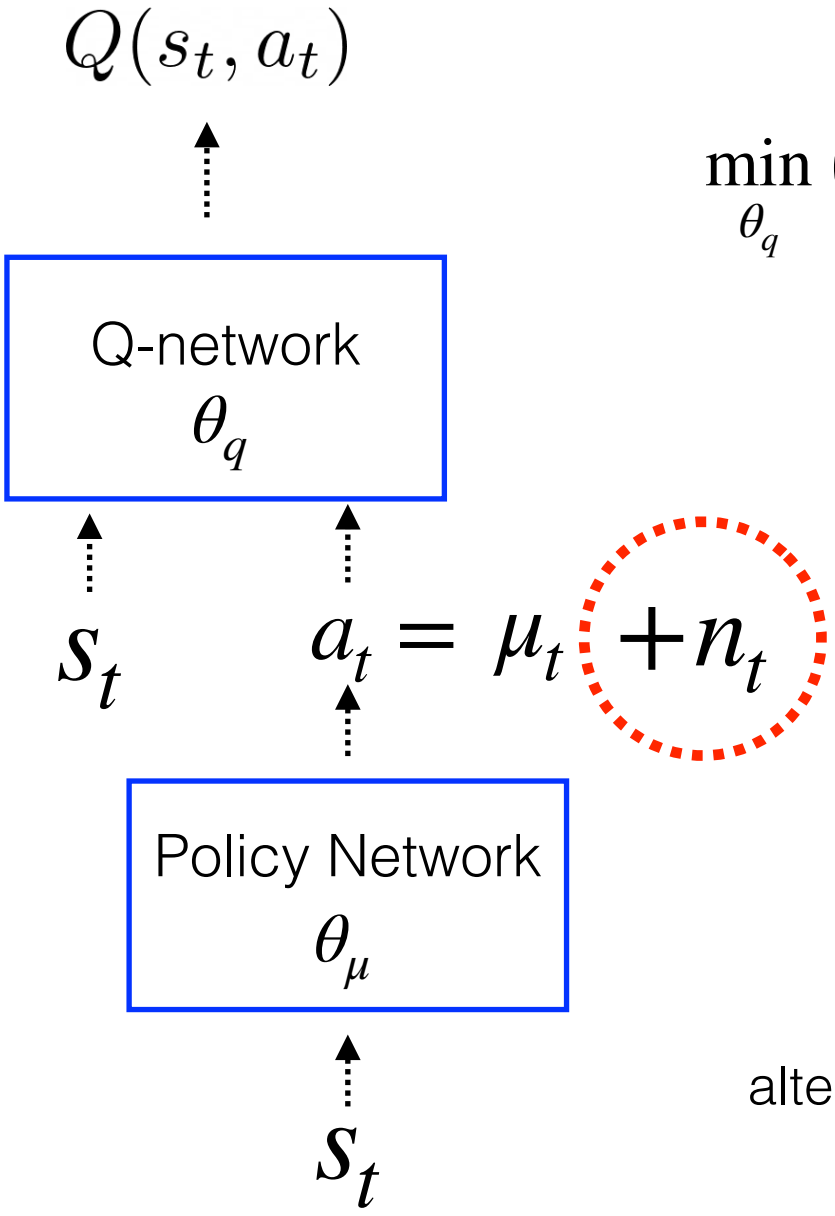
$$\max_{\theta_\mu} Q_{\theta_q}(s_t, a_t) \qquad \max_{\theta_\mu} Q_{\theta_q}(s_t, \mu_t)$$

$$\nabla_{\theta_\mu} Q_{\theta_q}(s_t, \mu_t)$$

$$\nabla_{\mu_t} Q_{\theta_q}(s_t, \mu_t) \nabla_{\theta_\mu} \mu_t$$

$$\theta_\mu^{k+1} \leftarrow \theta_\mu^k + \eta \nabla_{\mu_t} Q_{\theta_q}(s_t, \mu_t) \nabla_{\theta_\mu} \mu_t$$

Deep Deterministic Gradients (DDPG)



optimize the critic

$$\min_{\theta_q} \left(Q_{\theta_q}(s_t, a_t) - r_t + \gamma Q_{\theta'_q}(s_t, a_t) \right)$$

optimize the actor

$$\max_{\theta_\mu} Q_{\theta_q}(s_t, \mu_t)$$

Does it work?

