

# Lecture 6: Deep Reinforcement Learning

## Value Function Methods - Part II

Chunlin Chen & Zhi Wang

Department of Control and Systems Engineering  
Nanjing University

Dec. 17th, 2019

# Table of Contents

## 1 Q-learning with deep neural networks

- Replay buffers, target networks
- Double Q-learning, multi-step returns, continuous actions
- Tips and examples

## 2 Theories of value function methods

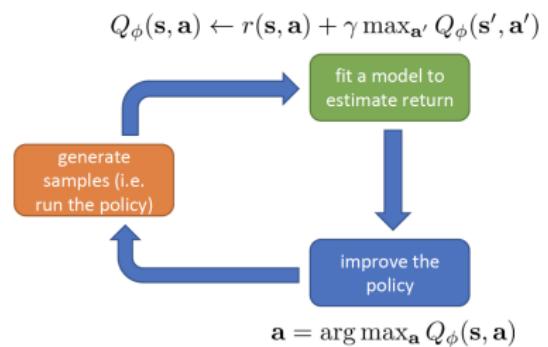
# Contents and Goals

- How we can make Q-learning work with deep networks
  - Use replay buffers, separate target networks
- Tricks for improving Q-learning in practice
  - Double Q-learning, multi-step Q-learning
- Continuous Q-learning methods
- Goals
  - Understand how to implement Q-learning so that it can be used with complex function approximators
  - Understand how to extend Q-learning to continuous actions

# Recall: Q-learning

full fitted Q-iteration algorithm:

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



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

# Problem 1: Correlated samples in MDPs

online Q iteration algorithm:

- 
1. take some action  $\mathbf{a}_i$  and observe  $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
  2.  $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
  3.  $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$
- these are correlated!
- isn't this just gradient descent? that converges, right?

Q-learning is *not* gradient descent!

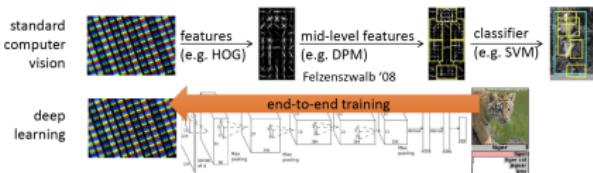
$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$$

no gradient through target value

# Recall: Supervised learning vs. Sequential decision making

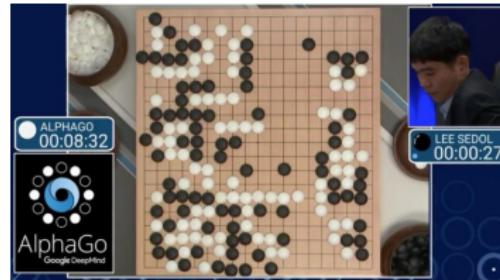
## Supervised learning

- Samples are independent and identically distributed (i.i.d.)
- Given an input, map an optimal output



## Reinforcement learning

- Samples are not i.i.d., temporally co-related
- Given an initial state, find a sequence of optimal actions



# Correlated samples in online Q-learning

online Q iteration algorithm:

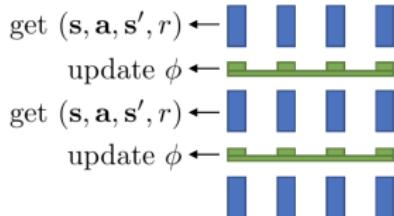


1. take some action  $\mathbf{a}_i$  and observe  $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
2.  $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i,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

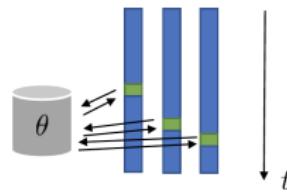
- sequential states are strongly correlated  
- target value is always changing



synchronized parallel Q-learning



asynchronous parallel Q-learning



# Another solution: replay buffers

online Q iteration algorithm:

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

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

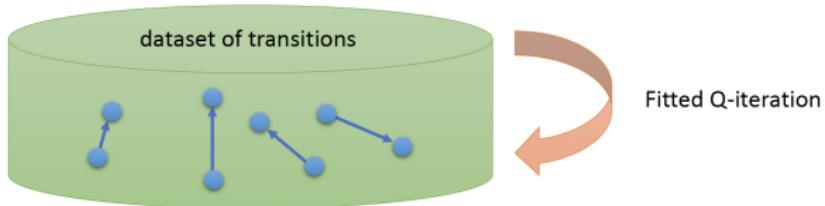
full fitted Q-iteration algorithm:

