Foundations of RL
Lecture 10: Modern RL Algorithms

Enrique Mallada

Goals:

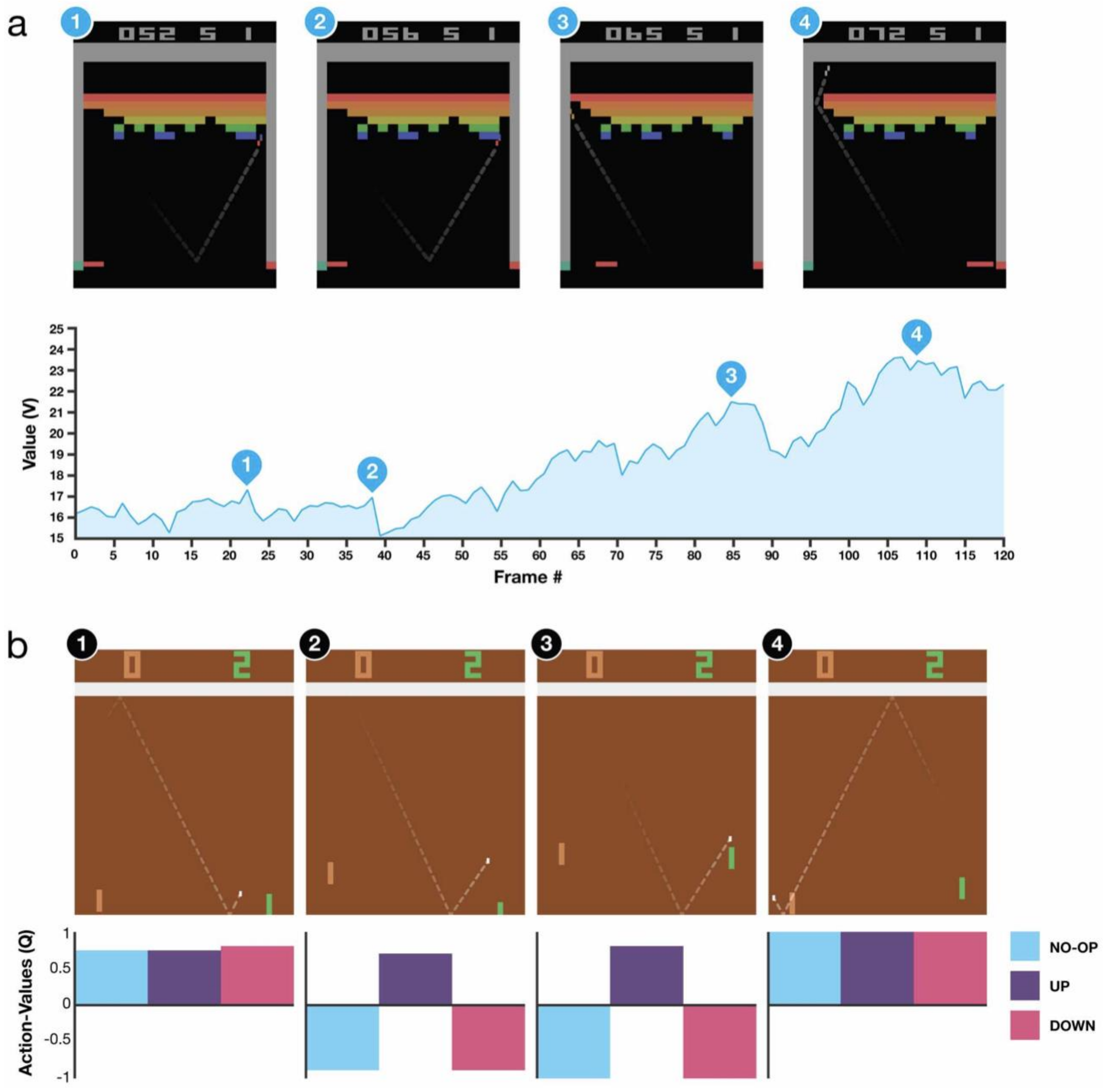
- Value-Based Methods
  - Intro, Q-Learning, Guarantees
- Advanced Methods
  - Practical Issues and Solutions, Improvements
  - Applications to Continuous Action Spaces
  - Advanced Discrete-Actions Methods

Value-Based Methods

- Intro to Value-Based Methods
- Q-Learning
- Guarantees
- Practical Issues and Solutions

Value-Based Methods

- **Intro to Value-Based Methods**
  - Policy Iteration
  - (Fitted) Value Iteration
  - (Fitted) Q Iteration
- **Q-Learning**
  - Why can Q iteration be off-policy?
  - What are we optimizing and how?
  - Online Q Iteration
  - Exploration
- **Guarantees**
  - Exist for tabular case
  - Not with function approximation



Mnih et al. Human-level control through deep reinforcement learning. *Nature* **518**, 529-533 (2015).

Intro to Value-Based Methods

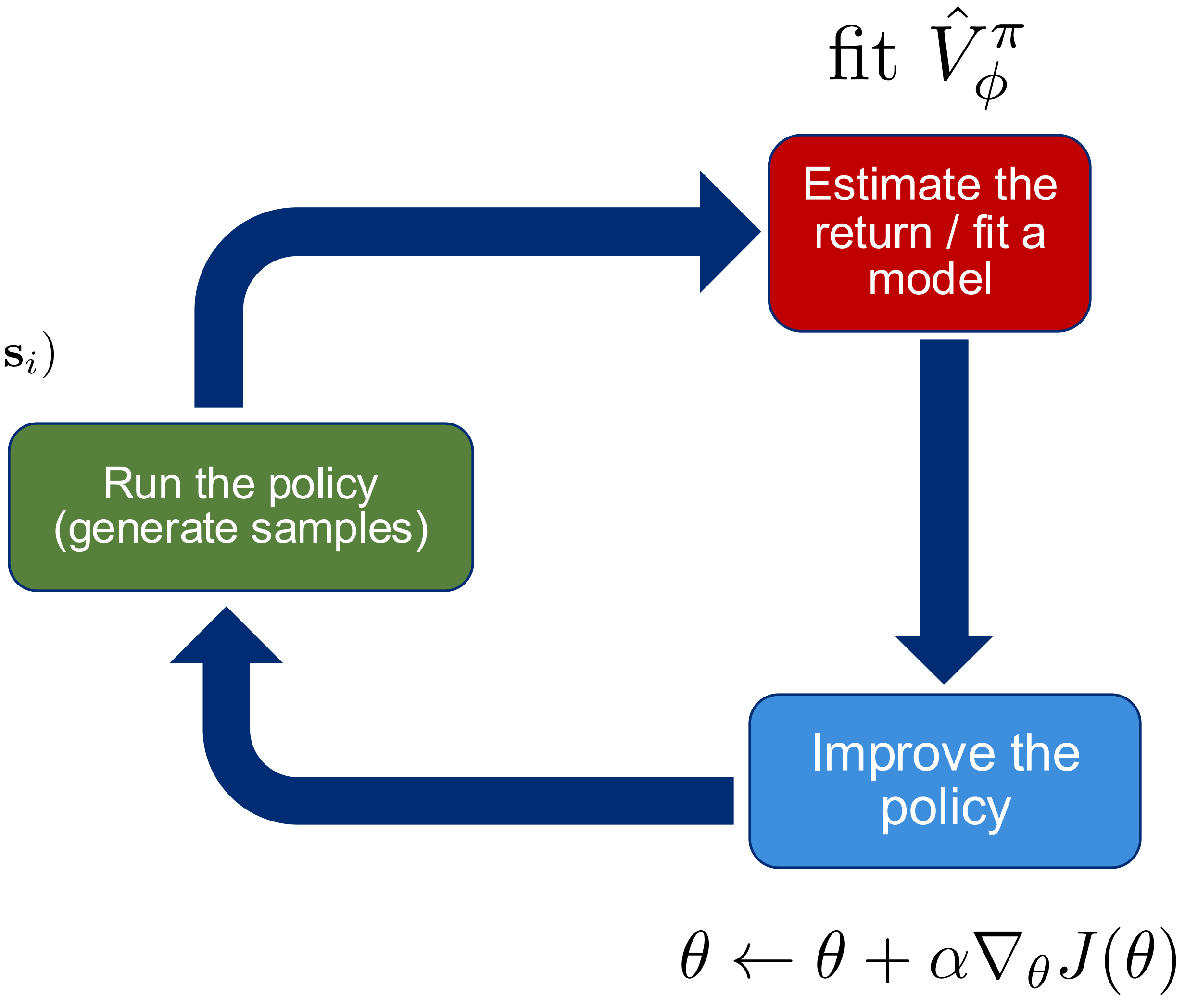
- Policy Iteration
- (Fitted) Value Iteration
- (Fitted) Q Iteration

Recap: Actor-Critic

Batch Actor-Critic Algorithm:

While not converged:

1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$
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. $\theta \longleftarrow \theta + \alpha \nabla_\theta J(\theta)$



- Actor-critic algorithms
- Policy evaluation
- Discount factors
- Actor-critic algorithm design
- State-dependent baselines
- Off-Policy Actor-Critic
- Advanced policy gradient methods

Can we omit the policy gradient?

$A^\pi(\mathbf{s}_t, \mathbf{a}_t)$: advantage; how much better $\mathbf{a}_t$ is 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

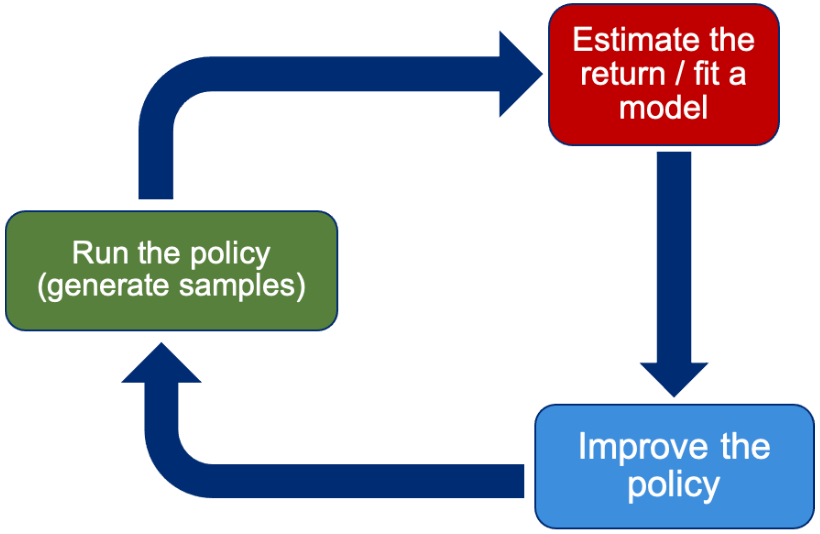
$\rightarrow$ at least as good as any $\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t)$

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

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

as good as π (better unless π is optimal)

fit A^π (or Q^π or V^π)



$\pi \leftarrow \pi'$

Policy Iteration

Policy iteration algorithm:

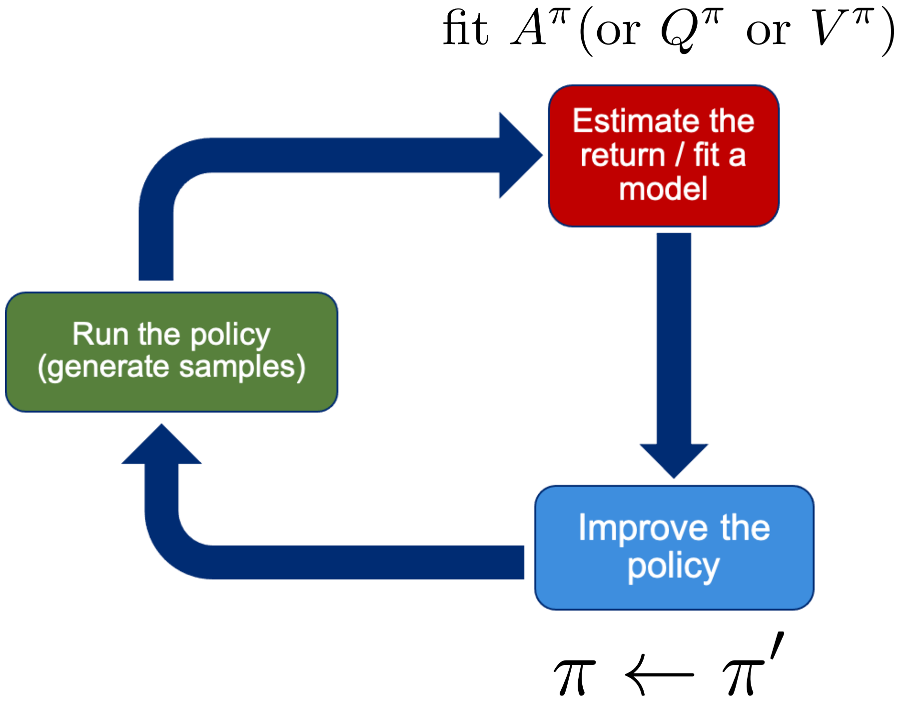
While not converged:

1. Evaluate $A^\pi(\mathbf{s}, \mathbf{a})$ ← How?
2. Set $\pi \leftarrow \pi'$

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

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

How do we evaluate $V^\pi(\mathbf{s})$?