Need exploration

$$n_t \sim \mathcal{N}(0, \sigma)$$

alternative: Ornstein-Uhlenbeck process

(time correlated noise)

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R

Receive initial observation state s_1
for t = 1, T **do**
Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R

Receive initial observation state s_1
for t = 1, T **do**
Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
Execute action a_t and observe reward r_t and observe new state s_{t+1}

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R

Receive initial observation state s_1
for t = 1, T **do**
Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
Execute action a_t and observe reward r_t and observe new state s_{t+1}
Store transition (s_t, a_t, r_t, s_{t+1}) in R

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R

Receive initial observation state s_1
for t = 1, T **do**
Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
Execute action a_t and observe reward r_t and observe new state s_{t+1}
Store transition (s_t, a_t, r_t, s_{t+1}) in R
Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R

Receive initial observation state s_1
for t = 1, T **do**
Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
Execute action a_t and observe reward r_t and observe new state s_{t+1}
Store transition (s_t, a_t, r_t, s_{t+1}) in R
Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R

Receive initial observation state s_1
for t = 1, T **do**
Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
Execute action a_t and observe reward r_t and observe new state s_{t+1}
Store transition (s_t, a_t, r_t, s_{t+1}) in R
Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R

Receive initial observation state s_1
for t = 1, T **do**
Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
Execute action a_t and observe reward r_t and observe new state s_{t+1}
Store transition (s_t, a_t, r_t, s_{t+1}) in R
Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

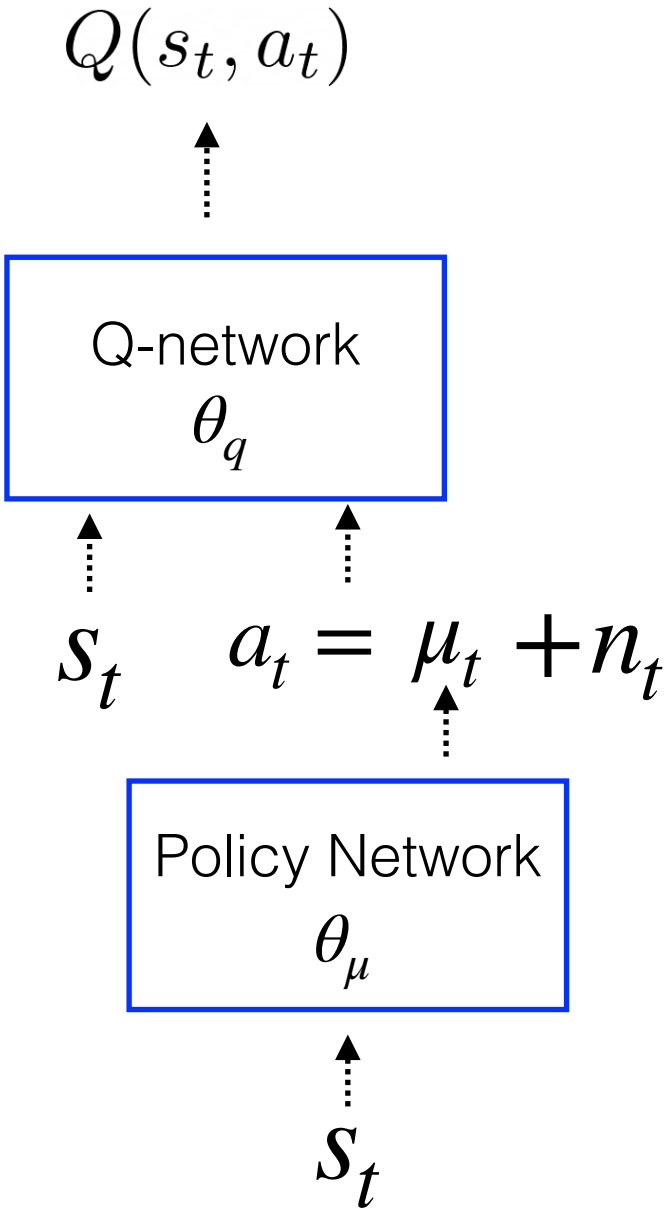
Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

end for
end for

Deep Deterministic Gradients (DDPG)



optimize the critic

$$\min_{\theta_q} \left(Q_{\theta_q}(s_t, a_t) - r_t + \gamma Q_{\theta'_q}(s_t, a_t) \right)$$

optimize the actor

$$\max_{\theta_\mu} Q_{\theta_q}(s_t, \mu_t)$$

Does something seem off?