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

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

**just load data from a buffer here**

**still use one gradient step**



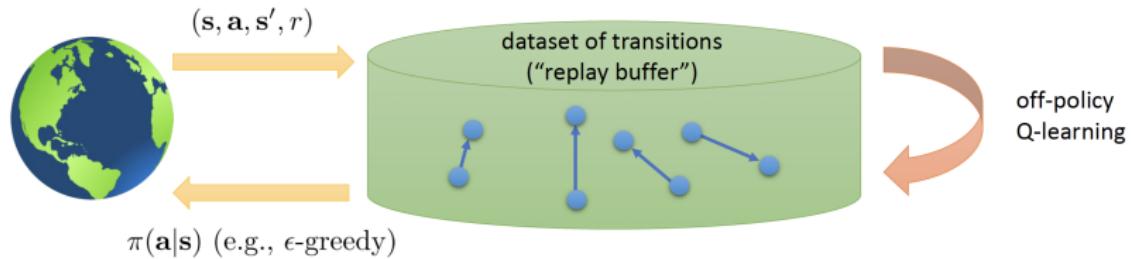
# Another solution: replay buffers

Q-learning with a replay buffer:

1. sample a batch  $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$  from  $\mathcal{B}$
  2.  $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$
- + samples are no longer correlated  
+ multiple samples in the batch (low-variance gradient)

but where does the data come from?

need to periodically feed the replay buffer...

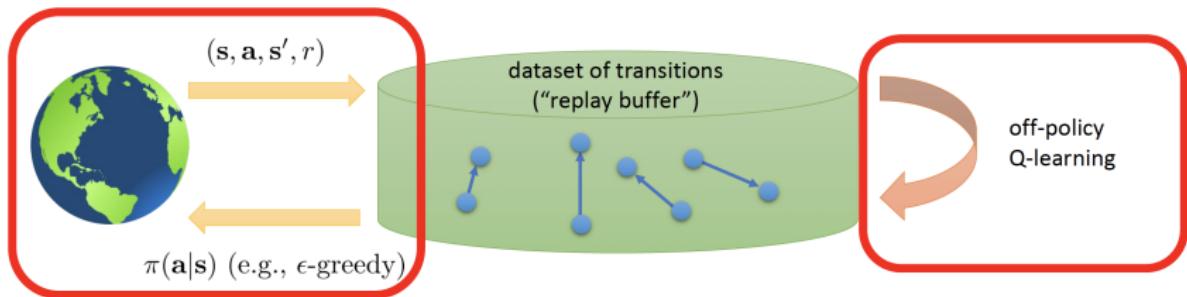


# Putting it together

full Q-learning with replay buffer:

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

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



## Problem 2: Moving target in the Bellman equation

online Q iteration algorithm:

- 
1. take some action  $\mathbf{a}_i$  and observe  $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$
  2.  $\mathbf{y}_i = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
  3.  $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i)$
- these are correlated!  
use replay buffer

Q-learning is *not* gradient descent!

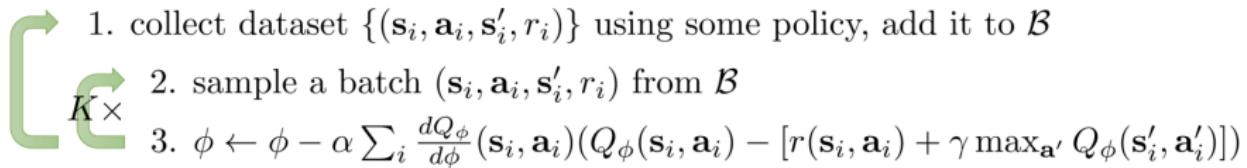
$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - (r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)))$$

no gradient through target value

This is still a  
problem!

# Q-learning and regression

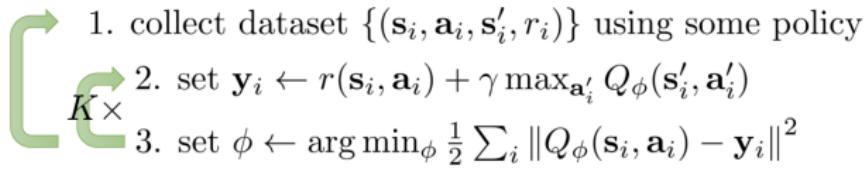
full Q-learning with replay buffer:

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

---

**one gradient step, moving target**

full fitted Q-iteration algorithm:

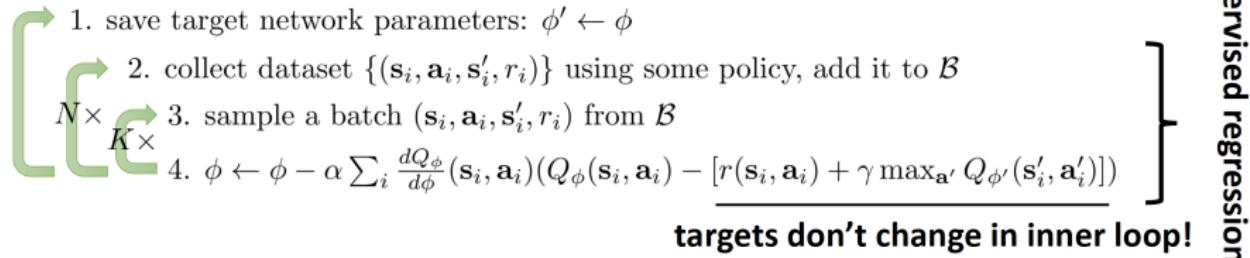
- 
1. collect dataset  $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$  using some policy
  2. set  $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'_i} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
  3. set  $\phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i,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