Tabular RL: Dynamic Programming

Assume we know $p(\mathbf{s}'|\mathbf{s},\mathbf{a})$(i.e. transition probabilities).

$V^{\pi}(s_{11})$	$V^{\pi}(s_{12})$	$V^{\pi}(s_{13})$	$V^{\pi}(s_{14})$
$V^{\pi}(s_{21})$	$V^{\pi}(s_{22})$	$V^{\pi}(s_{23})$	$V^{\pi}(s_{24})$
$V^{\pi}(s_{31})$	$V^{\pi}(s_{32})$	$V^{\pi}(s_{33})$	$V^{\pi}(s_{34})$
$V^{\pi}(s_{41})$	$V^{\pi}(s_{42})$	$V^{\pi}(s_{43})$	$V^{\pi}(s_{44})$

For example, in a system with 16 states and 4 actions per state:

We can store $V^{\pi}(\mathbf{s})$ in a table

$\mathcal{T}$ is $16 \times 16 \times 4$

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

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

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

Policy Iteration Using Dynamic Programming

Policy iteration algorithm:

While not converged:

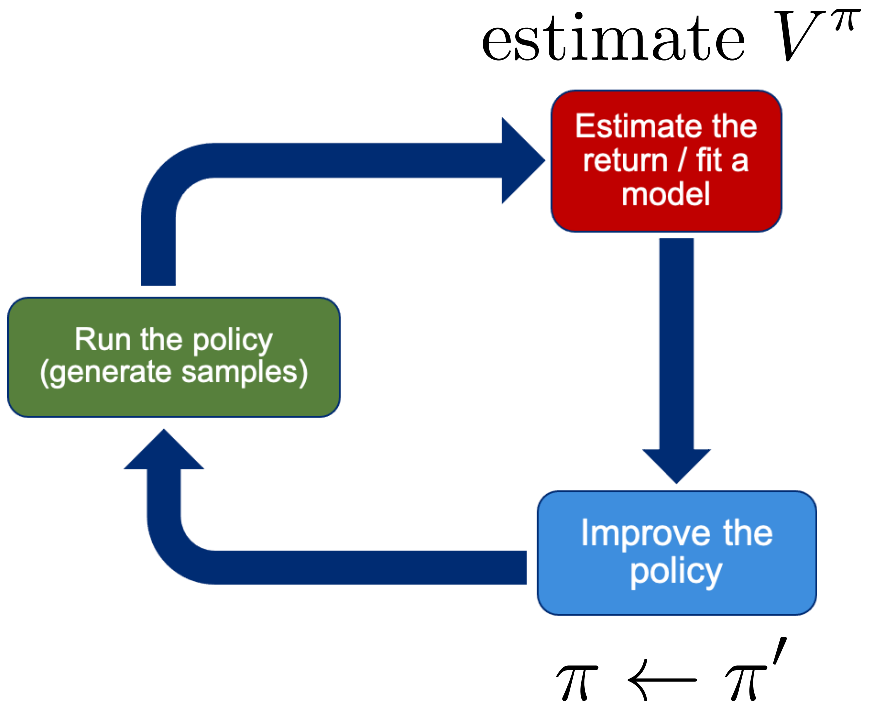
1. Evaluate $A^{\pi}(\mathbf{s}, \mathbf{a})$
2. Set $\pi \leftarrow \pi'$

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

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

$V^{\pi}(s_{11})$	$V^{\pi}(s_{12})$	$V^{\pi}(s_{13})$	$V^{\pi}(s_{14})$
$V^{\pi}(s_{21})$	$V^{\pi}(s_{22})$	$V^{\pi}(s_{23})$	$V^{\pi}(s_{24})$
$V^{\pi}(s_{31})$	$V^{\pi}(s_{32})$	$V^{\pi}(s_{33})$	$V^{\pi}(s_{34})$
$V^{\pi}(s_{41})$	$V^{\pi}(s_{42})$	$V^{\pi}(s_{43})$	$V^{\pi}(s_{44})$

policy evaluation



Even Simpler: Value Iteration

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

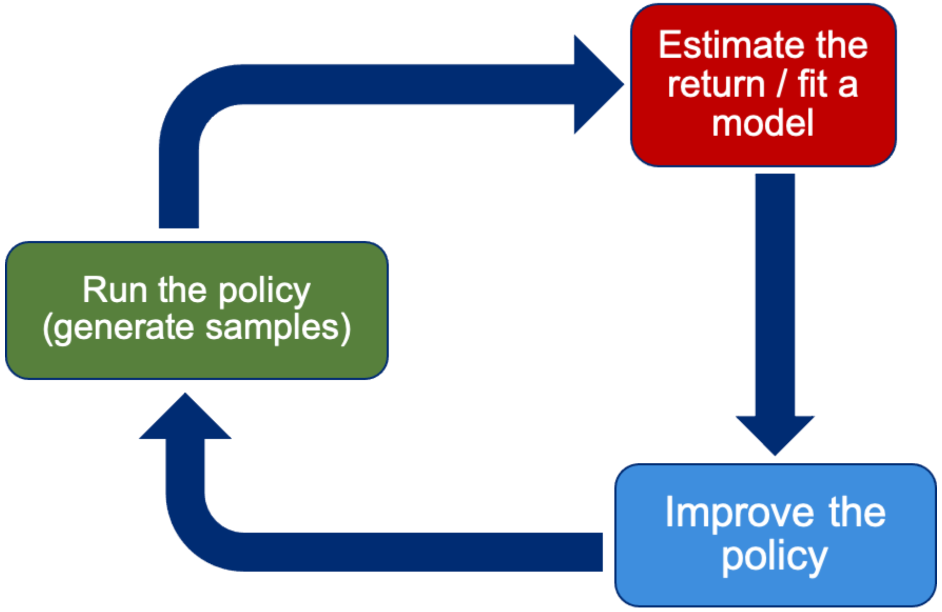
$$A^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma E[V^\pi(\mathbf{s}')] - V^\pi(\mathbf{s})$$

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

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

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

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



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

Value Iteration Algorithm:

While not converged:

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

Fitted Value Iteration

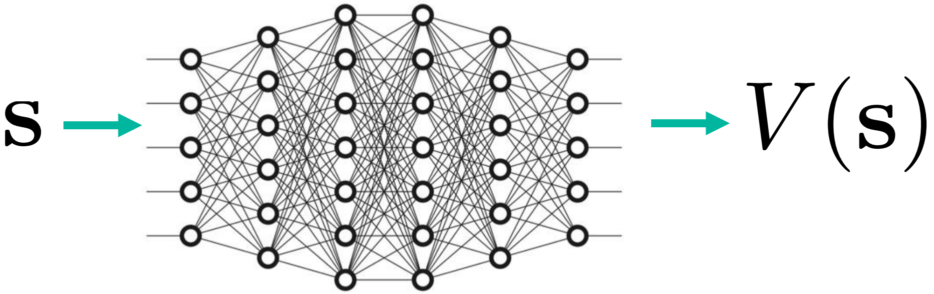
$$|\mathcal{S}| \sim (255^3)^{256\times 256}$$

Curse of dimensionality!

Fitted Value Iteration Algorithm:

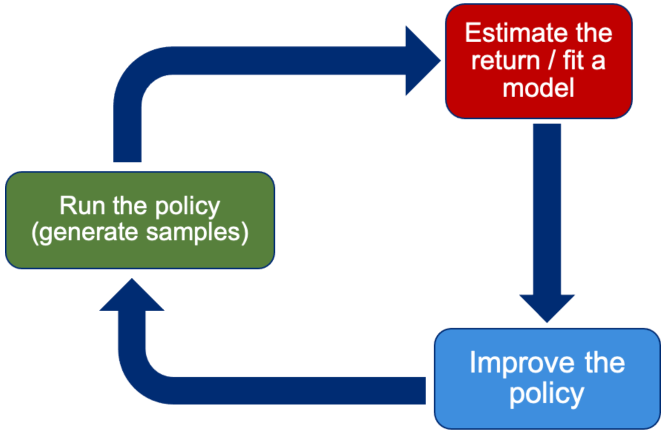
While not converged:

1. Set $y_i \leftarrow \max_{\mathbf{a}_i}(r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V(\mathbf{s}_i')])$
2. Set $\phi \leftarrow \text{argmin}_\phi \sum_i ||V_\phi(\mathbf{s}_i) - y_i||^2$



$$\mathcal{L}(\phi) = ||V_\phi(\mathbf{s}) - \max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})||^2$$

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



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

What if transition probabilities are unknown?

Fitted Value Iteration Algorithm:

While not converged:

1. Set $y_i \leftarrow \max_{\mathbf{a}_i}(r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V(\mathbf{s}_i')])$
2. Set $\phi \leftarrow \text{argmin}_\phi \sum_i ||V_\phi(\mathbf{s}_i) - y_i||^2$

Need to know outcomes for different actions!

Policy iteration algorithm:

While not converged:

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

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

policy evaluation

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

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

- **Can fit via sampling!**

Fitted Q Iteration

Policy iteration algorithm:

While not converged:

1. Evaluate $V^{\pi}(\mathbf{s}, \mathbf{a})$
2. Set $\pi \leftarrow \pi'$

Compute V directly (no policy)

Fitted Value Iteration Algorithm:

While not converged:

1. Set $y_i \leftarrow \max_{\mathbf{a}_i}(r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V(\mathbf{s}_i')])$
2. Set $\phi \leftarrow \arg\min_{\phi} \sum_i ||V_{\phi}(\mathbf{s}_i) - y_i||^2$

➢ **Can we do a similar thing with Q, removing the need to know transition probabilities?**

Fitted Q Iteration Algorithm:

While not converged:

1. Set $y_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V_{\phi}(\mathbf{s}_i')]$
2. Set $\phi \leftarrow \arg\min_{\phi} \sum_i ||Q_{\phi}(\mathbf{s}_i, \mathbf{a}_i) - y_i||^2$

approximate $E[V(\mathbf{s}_i')] \approx \max_{\mathbf{a}} Q_{\phi}(\mathbf{s}_i', \mathbf{a}_i')$

➢ **Don't need to try over all actions; only uses those that were sampled!**

Fitted Q Iteration

Full Fitted Q Iteration Algorithm:

While not converged:

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

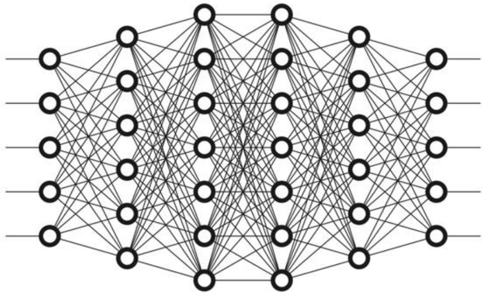
Until data refresh:

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

Hyperparameters:
- Collection policy
- Dataset size
- # Iterations
- # Gradient steps

$\mathbf{s}$

$\mathbf{a}$



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

parameters ϕ

often structured to have $\mathbf{s}$ as input, different output for each $\mathbf{a}$

- Works off-policy
- Only one network, no high-variance policy gradient
- No convergence guarantees for non-linear function approximation

Q-Learning

- Why can Q iteration be off-policy?
- What are we optimizing and how?
- Online Q Iteration
- Exploration

Why can this be off-policy?