acting with a_t

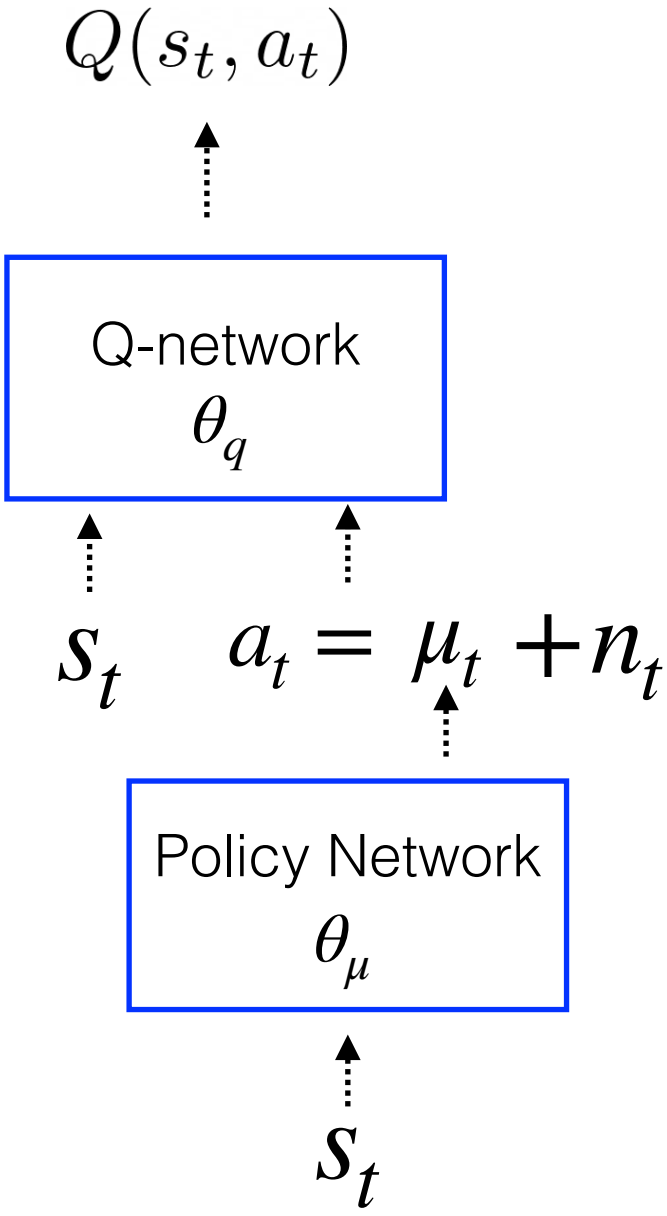
But we calculate gradient with μ_t

Can we do this?

Yes, Q-learning is off-policy

Stochastic policy helps explore!

Deep Deterministic Gradients (DDPG)



optimize the critic

$$\min_{\theta_q} \left(Q_{\theta_q}(s_t, a_t) - r_t + \gamma Q_{\theta'_q}(s_t, a_t) \right)$$

optimize the actor

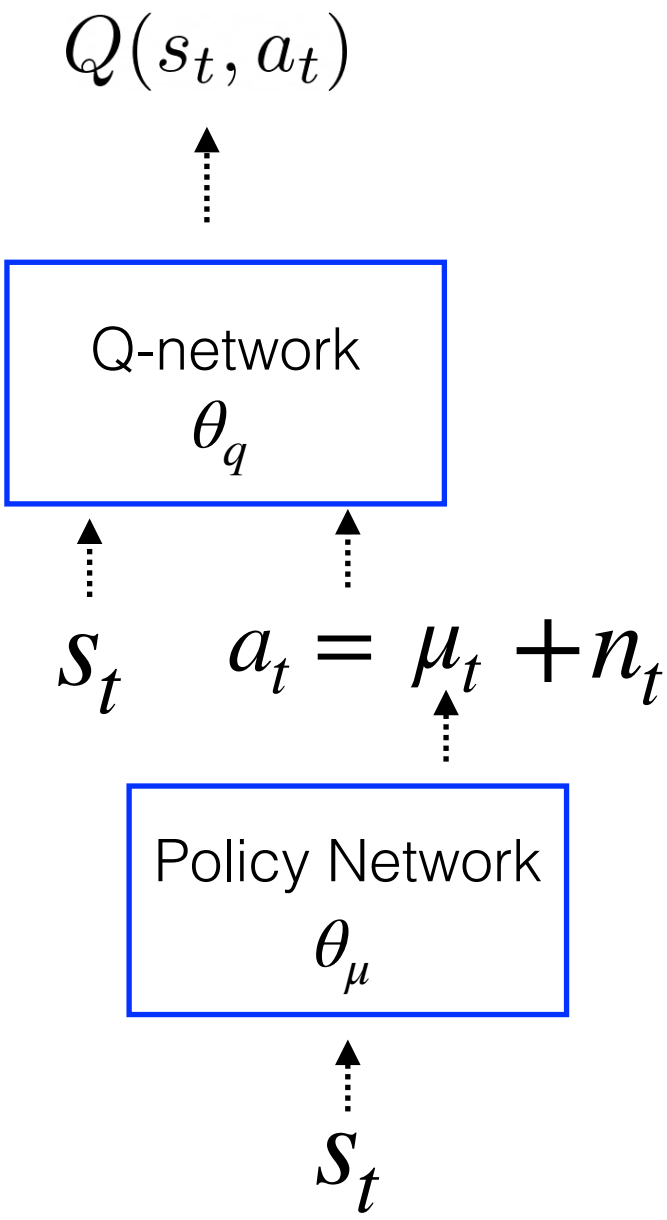
$$\max_{\theta_\mu} Q_{\theta_q}(s_t, \mu_t)$$