---

**perfectly well-defined, stable regression**

# Q-learning with target networks

Q-learning with replay buffer and target network:

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

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

Q-learning with replay buffer and target network:

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

“classic” deep Q-learning algorithm:

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

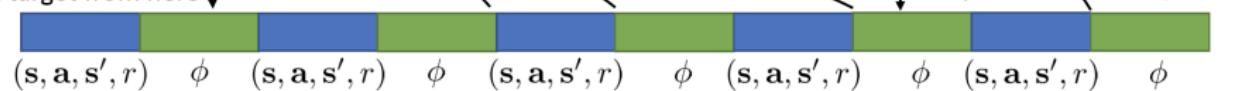
# Alternative target network

“classic” deep Q-learning algorithm:

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

**Intuition:**

get target from here



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

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

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

# Fitted Q-iteration and Q-learning

Q-learning with replay buffer and target network:

DQN:  $N = 1, K = 1$

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

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

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

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

Fitted Q-learning (written similarly as above):

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

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

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

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

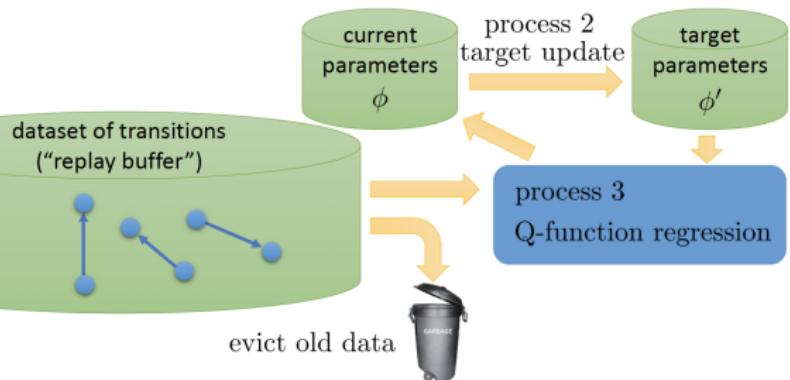
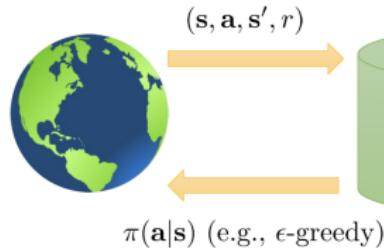
just SGD

# A more general view

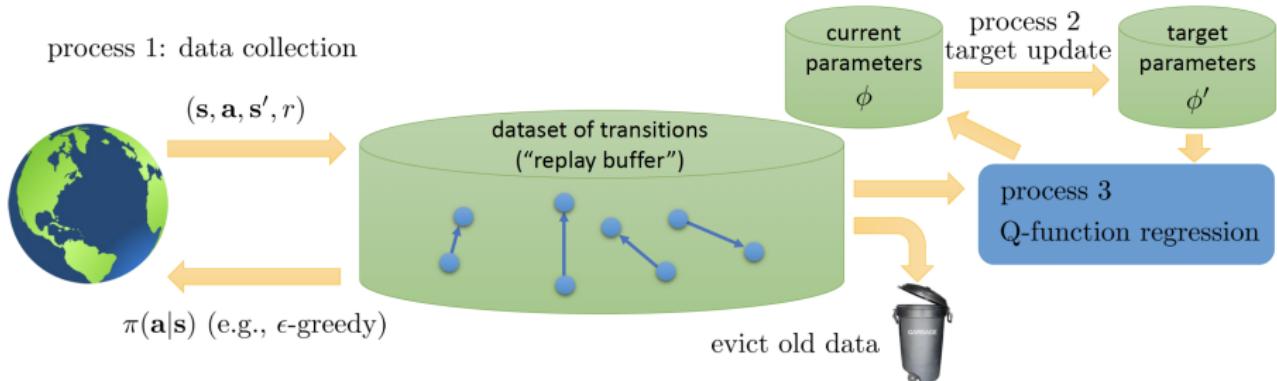
Q-learning with replay buffer and target network:

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

process 1: data collection



# A more general view



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

# Table of Contents

## 1 Q-learning with deep neural networks

- Replay buffers, target networks
- Double Q-learning, multi-step returns, continuous actions
- Tips and examples

## 2 Theories of value function methods

# Overestimation in Q-learning

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



this last term is the problem

imagine we have two random variables:  $X_1$  and  $X_2$

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

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

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

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

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

# Double Q-learning

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

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

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

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

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

“double” Q-learning: use two networks:

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

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

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

# Double Q-learning in practice

where to get two Q-functions?

just use the current and target networks!