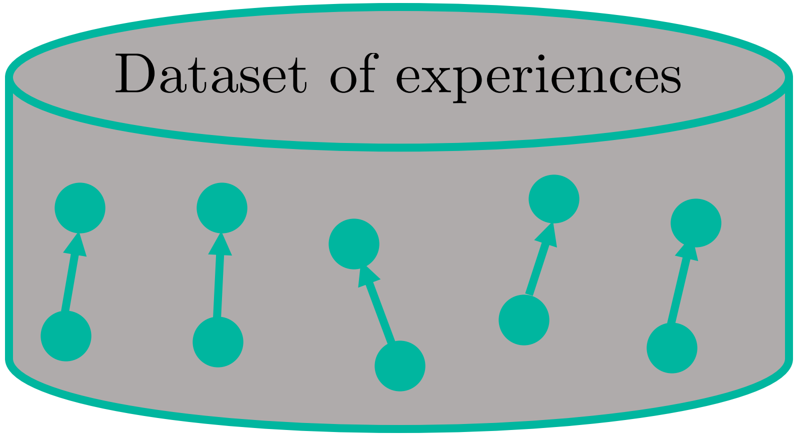
Full Fitted Q Iteration Algorithm:

While not converged:

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

Until data refresh:

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



Given $\mathbf{s}$ and $\mathbf{a}$, transition is independent of π!

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

➢ **We can base our updates off of a database of transitions from different policies!**

What are we optimizing?

Full Fitted Q Iteration Algorithm:

While not converged:

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

Until data refresh:

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

Bellman Error:

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

- If Bellman error is 0, both the Q function and the policy are optimal
- We can get there in tabular case
- However, once function approximators are added, there is no guarantee.

Online Q-Learning

Full Fitted Q Iteration Algorithm:

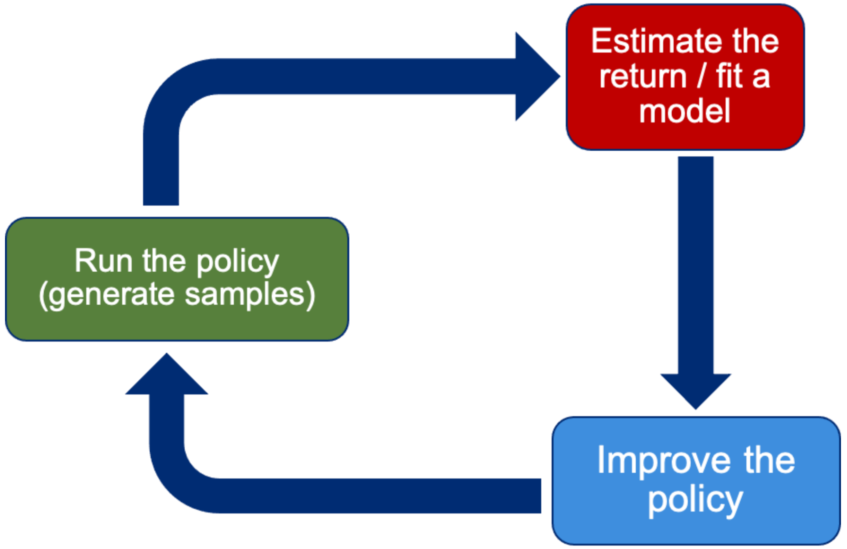
While not converged:

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

   Until data refresh:

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

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



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

Online Q Iteration Algorithm:

While not converged:

1. Take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. Set $y_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi(\mathbf{s}_i, \mathbf{a}_i)}{d\phi} \underbrace{(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - y_i)}_{\text{"temporal difference error"}}$

Watkins Q-Learning

Exploration in Q-Learning

Online Q Iteration Algorithm:

While not converged:

1. Take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. Set $y_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. $\phi \leftarrow \phi - \alpha \dfrac{dQ_\phi(\mathbf{s}_i, \mathbf{a}_i)}{d\phi}(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - y_i)$

- **To get this to work well, more work needs to be done.**

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

"Epsilon-greedy"

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

"Boltzmann"

Recap: Q-learning

Full Fitted Q Iteration Algorithm:

While not converged:

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

   Until data refresh:

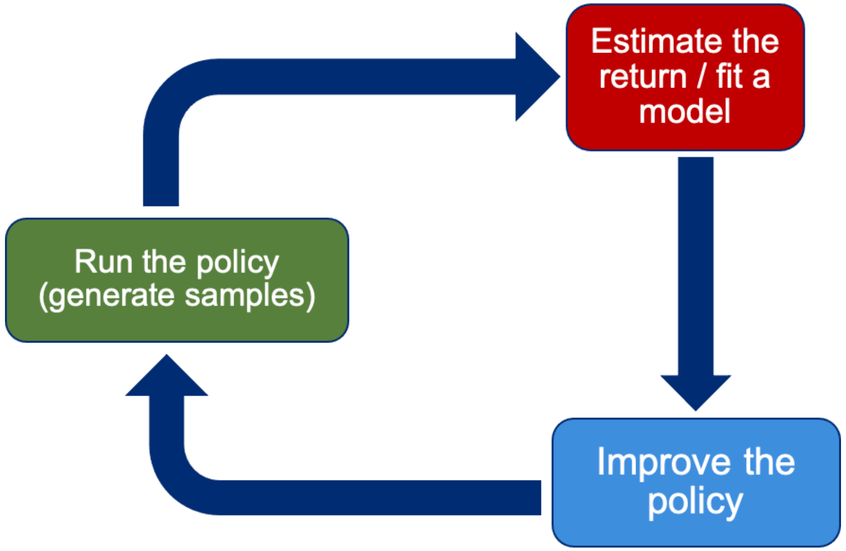
   2. Set $y_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
   3. Set $\phi \leftarrow \arg\min_\phi \sum_i ||Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - y_i||^2$

Online Q Iteration Algorithm:

While not converged:

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

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



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

Lack of Guarantees

- Guarantees exist for tabular case
- Not with function approximation

Learning Q Functions: Tabular Case

Get to Q via argmax policy

$V^{\pi}(s_{11})$	$V^{\pi}(s_{12})$	$V^{\pi}(s_{13})$	$V^{\pi}(s_{14})$
$V^{\pi}(s_{21})$	$V^{\pi}(s_{22})$	$V^{\pi}(s_{23})$	$V^{\pi}(s_{24})$
$V^{\pi}(s_{31})$	$V^{\pi}(s_{32})$	$V^{\pi}(s_{33})$	$V^{\pi}(s_{34})$
$V^{\pi}(s_{41})$	$V^{\pi}(s_{42})$	$V^{\pi}(s_{43})$	$V^{\pi}(s_{44})$

Value Iteration Algorithm:

While not converged:

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

Define Bellman operator $\mathcal{B}$: $\mathcal{B}Q = r + \gamma \mathcal{T} \max_{\mathbf{a}} Q$

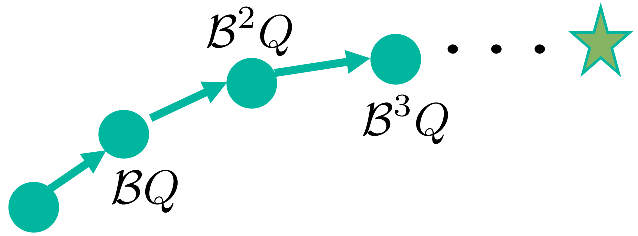
Optimal $Q^\star$ is a *fixed* point of $\mathcal{B}$, i.e. $Q^\star = r + \gamma \mathcal{T} \max_{\mathbf{a}} Q^\star$

$\rightarrow$ Always exists, is always unique, always corresponds to optimal policy

$\rightarrow$ Can prove that Q iteration always reaches $Q^\star$ because $\mathcal{B}$ is a *contraction*

Contraction: $\forall Q, \bar{Q}$: $||\mathcal{B}Q - \mathcal{B}\bar{Q}||_\infty \leq \gamma ||Q - \bar{Q}||_\infty$

$||\mathcal{B}Q - \mathcal{B}Q^\star||_\infty \leq \gamma ||Q - Q^\star||_\infty$



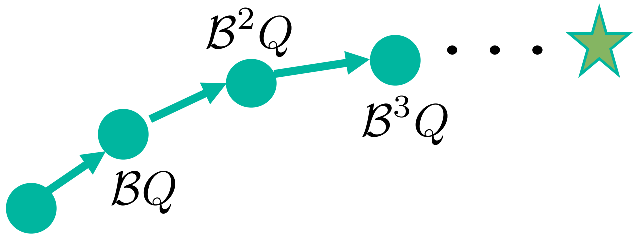
< 1, gap decreases with iterations

Fitted Q Iteration: No Guarantees

Fitted Q Iteration Algorithm:

While not converged:

1. Set $y_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma E[V_\phi(\mathbf{s}_i')]$
2. Set $\phi \leftarrow \text{argmin}_\phi \sum_i ||Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - y_i||^2$



Define an operator Π : $\Pi Q = \text{argmin}_{Q' \in \Omega} \frac{1}{2} \sum ||Q'(\mathbf{s}, \mathbf{a}) - Q(\mathbf{s}, \mathbf{a})||^2$

$\rightarrow \Pi$ is a projection onto Ω in terms of $\ell 2$ norm.

Fitted Q Iteration algorithm:

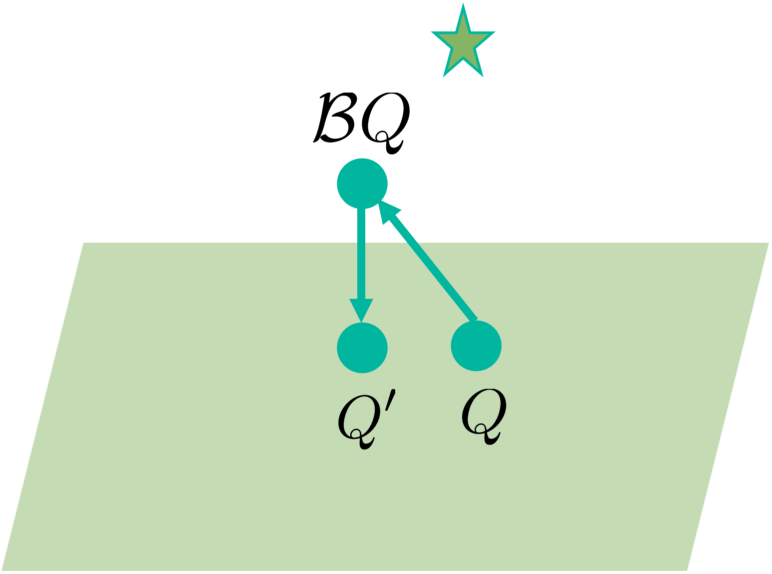
While not converged:

1.$Q \leftarrow \Pi \mathcal{B} Q$

$\rightarrow \mathcal{B}$ is a contraction w.r.t. ∞-norm: $||Q - \mathcal{B}\bar{Q}||_\infty \leq \gamma ||Q - \bar{Q}||_\infty$

$\rightarrow \Pi$ is a contraction w.r.t. $\ell 2$-norm: $||Q - \Pi \bar{Q}||^2 \leq ||Q - \bar{Q}||^2$

$\rightarrow \Pi \mathcal{B}$ is *not* a contraction w.r.t any norm!