Could something else go wrong?

$$Q(s, a) \sim Q_{\theta_q}(s, a)$$

If approximation is not perfect,
actor gradients will be noisy!
(may not be even an ascent direction)

Deep Deterministic Gradients (DDPG)



Could something else go wrong?

$$Q(s,a) \sim Q_{\theta_q}(s,a)$$

Two conditions for accurate gradient

$$\nabla_a Q_{\theta_q}(s,a)|_{a=\mu(s)} = \nabla_{\theta_\mu}\mu(s)^T\theta_q$$

And

θ_q minimizes

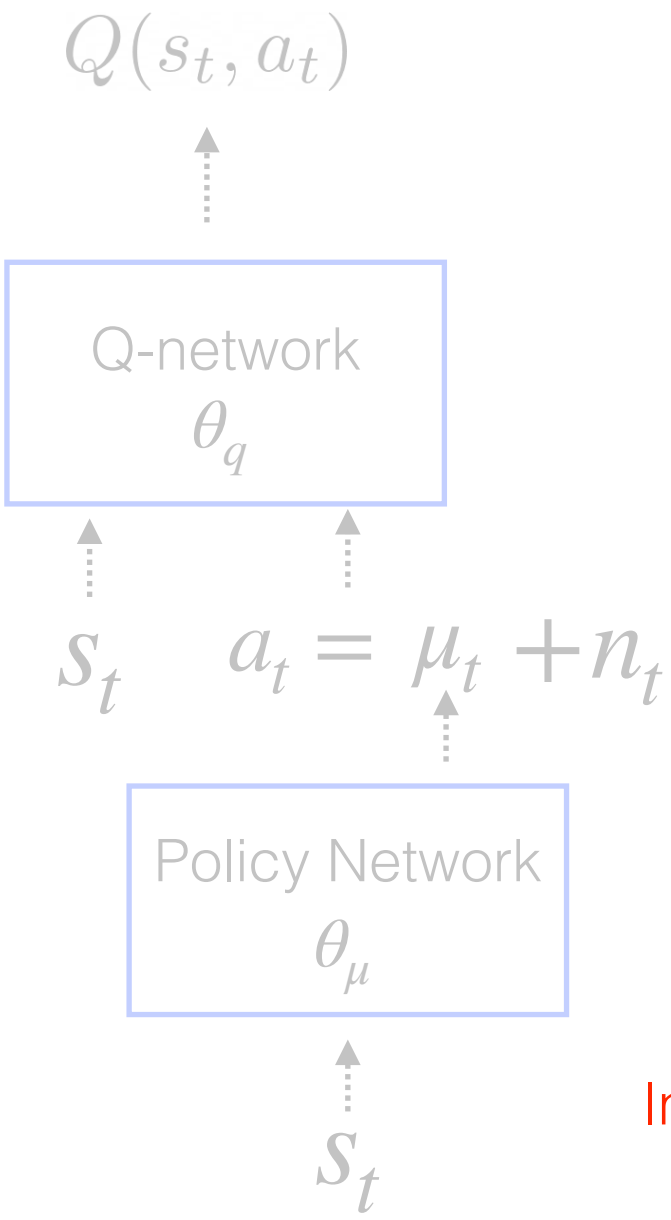
$$E[\epsilon(s;\theta_q,\theta_\mu)^T\epsilon(s;\theta_q,\theta_\mu)]$$

$$\epsilon(s;\theta_q,\theta_\mu) = \nabla_a\left(Q(s,a) - Q_{\theta_q}(s,a)\right)|_{a=\mu(s)}$$

Don't have access!