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

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

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

still use target network to evaluate value!

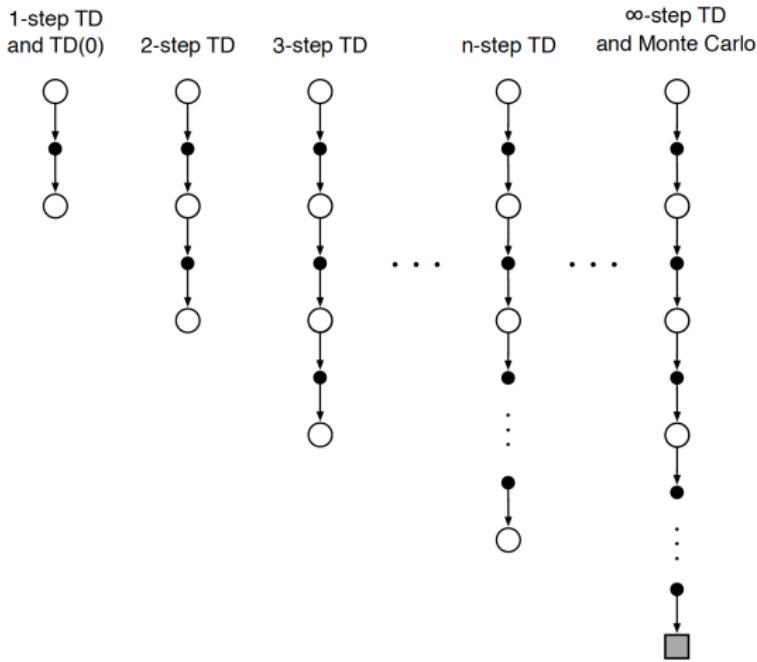
## $n$ -step bootstrapping: Combine MC and one-step TD

- Neither MC or one-step TD is always the best, we generalize both methods so that one can shift from one to the other smoothly as needed to meet the demands of a particular task
- One-step TD: In many applications, one wants to be able to update the action very fast to take into account anything that has changed
- However, bootstrapping works best if it is over a length of time in which a significant and recognizable state change has occurred

$n = 1$	$n$ -step TD	$n = \infty$
$\text{TD}(0)$	$\leftrightarrow$	$MC$

# $n$ -step TD prediction

- Perform an update based on an intermediate number of rewards, more than one, but less than all of them until termination



## Recall MC and TD(0) updates

- In MC updates, the target is the **complete return**

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t+1} R_T$$

$$\begin{aligned} V(S_t) &\leftarrow V(S_t) + \alpha[\textcolor{red}{G_t} - V(S_t)] \\ &= V(S_t) + \alpha[\textcolor{red}{R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t+1} R_T} - V(S_t)] \end{aligned}$$

- In TD(0) updates, the target is the **one-step return**

$$G_{t:t+1} = R_{t+1} + \gamma V(S_{t+1})$$

$$\begin{aligned} V(S_t) &\leftarrow V(S_t) + \alpha[\textcolor{red}{G_{t:t+1}} - V(S_t)] \\ &= V(S_t) + \alpha[\textcolor{red}{R_{t+1} + \gamma V(S_{t+1})} - V(S_t)] \end{aligned}$$

## $n$ -step TD update rule

- For  $n$ -step TD, set the target as the  **$n$ -step return**

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n})$$

- All  $n$ -step returns can be considered approximations to the complete return, truncated after  $n$  steps and then corrected for the remaining missing terms by  $V(S_{t+n})$

$$V(S_t) \leftarrow V(S_t) + \alpha[G_{t:t+n} - V(S_t)]$$

$$= V(S_t) + \alpha[R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) - V(S_t)]$$

# Deep Q-learning with $n$ -step bootstrapping

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

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

where does the signal come from?

Q-learning does this: max bias, min variance

remember this?

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

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

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

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

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

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

$N$ -step return estimator

# Deep Q-learning with $n$ -step bootstrapping

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

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

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

+ less biased target values when Q-values are inaccurate

+ typically faster learning, especially early on

- only actually correct when learning on-policy

why?

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

(not an issue when  $N = 1$ )

how to fix?

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

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

# Q-learning with continuous actions

What's the problem with continuous actions?

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

this max

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

this max  
particularly problematic (inner loop of training)

How do we perform the max?