➢ **No guarantees, in principle or practice…**

Ω(e.g. set of functions that can be represented by NN)

Online Q-Learning: Practical Issues

Online Q Iteration Algorithm:

While not converged:

1. Take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. Set $y_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi(\mathbf{s}_i, \mathbf{a}_i)}{d\phi}(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - y_i)$ → ~~Gradient descent?~~

Not gradient descent!

$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi(\mathbf{s}_i, \mathbf{a}_i)}{d\phi}(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \underbrace{[r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)]}_{\text{No gradient through this!}})$$

Advanced Value-Based Methods

- Practical Issues and Solutions
- Improvements
- Applications to Continuous Action Spaces
- Advanced Discrete-Actions Methods

Practical Issues and Solutions

- Correlated data → replay buffer
- Moving target → target network

Online Q-Learning: Practical Issues

Online Q Iteration Algorithm:

While not converged:

1. Take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2. Set $y_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi(\mathbf{s}_i, \mathbf{a}_i)}{d\phi}(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - y_i)$ → ~~Gradient descent?~~

Not gradient descent!

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

No gradient through this!

Correlated Samples: Parallelization

Online Q Iteration Algorithm:

While not converged:

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

Correlation $\Longrightarrow$ local overfitting, forgetting!

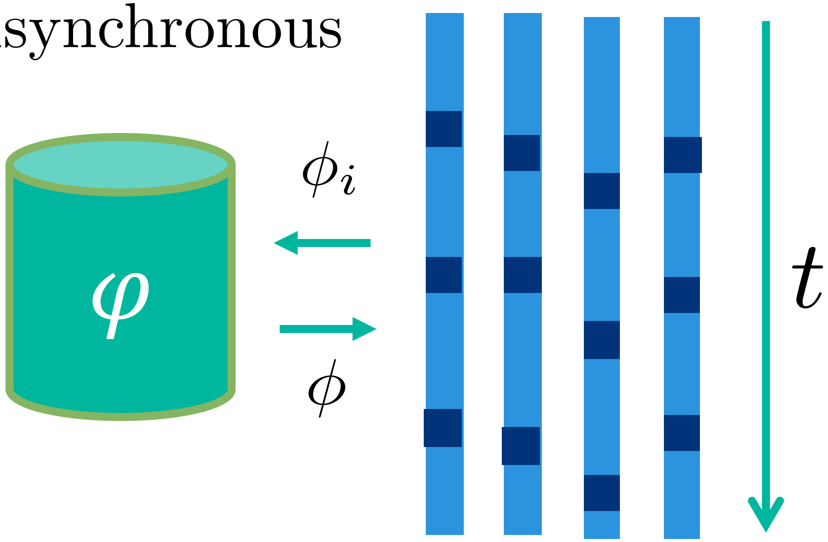
$\rightarrow$ A partial solution: parallelize!

Synchronous

Get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$

Update ϕ

Asynchronous



Correlated Samples: Replay Buffer

Online Q Iteration Algorithm:

While not converged:

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

Special case with 1 gradient step per experience

Full Fitted Q Iteration Algorithm:

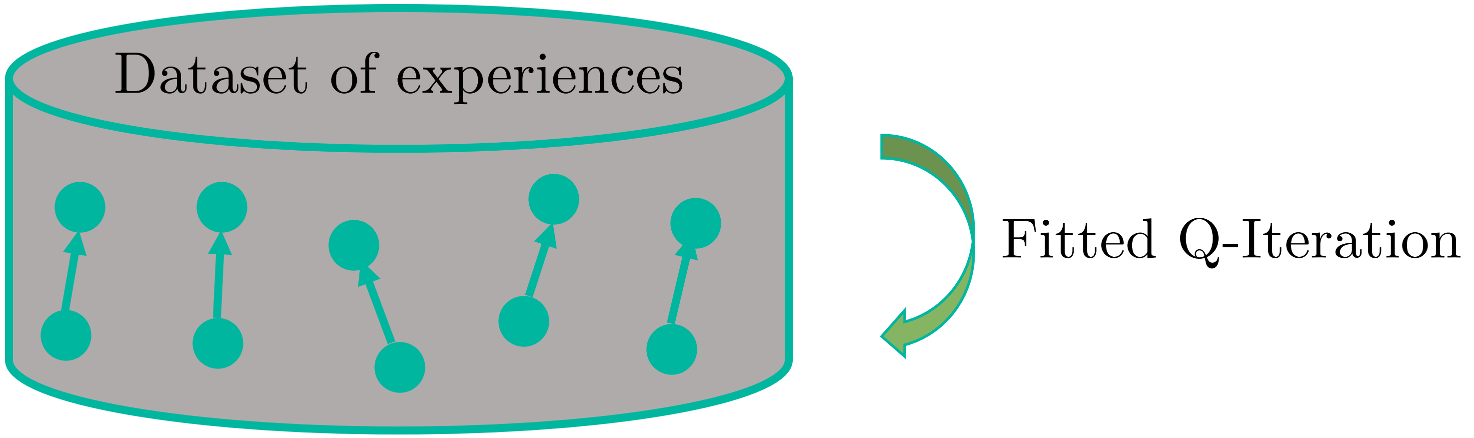
While not converged:

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

   Until data refresh:

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

Off-policy: any policy will work (some better than others)



Replay Buffers

Q-Learning with a Replay Buffer:

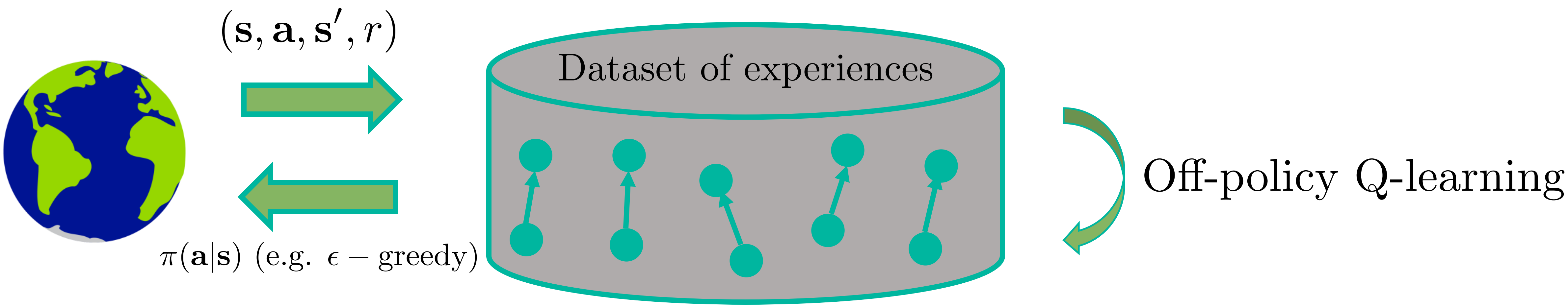
While not converged:

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

→ Multiple samples in the batch (low-variance gradient)

→ Samples are no longer correlated

→ Requires periodically feeding the replay buffer

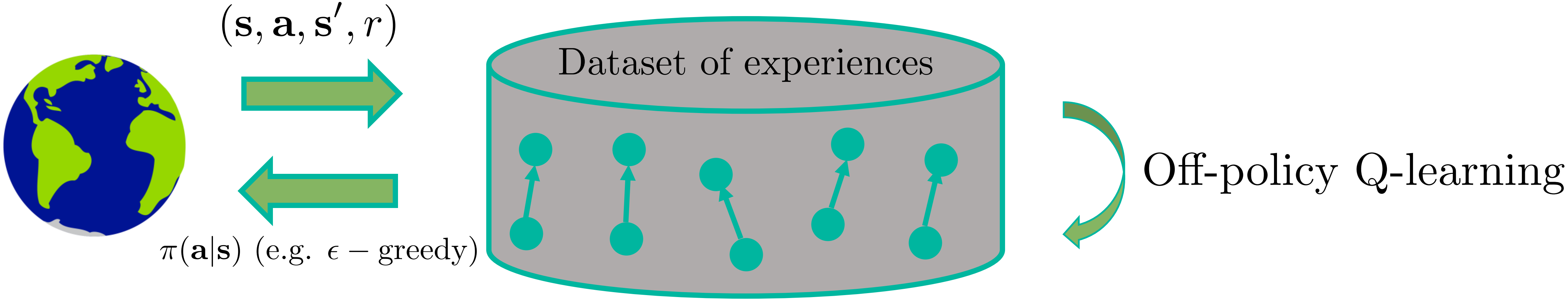


Full Algorithm

Full Q-learning with a replay buffer:

While not converged:

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



Q-Learning: Practical Issues

Online Q Iteration Algorithm:

While not converged:

1. Take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ → One experience, data are correlated!
2. Set $y_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi(\mathbf{s}_i, \mathbf{a}_i)}{d\phi}(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - y_i)$ → ~~Gradient descent?~~

Still need to address this!

Not gradient descent!

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

No gradient through this!

A Moving Target?

Full Fitted Q Iteration Algorithm:

While not converged:

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

   Until data refresh:

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

Perfectly fine regression

$\rightarrow$ Converges close to targets, not necessarily to good policy

Full Q-learning with a replay buffer:

While not converged:

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

One gradient step toward a moving target

Q-learning with target networks

Q-learning with a replay buffer and target network:

While not converged:

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

Until target refresh:

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

Until data refresh:

$N\times$

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

$K\times$

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

- Targets don't change in inner loop
- Back to resembling supervised regression!

“Classic” Deep Q-Learning algorithm

Mnih et al. Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602* (2013).

Q-learning with a replay buffer and target network:

While not converged:

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

Until target refresh:

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

Until data refresh:

$N\times$

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

$K\times$

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

“Classic” Deep Q-Learning Algorithm

While not converged:

1. Take some action $\mathbf{a}_i$, observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add 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. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi(\mathbf{s}_j, \mathbf{a}_j)}{d\phi}(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. Update ϕ': copy ϕ every N steps

$(K \rightarrow 1)$

$\rightarrow$ Much more stable than online deep Q learning

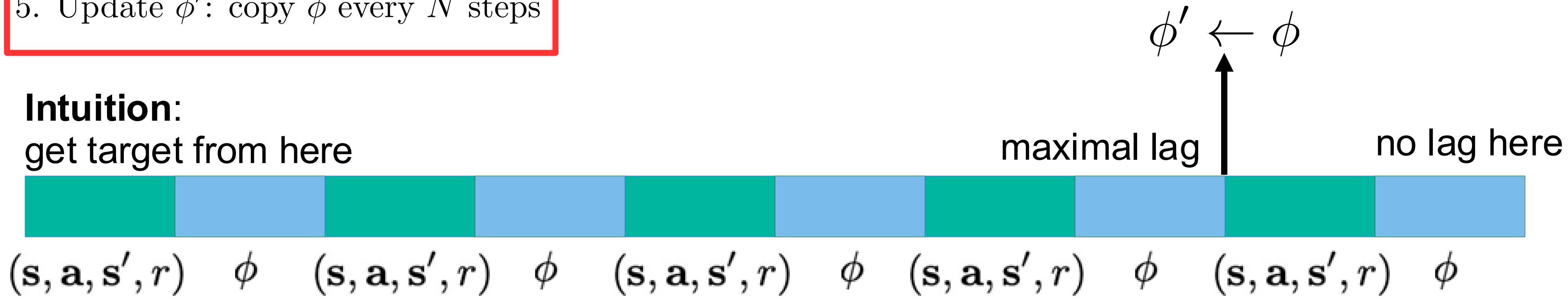
Alternate Target Network Formulation

"Classic" Deep Q-Learning Algorithm

While not converged:

1. Take some action $\mathbf{a}_i$, observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add 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. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi(\mathbf{s}_j, \mathbf{a}_j)}{d\phi}(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. Update ϕ': copy ϕ every N steps

Intuition:



Popular alternative:

5. Update ϕ' : $\phi' \leftarrow \tau\phi' + (1-\tau)\phi$ e.g. $\tau = 0.999$ works well

Process View

Q-learning with a replay buffer and target network:

While not converged:

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

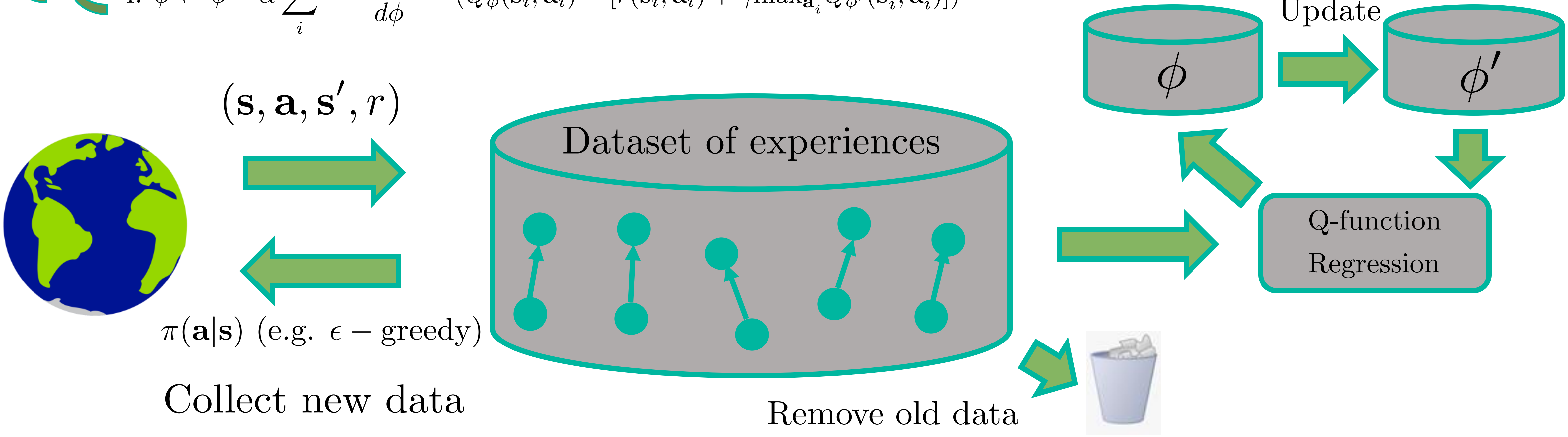
   Until target refresh:

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

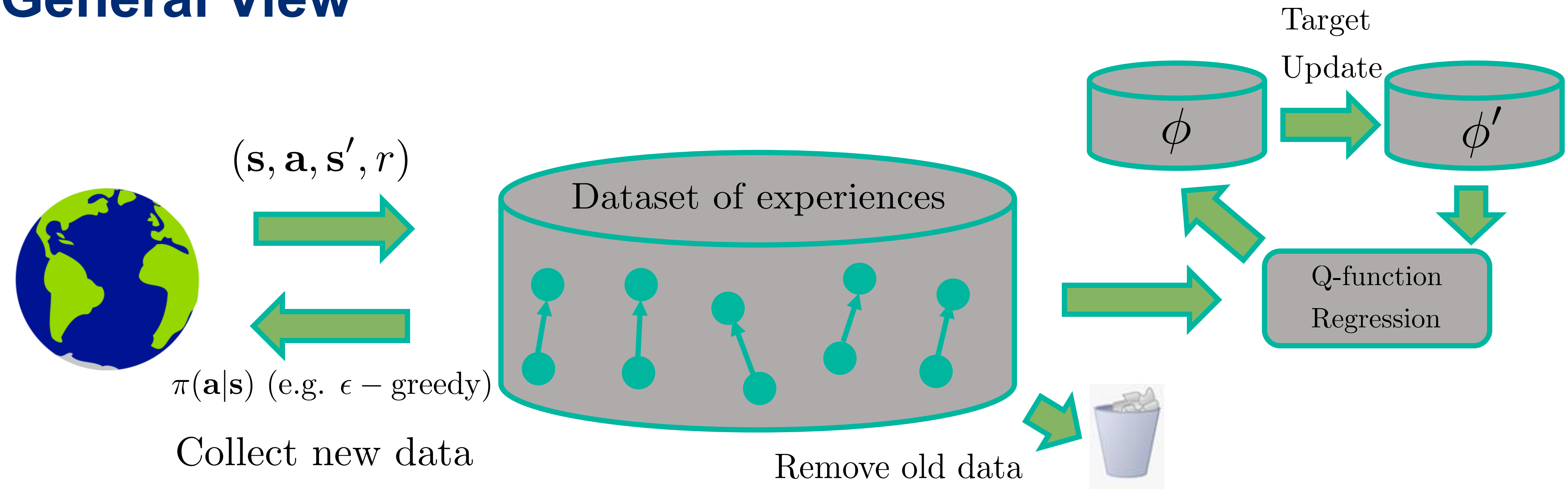
   Until data refresh:

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

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



General View

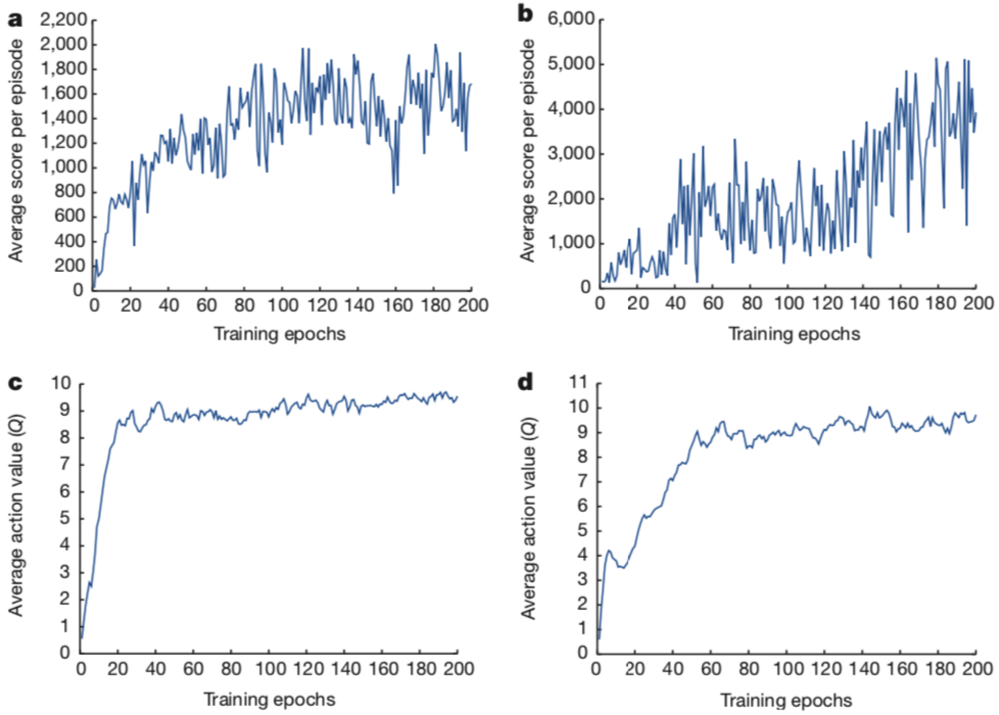


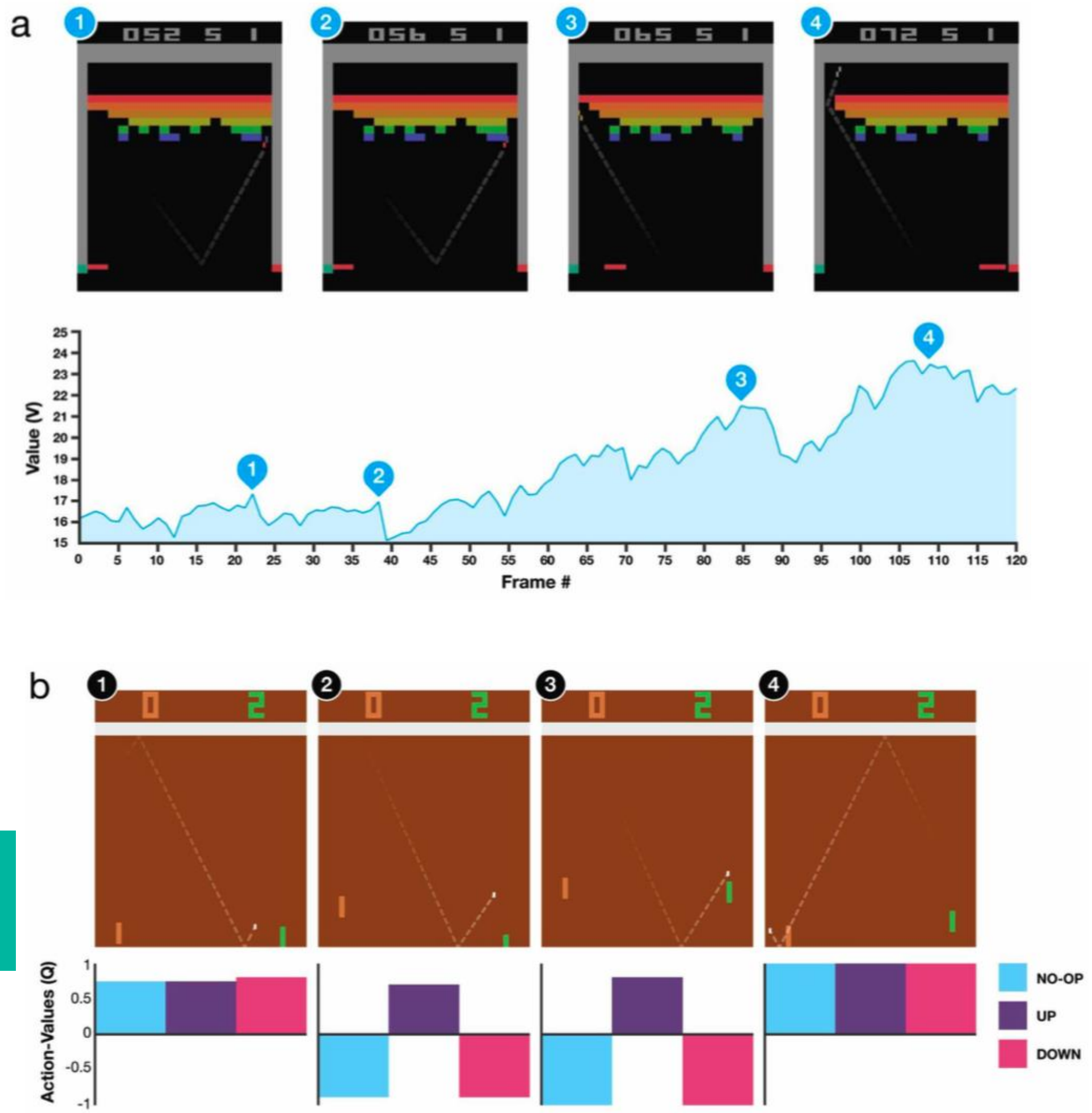
- **Fitted Q-iteration**: Q-function regression in inner loop of target update, in inner loop of data collection
- **Online Q-Learning**: Update on 1 data point, remove immediately, all processes run at same speed
- **DQN**: Data collection, Q-function regression at same speed; target update is slow

Improvements

- Double Q-Learning
- Multi-step returns

Are the Q-values accurate?