In practice: use TD Error

Deep Deterministic Gradients (DDPG)



Two conditions for accurate gradient

$$\nabla_a Q_{\theta_q}(s,a)\,|_{a=\mu(s)} = \nabla_{\theta_\mu}\mu(s)^T\theta_q$$

And

θ_q minimizes

$$E[\epsilon(s;\theta_q,\theta_\mu)^T\epsilon(s;\theta_q,\theta_\mu)]$$

$$\epsilon(s;\theta_q,\theta_\mu) = \nabla_a\left(Q(s,a) - Q_{\theta_q}(s,a)\right)\,|_{a=\mu(s)}$$

To satisfy condition 1

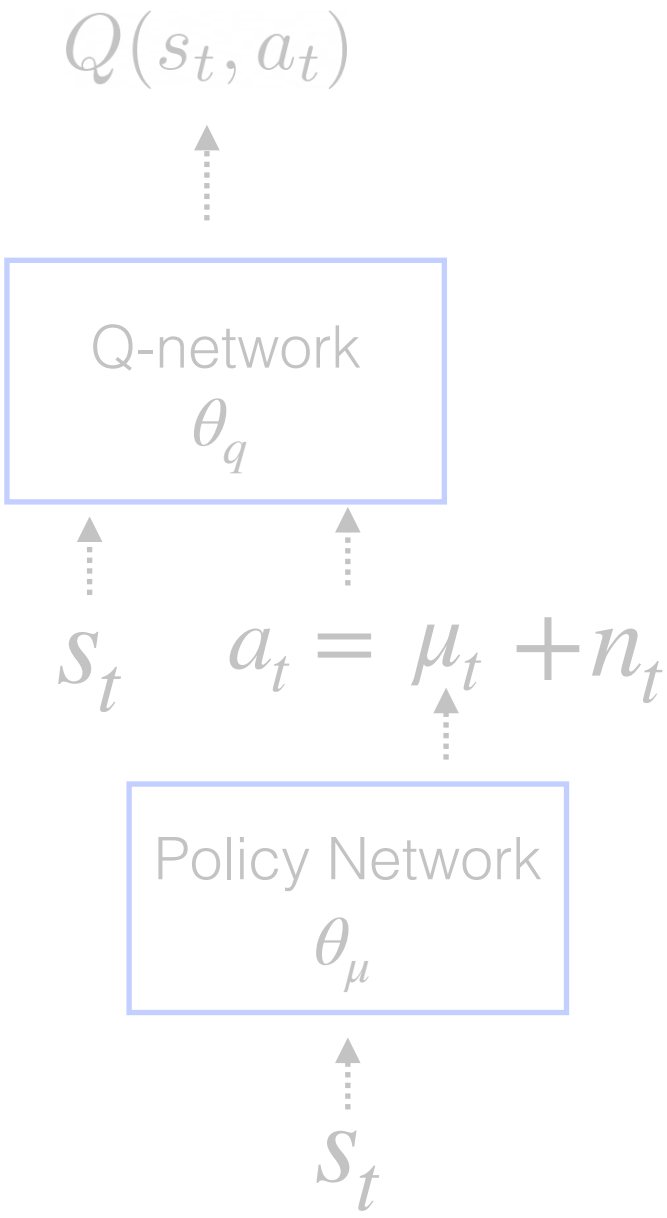
$$Q_{\theta_q}(s,a) = \phi(s,a)^T\theta_q$$

$$\phi(s,a) = \nabla_{\theta_\mu}\mu(s)(a - \mu(s))$$

In practice: $Q_{\theta_q}(s,a)$ is not linear in $\phi(s,a)$

$\phi(s,a)$ is learned using function appx.

Deep Deterministic Gradients (DDPG)



How do we solve in practice?

$$Q(s,a) \sim Q_{\theta_q}(s,a)$$

Big reason for instability of DDPG

Double Q-learning

Measure uncertainty of Q

Other methods

TD3: Twin Delayed DDPG

Trick One: Clipped Double-Q Learning. TD3 learns *two* Q-functions instead of one (hence "twin"), and uses the smaller of the two Q-values to form the targets in the Bellman error loss functions.

Slide adapted from Open AI Spinning Up

Soft Actor Critic (SAC)

$$Q^{\pi}(s,a) = \underset{\substack{s' \sim P \\ a' \sim \pi}}{\mathrm{E}} \left[R(s,a,s') + \gamma \left(Q^{\pi}(s',a') + \alpha H\left(\pi(\cdot|s')\right) \right) \right]$$

Entropy term: key difference

Slide adapted from Open AI Spinning Up