Option 1: optimization

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

# Q-learning with stochastic optimization

Simple solution:

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

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

- + dead simple
- + efficiently parallelizable
- not very accurate

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

More accurate solution:

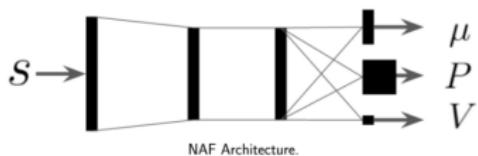
- cross-entropy method (CEM)
  - simple iterative stochastic optimization
- CMA-ES
  - substantially less simple iterative stochastic optimization

works OK, for up to about 40 dimensions

# Easily maximizable Q-functions

Option 2: use function class that is easy to optimize

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



## NAF: Normalized Advantage Functions

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

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

# Q-learning with continuous actions

Option 3: learn an approximate maximizer

DDPG (Lillicrap et al., ICLR 2016)

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

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

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

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

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

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

# Deep deterministic policy gradient (DDPG)

## Option 3: learn an approximate maximizer

DDPG:

- 
1. take some action  $\mathbf{a}_i$  and observe  $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ , add it to  $\mathcal{B}$
  2. sample mini-batch  $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$  from  $\mathcal{B}$  uniformly
  3. compute  $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mu_{\theta'}(\mathbf{s}'_j))$  using *target* nets  $Q_{\phi'}$  and  $\mu_{\theta'}$
  4.  $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
  5.  $\theta \leftarrow \theta + \beta \sum_j \frac{d\mu}{d\theta}(\mathbf{s}_j) \frac{dQ_\phi}{d\mathbf{a}}(\mathbf{s}_j, \mathbf{a})$
  6. update  $\phi'$  and  $\theta'$  (e.g., Polyak averaging)

# Recall: Actor-critic algorithms

batch actor-critic algorithm:

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

online actor-critic algorithm:

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

# DDPG vs. Actor-critic

- DDPG

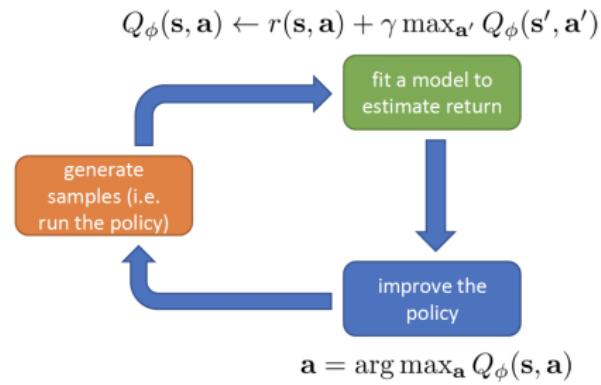
- The actor: approximate the optimal policy  $a = \arg \max Q_\phi(s, a)$
- The critic: approximate the optimal action-value function  $Q_\phi^*$
- Off-policy, more sample efficient

- Actor-critic

- The actor: approximate the current policy  $a \sim \pi(a|s)$
- The critic: approximate the state-value function  $V_\phi^\pi$
- On-policy

# Review

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



# Table of Contents

## 1 Q-learning with deep neural networks

- Replay buffers, target networks
- Double Q-learning, multi-step returns, continuous actions
- Tips and examples

## 2 Theories of value function methods

# Simple practical tips for Q-learning

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

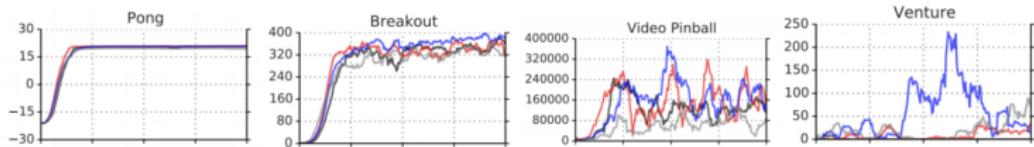


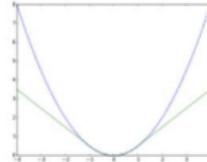
Figure: From T. Schaul, J. Quan, I. Antonoglou, and D. Silver. "Prioritized experience replay". [arXiv preprint arXiv:1511.05952 \(2015\)](https://arxiv.org/abs/1511.05952), Figure 7