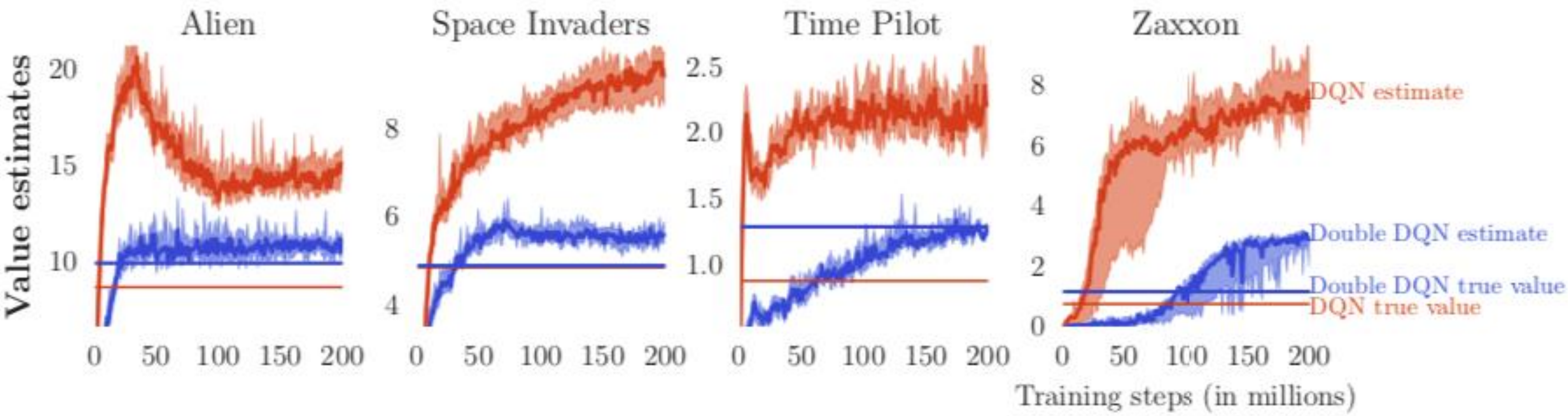




- ➢ **Increase with return**
- ➢ **Seem to qualitatively track what is going on…**

V. Mnih et al. Human-level control through deep reinforcement learning. *Nature* **518**, p. 529–533 (2015).

Overestimation of Q-values



➢ **Why is this?**

van Hasselt et al. Deep Reinforcement Learning with Double Q-Learning. *AAAI*, 2016.

Overestimation of Q values

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

Consider two random variables: $X1, X2$

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

$Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$is noisy $\implies$ $\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ overestimates the next value!

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

$\rightarrow$ Action selected according to $Q_{\phi'}$

$\rightarrow$ Value also comes from $Q_{\phi'}$

Double Q-Learning

van Hasselt et al. Deep Reinforcement Learning with Double Q-Learning. *AAAI*, 2016.

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

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

$\rightarrow$ Can we decorrelate these sources of error?

$\rightarrow$ This would remove the tendency of regression targets to be too large!

$\rightarrow$ Use different networks for choosing and evaluating value!

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

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

$\rightarrow$ "Double" Q-Learning

What two networks?

$\rightarrow$ Just use current and target networks!

$\rightarrow$ Standard Q-Learning: $y = r + \gamma Q_{\phi'}(\mathbf{s}', \text{argmax}_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}'))$

$\rightarrow$ Double Q-Learning: $y = r + \gamma Q_{\phi'}(\mathbf{s}', \text{argmax}_{\mathbf{a}'} Q_{\phi}(\mathbf{s}', \mathbf{a}'))$

$\rightarrow$ Current network for evaluating action

$\rightarrow$ Target network for evaluating value

$\rightarrow$ Decorrelates it enough!

Multi-Step Returns

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

More important if $Q_{\phi'}$ is bad (early)

More important if $Q_{\phi'}$ is good (late)

Like actor-critic, construct n-step returns:

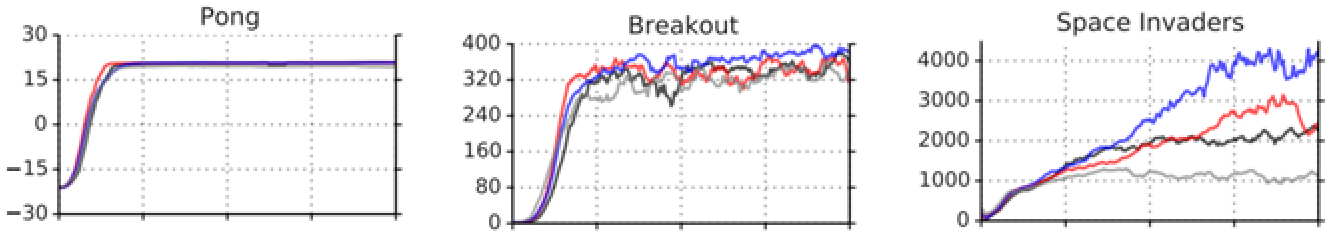
$$y_{j,t} = \sum_{t'=t}^{t+N-1} \gamma^{t'-t} r_{j,t'} + \gamma^N \max_{\mathbf{a}_{j,t+N}} Q_{\phi'}(\mathbf{s}_{j,t+N}, \mathbf{a}_{j,t+N})$$

$\rightarrow$ Another bias-variance tradeoff; $N = 4 - 10$ typically works well.

$\rightarrow$ Technically incorrect; we should be on-policy to do this.

$\rightarrow$ Works well in practice; can also force on-policy data.

Q-Learning Implementation Tips



T. Schaul, et al. "Prioritized experience replay". *arXiv:1511.05952* (2015).

- Q –Learning can be difficult to stabilize
  - ➢ Test your code on easy tasks first!
- Learning may make little progress initially
- Start with larger epsilon initially, taper over time
- Large replay buffers help
- Gradient clipping on Bellman errors may help
- Double Q learning is very helpful- should always be used
- N-step returns can also be helpful, though not always
- Use an adaptive optimizer (e.g. Adam)
- Try multiple random seeds, as results can be inconsistent

Application to Continuous Action Spaces

- Sampling outcomes
- Separating state and action dependence
- DDPG

Q-Learning with Continuous Actions

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

$\rightarrow$ How do we handle these?

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

Options
1.) Gradient-based or stochastic optimization to take max
2.) Use function class for Q that is easy to optimize w.r.t. actions
3.) Learn an approximate optimizer (DDPG)

Q-Learning with Stochastic Optimization

Search over candidate values:

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

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

- Simple
- Parallelizable
- Inaccurate
- Sometimes good enough; target doesn't have to be incredibly accurate.

Slightly Better:

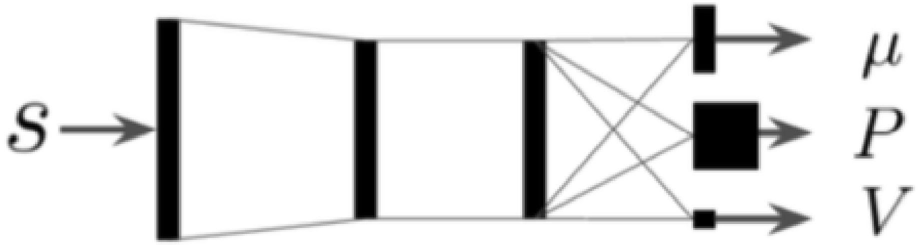
- Cross-Entropy Method (good to about 40 dimensions)
  - Fit distribution to sampled Q values, use max of distribution
  - Repeat (iterative process)
- Covariance Matrix Adaptation Evolution Strategy (CMA-ES)
  - Less simple, more effective

Easily Maximizable Q-Functions

Use a function class that is easy to optimize

- **Only has to be easy to optimize in w.r.t. actions**

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



NAF Architecture.

NAF: Normalized Advantaged Functions

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

- ➢ **Algorithm largely unchanged; remains efficient**
- ➢ **Drawback is the reduction in representational power**

S. Gu et al. Continuous Deep Q-Learning with Model-based Acceleration. *ICML* (2016).

Deep Deterministic Policy Gradient (DDPG)

- **Learn an approximate maximizer**
  - ➢ **"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 neural network $\mu_\theta(\mathbf{s})$ such that $\mu_\theta(\mathbf{s}) \approx \text{argmax}_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})$

How? Just solve $\theta \leftarrow \text{argmax}_\theta Q_\phi(\mathbf{s}, \mu_\theta(\mathbf{s}))$

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

New target: $y_j = r_j + \gamma Q_{\phi'}(\mathbf{s}'_j, \mu_\theta(\mathbf{s}'_j)) \approx r_j + \gamma Q_{\phi'}(\mathbf{s}'_j, \text{argmax}_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j))$

T.P. Lillicrap et al. Continuous Control with Deep Reinforcement Learning. *ICLR* (2016).

Deep Deterministic Policy Gradient (DDPG)

DDPG

While not converged:

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 $y_j = r_j + \gamma \text{max}_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mu_{\theta'}(\mathbf{s}'_j))$ using target nets $Q_{\phi'}$ and $\mu_{\theta'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi(\mathbf{s}_j, \mathbf{a}_j)}{d\phi}(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. $\theta \leftarrow \theta + \beta \sum_j \frac{d\mu(\mathbf{s}_j)}{d\theta} \frac{dQ_\phi(\mathbf{s}_j, \mathbf{a})}{d\mathbf{a}}$
6. Update parameters of target functions (ϕ' and θ')

T.P. Lillicrap et al. Continuous Control with Deep Reinforcement Learning. *ICLR* (2016).

DDPG: Pseudocode

Algorithm 1 Deep Deterministic Policy Gradient

```
Input: initial policy parameters θ, Q-function parameters φ, empty replay buffer D
Set target parameters equal to main parameters θ_targ ← θ, φ_targ ← φ
repeat
  Observe state s and select action a = clip(μ_θ(s) + ε, a_Low, a_High), where ε ~ N
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer D
  If s' is terminal, reset environment state.
  if it's time to update then
    for however many updates do
      Randomly sample a batch of transitions, B = {(s, a, r, s', d)} from D
      Compute targets
```

$$y(r, s', d) = r + \gamma(1 - d) Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

```
      Update Q-function by one step of gradient descent using
```

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} \left(Q_\phi(s,a) - y(r,s',d)\right)^2$$

```
      Update policy by one step of gradient ascent using
```

$$\nabla_\theta \frac{1}{|B|} \sum_{s\in B} Q_\phi(s, \mu_\theta(s))$$

```
      Update target networks with
```

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1-\rho)\theta$$

```
    end for
  end if
until convergence
```

Noise added for exploration

Parameters ϕ held constant in this step

$\rightarrow$ Off-policy

$\rightarrow$ Continuous actions

Advanced Discrete-Action Methods

- Distributional, Rainbow

Dueling Architecture

Z. Wang et al. “Dueling Network Architectures for Deep Reinforcement Learning” *arXiv:1511.06581.*

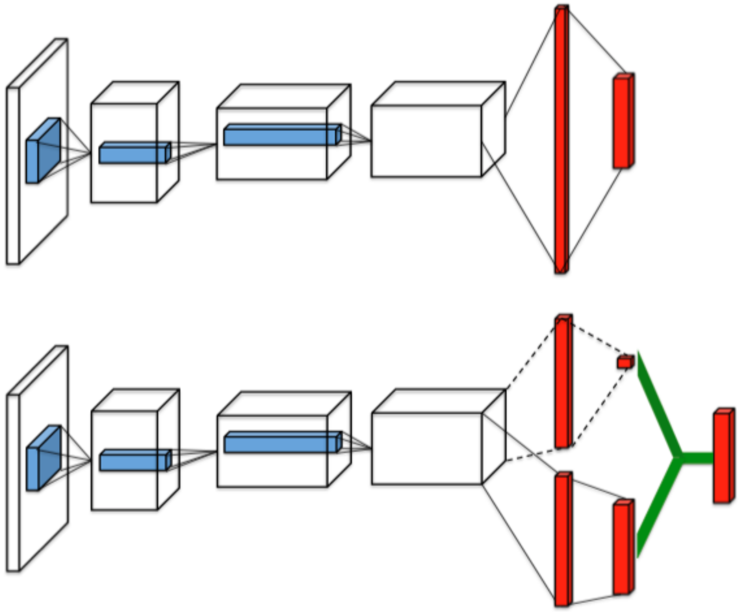
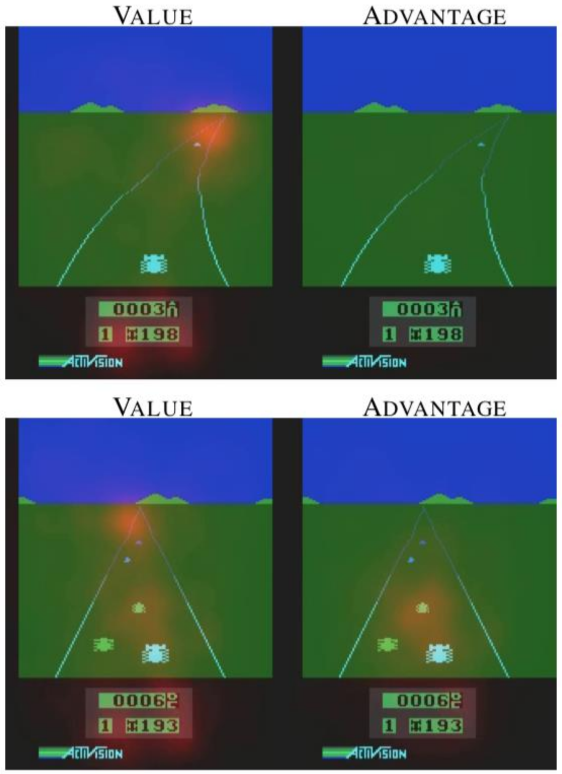


Figure 1. A popular single stream Q-network (**top**) and the dueling Q-network (**bottom**). The dueling network has two streams to separately estimate (scalar) state-value and the advantages for each action; the green output module implements equation (9) to combine them. Both networks output Q-values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



$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \max_{a' \in |\mathcal{A}|} A(s,a;\theta,\alpha)\right)$$

Intuition: Forces network to consider value and advantage separately, leading to performance gains.