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

# Advanced tips for Q-learning

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

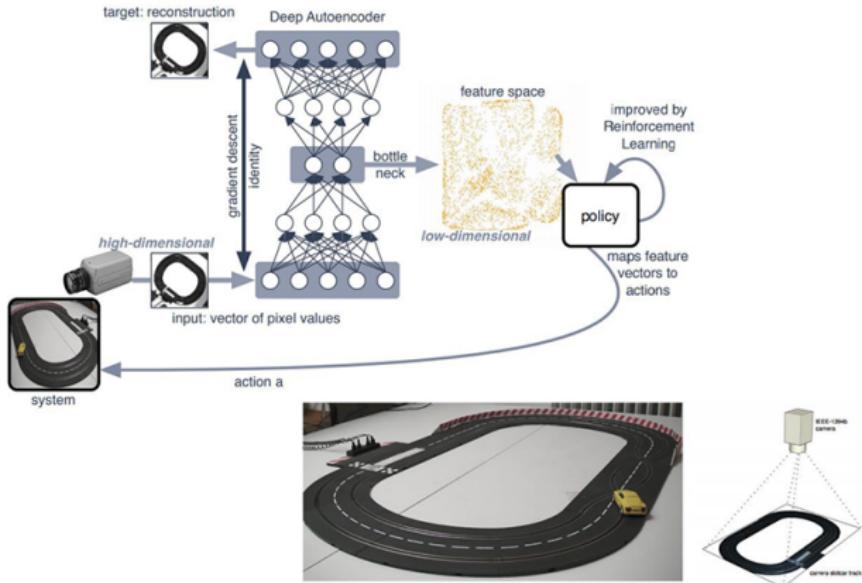
$$L(x) = \begin{cases} x^2/2 & \text{if } |x| \leq \delta \\ \delta|x| - \delta^2/2 & \text{otherwise} \end{cases}$$



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

# Fitted Q-iteration in a latent space

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



# Q-learning with convolutional networks

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

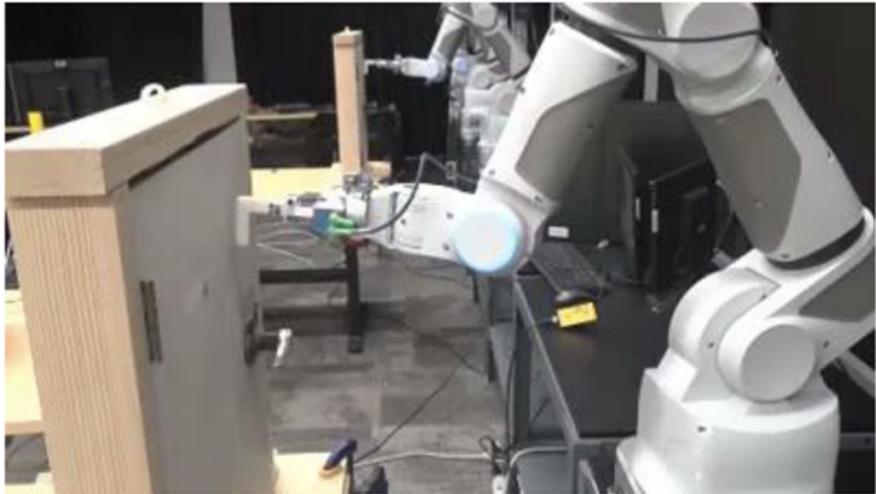


# Q-learning with continuous actions

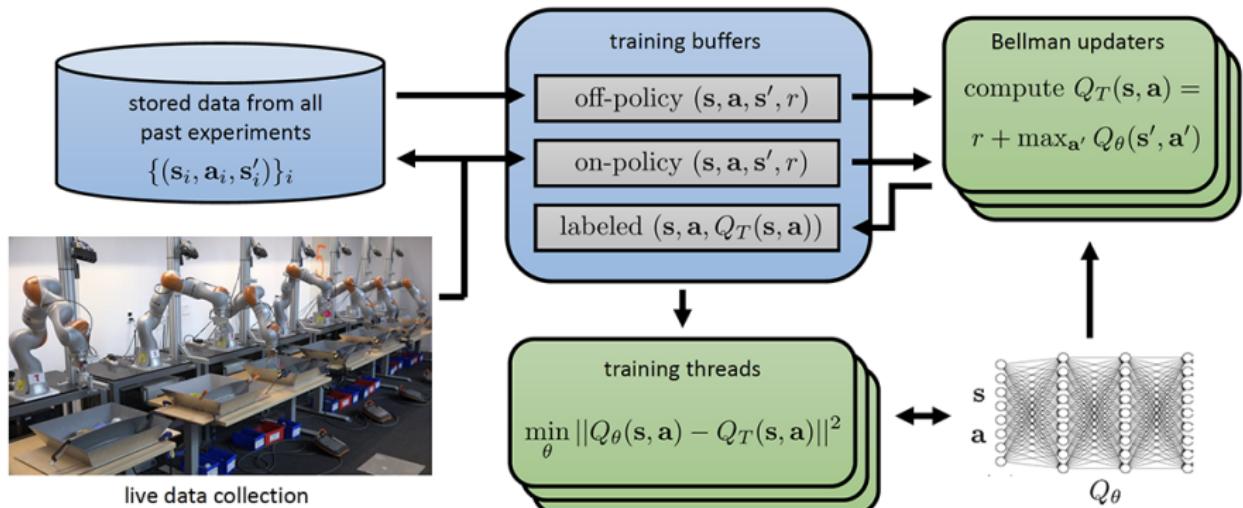
- “Continuous control with deep reinforcement learning,” Lillicrap et al. 2015.
- Continuous actions with maximizer network
- Uses replay buffer and target network (with Polyak averaging)
- One-step backup
- One gradient step per simulator step