Prioritized Experience Replay

T. Schaul et al. "Prioritized Experience Replay." *ICLR*, 2016.

Algorithm 1 Double DQN with proportional prioritization

Input: minibatch k, step-size η, replay period K and size N, exponents α and β, budget T.
Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
Observe S_0 and choose $A_0 \sim \pi_\theta(S_0)$
for $t = 1$ **to** T **do**
  Observe S_t, R_t, γ_t
  Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t = \max_{i<t} p_i$
  if $t \equiv 0 \mod K$ **then**
    for $j = 1$ **to** k **do**
      Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
      Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
      Compute TD-error $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
      Update transition priority $p_j \leftarrow |\delta_j|$
      Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
    end for
    Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
    From time to time copy weights into target network $\theta_{\text{target}} \leftarrow \theta$
  end if
  Choose action $A_t \sim \pi_\theta(S_t)$
end for

Preferentially sampling experiences based on TD-error Improves DQN performance.

D. Horgan et al. "Distributed Prioritized Experience Replay." *ICLR*, 2018.

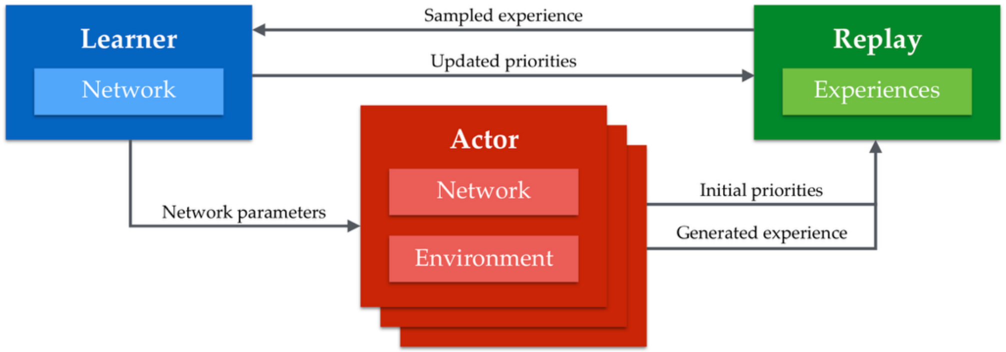
Distributed Prioritized Experience Replay (Ape-X)

Algorithm 1 Actor

procedure ACTOR(B, T) ▷ Run agent in environment instance, storing experiences.
  $\theta_0 \leftarrow$ LEARNER.PARAMETERS( ) ▷ Remote call to obtain latest network parameters.
  $s_0 \leftarrow$ ENVIRONMENT.INITIALIZE( ) ▷ Get initial state from environment.
  for $t = 1$ **to** T **do**
    $a_{t-1} \leftarrow \pi_{\theta_{t-1}}(s_{t-1})$ ▷ Select an action using the current policy.
    $(r_t, \gamma_t, s_t) \leftarrow$ ENVIRONMENT.STEP(a_{t-1}) ▷ Apply the action in the environment.
    LOCALBUFFER.ADD($(s_{t-1}, a_{t-1}, r_t, \gamma_t)$) ▷ Add data to local buffer.
    if LOCALBUFFER.SIZE( ) $\geq B$ **then** ▷ In a background thread, periodically send data to replay.
      $\tau \leftarrow$ LOCALBUFFER.GET(B) ▷ Get buffered data (e.g. batch of multi-step transitions).
      $p \leftarrow$ COMPUTEPRIORITIES(τ) ▷ Calculate priorities for experience (e.g. absolute TD error).
      REPLAY.ADD(τ, p) ▷ Remote call to add experience to replay memory.
    end if
    PERIODICALLY($\theta_t \leftarrow$ LEARNER.PARAMETERS()) ▷ Obtain latest network parameters.
  end for
end procedure

Algorithm 2 Learner

procedure LEARNER(T) ▷ Update network using batches sampled from memory.
  $\theta_0 \leftarrow$ INITIALIZENETWORK( )
  for $t = 1$ **to** T **do** ▷ Update the parameters T times.
    $id, \tau \leftarrow$ REPLAY.SAMPLE( ) ▷ Sample a prioritized batch of transitions (in a background thread).
    $l_t \leftarrow$ COMPUTELOSS($\tau; \theta_t$) ▷ Apply learning rule; e.g. double Q-learning or DDPG
    $\theta_{t+1} \leftarrow$ UPDATEPARAMETERS($l_t; \theta_t$)
    $p \leftarrow$ COMPUTEPRIORITIES( ) ▷ Calculate priorities for experience, (e.g. absolute TD error).
    REPLAY.SETPRIORITY(id, p) ▷ Remote call to update priorities.
    PERIODICALLY(REPLAY.REMOVETOFIT()) ▷ Remove old experience from replay memory.
  end for
end procedure



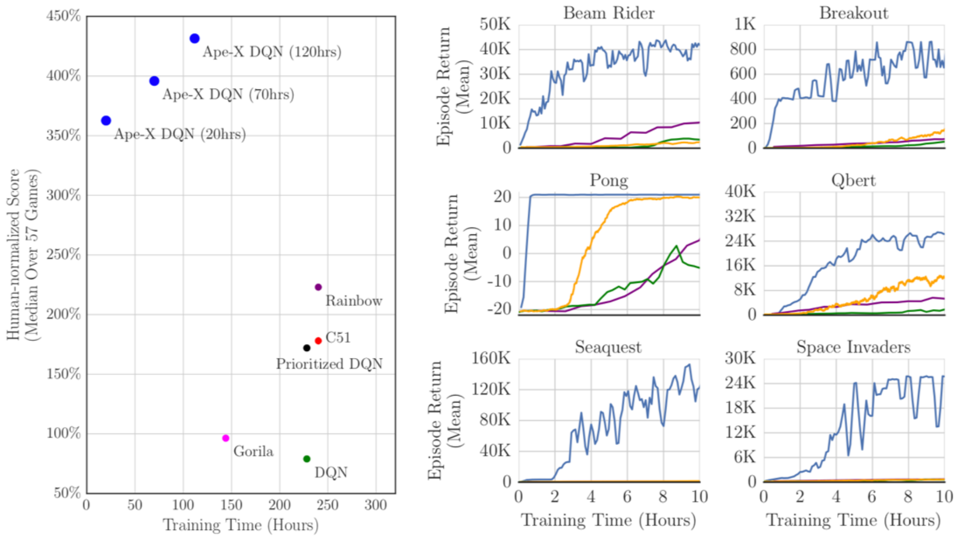


Figure 2: Left: Atari results aggregated across 57 games, evaluated from random no-op starts. Right: Atari training curves for selected games, against baselines. Blue: Ape-X DQN with 360 actors; Orange: A3C; Purple: Rainbow; Green: DQN. See appendix for longer runs over all games.

Distributional Reinforcement Learning: C51

M. G. Bellemare et al. “A Distributional Perspective on Reinforcement Learning”
arXiv:1707.06887

$$Q(x,a) = \mathrm{E}[R(x,a)] + \gamma \mathrm{E}[Q(X',A')]$$

$$\rightarrow Z(x,a) = R(x,a) + \gamma Z(X',A')$$

Algorithm 1 Categorical Algorithm

input A transition $x_t, a_t, r_t, x_{t+1}, \gamma_t \in [0,1]$
  $Q(x_{t+1}, a) := \sum_i z_i p_i(x_{t+1}, a)$
  $a^* \leftarrow \arg\max_a Q(x_{t+1}, a)$
  $m_i = 0, \quad i \in 0, \ldots, N-1$
  for $j \in 0, \ldots, N-1$ **do**
    # Compute the projection of $\hat{\mathcal{T}} z_j$ onto the support $\{z_i\}$
    $\hat{\mathcal{T}} z_j \leftarrow [r_t + \gamma_t z_j]_{V_{\text{MIN}}}^{V_{\text{MAX}}}$
    $b_j \leftarrow (\hat{\mathcal{T}} z_j - V_{\text{MIN}})/\Delta z$ # $b_j \in [0, N-1]$
    $l \leftarrow \lfloor b_j \rfloor, u \leftarrow \lceil b_j \rceil$
    # Distribute probability of $\hat{\mathcal{T}} z_j$
    $m_l \leftarrow m_l + p_j(x_{t+1}, a^*)(u - b_j)$
    $m_u \leftarrow m_u + p_j(x_{t+1}, a^*)(b_j - l)$
  end for
output $-\sum_i m_i \log p_i(x_t, a_t)$ # Cross-entropy loss