# Q-learning on a real robot

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



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



Kalashnikov, Irpan, Pastor, Ibarz, Herzog, Jang, Quillen, Holly, Kalakrishnan, Vanhoucke, Levine. QT-Opt: Scalable Deep Reinforcement Learning of Vision-Based Robotic Manipulation Skills

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

# Table of Contents

## 1 Q-learning with deep neural networks

- Replay buffers, target networks
- Double Q-learning, multi-step returns, continuous actions
- Tips and examples

## 2 Theories of value function methods

# Value iteration theory in tabular cases

- Bellman optimality equation

$$v_*(s) = \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma v_*(s')]$$

- Value iteration

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma v_k(s')]$$

- Turn Bellman optimality equation into an update rule
- Directly approximate the optimal state-value function,  $v_*$
- For arbitrary  $v_0$ , the sequence  $\{v_k\}$  converges to  $v_*$  under the same conditions that guarantee the existence of  $v_*$

# Value function learning theory

value iteration algorithm:

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

does it converge? and if so, to what?

0.2	0.3	0.4	0.3
0.3	0.3	0.5	0.3
0.4	0.4	0.6	0.4
0.5	0.5	0.7	0.5

stacked vector of rewards at all states for action  $\mathbf{a}$   
define an operator  $\mathcal{B}$ :  $\mathcal{B}V = \max_{\mathbf{a}} r_{\mathbf{a}} + \gamma \mathcal{T}_{\mathbf{a}}V$

matrix of transitions for action  $\mathbf{a}$  such that  $\mathcal{T}_{\mathbf{a}, i, j} = p(\mathbf{s}' = i | \mathbf{s} = j, \mathbf{a})$

$V^*$  is a *fixed point* of  $\mathcal{B}$

$$V^*(\mathbf{s}) = \max_{\mathbf{a}} r(\mathbf{s}, \mathbf{a}) + \gamma E[V^*(\mathbf{s}')], \text{ so } V^* = \mathcal{B}V^*$$

always exists, is always unique, always corresponds to the optimal policy

...but will we reach it?

# Value function learning theory

value iteration algorithm:

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

$V^*$  is a *fixed point* of  $\mathcal{B}$

$$V^*(\mathbf{s}) = \max_{\mathbf{a}} r(\mathbf{s}, \mathbf{a}) + \gamma E[V^*(\mathbf{s}')], \text{ so } V^* = \mathcal{B}V^*$$

0.2	0.3	0.4	0.3
0.3	0.3	0.5	0.3
0.4	0.4	0.6	0.4
0.5	0.5	0.7	0.5

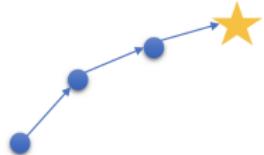
we can prove that value iteration reaches  $V^*$  because  $\mathcal{B}$  is a *contraction*

contraction: for any  $V$  and  $\bar{V}$ , we have  $\|\mathcal{B}V - \mathcal{B}\bar{V}\|_\infty \leq \underline{\gamma} \|\mathcal{V} - \bar{V}\|_\infty$

gap always gets smaller by  $\gamma$ !  
(with respect to  $\infty$ -norm)

what if we choose  $V^*$  as  $\bar{V}$ ?  $\mathcal{B}V^* = V^*$ !

$$\|\mathcal{B}V - V^*\|_\infty \leq \gamma \|V - V^*\|_\infty$$



# Non-tabular value function learning

value iteration algorithm (using  $\mathcal{B}$ ):

$$\text{C} \quad 1. \quad V \leftarrow \mathcal{B}V$$

fitted value iteration algorithm (using  $\mathcal{B}$  and  $\Pi$ ):

$$\text{C} \quad 1. \quad V \leftarrow \Pi \mathcal{B}V$$

fitted value iteration algorithm:

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

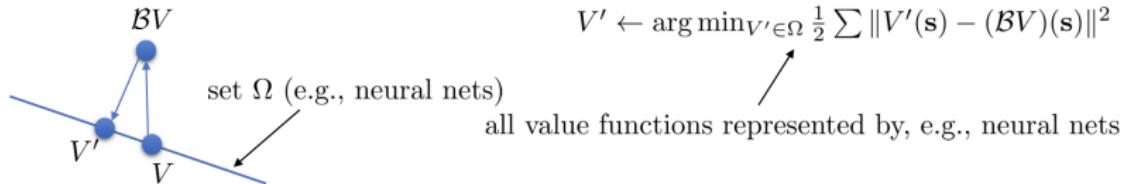
what does this do?