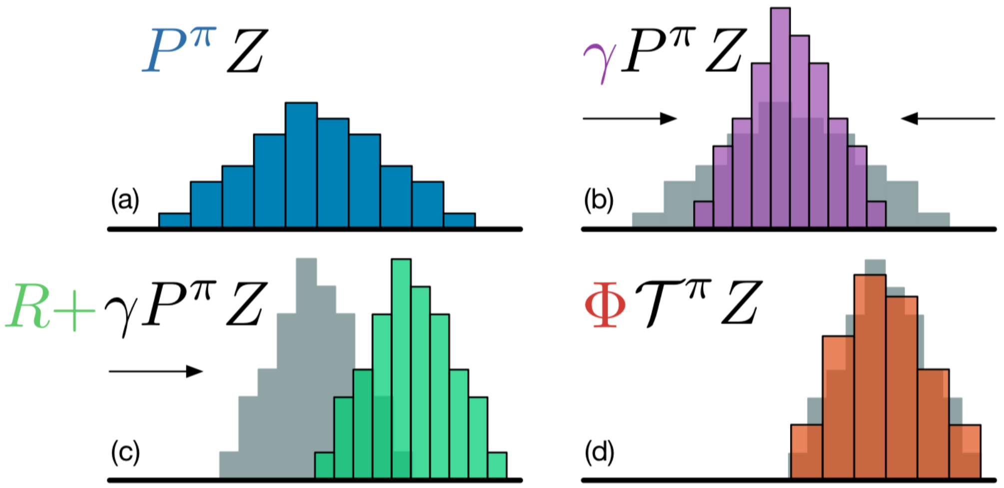
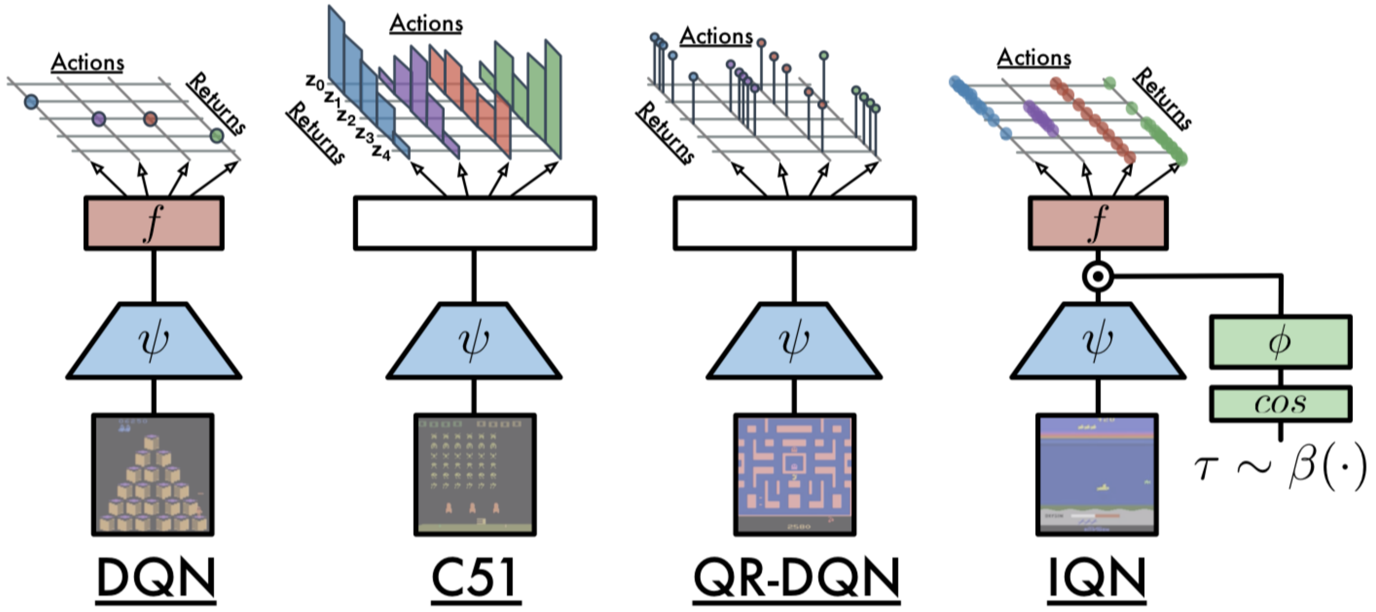


Figure 1. A distributional Bellman operator with a deterministic reward function: (a) Next state distribution under policy π, (b) Discounting shrinks the distribution towards 0, (c) The reward shifts it, and (d) Projection step (Section 4).

➢ **Despite lack of guarantees, often works well!**

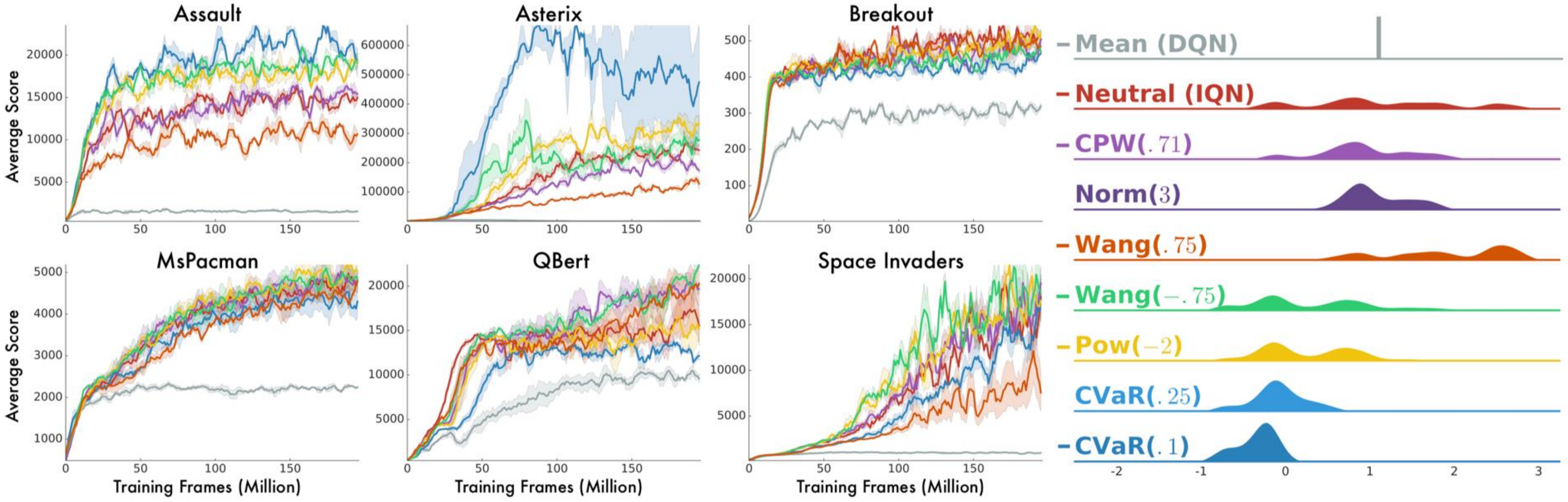
Distributional DRL: Implicit Quantile Networks (IQN)

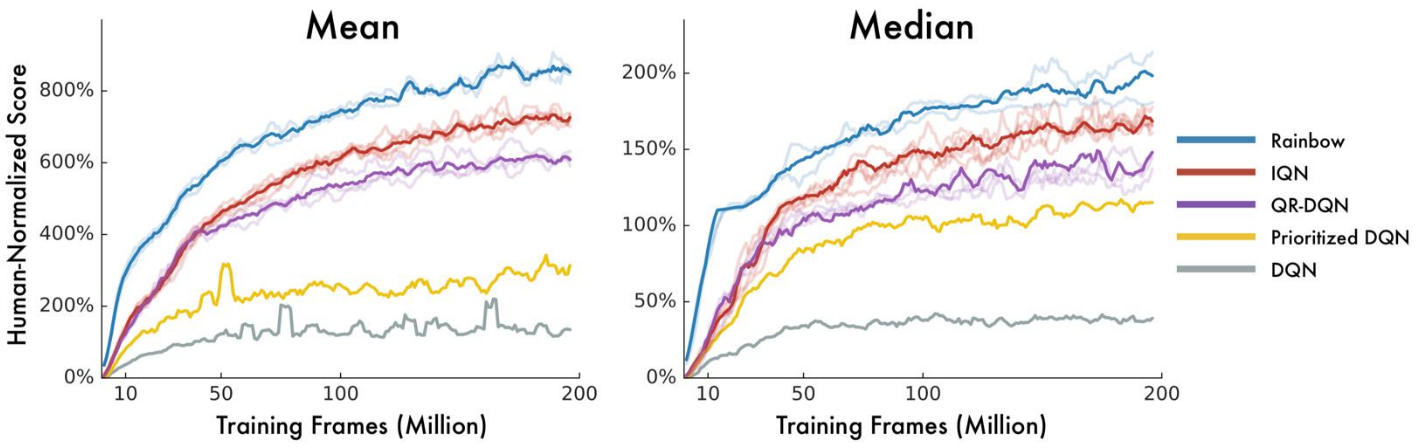
- Learns probability distribution of returns via quantile regression.
  - ➢ Quantile: fixed portion of a probability distribution
  - ➢ Uses ∞-Wasserstein metric for distribution distance (contraction)
  - ➢ Quantile regression loss is on pairwise TD errors
- Improve resolution of estimate with increased network capacity.
- Can be used to expand exploration strategies from basic epsilon-greedy via inclusion of distortion risk measures.



W. Dabney et al. "Implicit Quantile Networks for Distributional Reinforcement Learning" *arXiv:1806.06923.*

Distributional DRL: Implicit Quantile Networks (IQN)

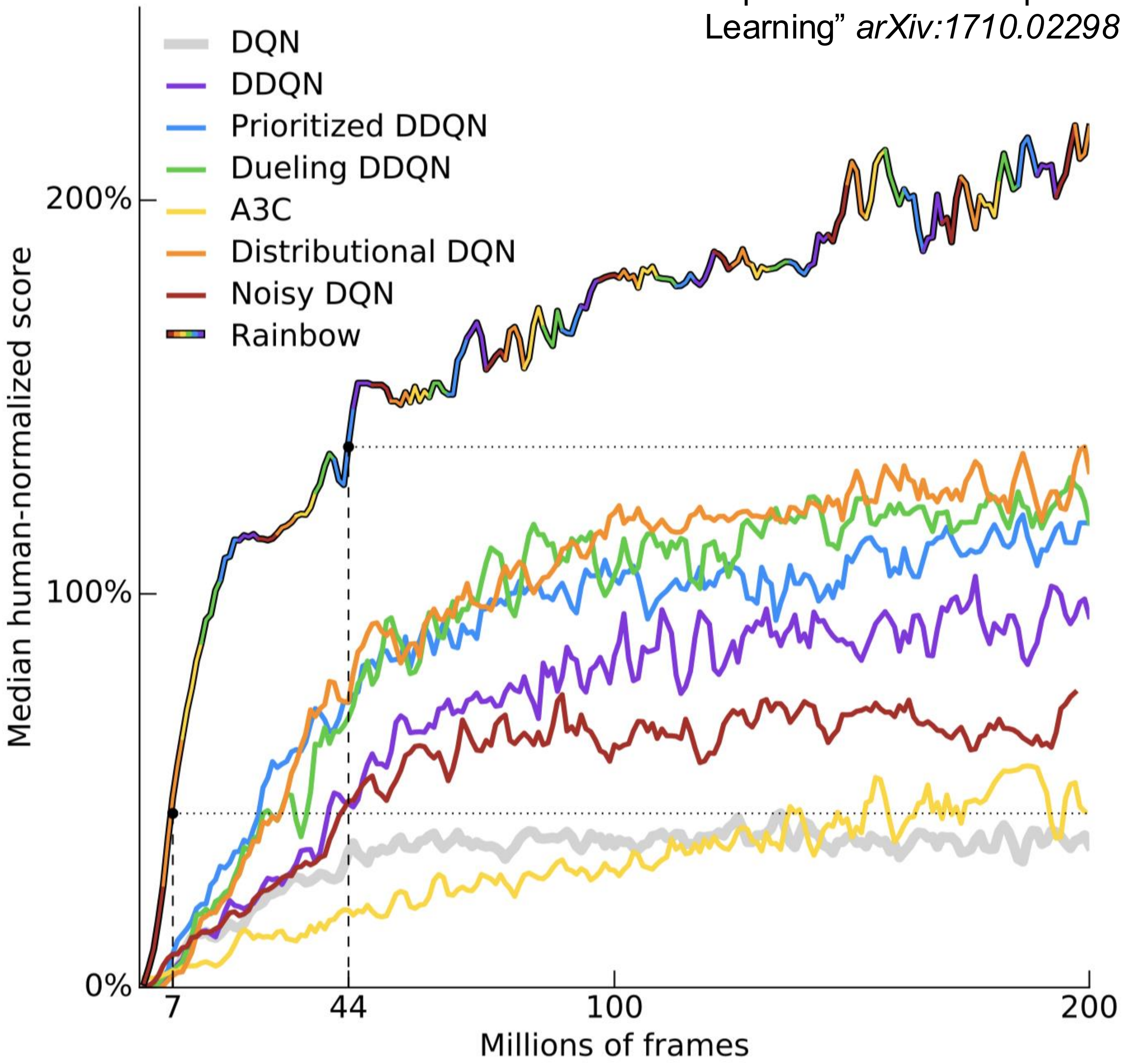




Rainbow Q-Learning

M. Hessel et al. "Rainbow: Combining Improvements in Deep Reinforcement Learning" *arXiv:1710.02298*

- **Combines numerous improvements to basic DQN algorithm to produce one very strong version.**
- **Elements:**
  - ➢ **Double**
  - ➢ **Multi-step returns**
  - ➢ **Dueling**
  - ➢ **Prioritized Replay Buffer**
  - ➢ **Distributional**
  - ➢ **Noisy Nets**



Thanks!

Enrique Mallada
mallada@jhu.edu
http://mallada.ece.jhu.edu