define new operator  $\Pi$ :  $\Pi V = \arg \min_{V' \in \Omega} \frac{1}{2} \sum \|V'(\mathbf{s}) - V(\mathbf{s})\|^2$

$\Pi$  is a *projection* onto  $\Omega$  (in terms of  $\ell_2$  norm)

updated value function

$$V' \leftarrow \arg \min_{V' \in \Omega} \frac{1}{2} \sum \|V'(\mathbf{s}) - (\mathcal{B}V)(\mathbf{s})\|^2$$

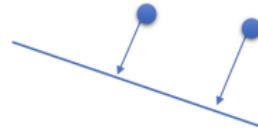


# Non-tabular value function learning

fitted value iteration algorithm (using  $\mathcal{B}$  and  $\Pi$ ):

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

$\mathcal{B}$  is a contraction w.r.t.  $\infty$ -norm (“max” norm)

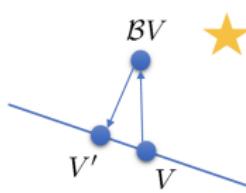


$$\|\mathcal{B}V - \mathcal{B}\bar{V}\|_\infty \leq \gamma \|V - \bar{V}\|_\infty$$

$\Pi$  is a contraction w.r.t.  $\ell_2$ -norm (Euclidean distance)

$$\|\Pi V - \Pi \bar{V}\|^2 \leq \|V - \bar{V}\|^2$$

but...  $\Pi \mathcal{B}$  is not a contraction of any kind



Conclusions:  
value iteration converges (tabular case)  
fitted value iteration does **not** converge  
not in general  
often not in practice

# What about fitted Q-iteration?

fitted Q iteration algorithm:

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

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

↑  
max now after the transition operator

define an operator  $\Pi$ :  $\Pi Q = \arg \min_{Q' \in \Omega} \frac{1}{2} \sum \|Q'(\mathbf{s}, \mathbf{a}) - Q(\mathbf{s}, \mathbf{a})\|^2$

fitted Q-iteration algorithm (using  $\mathcal{B}$  and  $\Pi$ ):

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

$\mathcal{B}$  is a contraction w.r.t.  $\infty$ -norm (“max” norm)

$\Pi$  is a contraction w.r.t.  $\ell_2$ -norm (Euclidean distance)

$\Pi \mathcal{B}$  is not a contraction of any kind

Applies also to online Q-learning

# Fitted Q-iteration – Regression, but not 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

# A sad corollary

batch actor-critic algorithm:

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

$\ell_\infty$  contraction  $\mathcal{B}$  (but without max)

$$y_{i,t} \approx r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1})$$

$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\| \hat{V}_\phi^\pi(\mathbf{s}_i) - y_i \right\|^2$$

$\ell_2$  contraction  $\Pi$

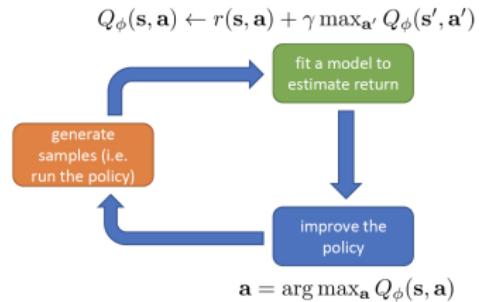
An aside regarding terminology

$V^\pi$ : value function for policy  $\pi$   
this is what the critic does

$V^*$ : value function for optimal policy  $\pi^*$   
this is what value iteration does

# Review

- Value iteration theory
  - Linear operator for backup
  - Linear operator for projection
  - Backup is contraction
  - Value iteration converges
- Convergence with function approximation
  - Projection is also a contraction
  - Projection + backup is not a contraction
  - Fitted value iteration does not in general converge
- Implications for Q-learning
  - Q-learning, fitted Q-iteration, etc. does not converge with function approximation
- But we can make it work in practice!



# Learning objectives of this lecture

- You should be able to...
  - Extend discrete value iteration to fitted value iteration with function approximation
  - Understand fitted Q-iteration (FQI)
  - Use deep neural networks to approximate Q-functions, be able to implement deep Q-learning with replay buffers and target networks
  - Know deep Q-learning with  $n$ -step returns
  - Know how to make deep Q-learning work with continuous actions
  - Be aware of some theories of value function methods

# Deep Q-learning suggested readings

- Lecture 7 & 8 of CS285 at UC Berkeley, **Deep Reinforcement Learning, Decision Making, and Control**
  - <http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-7.pdf>
  - <http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-8.pdf>
- Classic papers
  - Watkins. (1989). **Learning from delayed rewards**: introduces Q-learning
  - Lagoudakis. 2003. **Least-squares policy iteration**: linear function approximation
  - Riedmiller. (2005). **Neural fitted Q-iteration**: batch mode Q-learning with neural networks
- DRL Q-learning papers
  - Mnih et al. (2013). **Human level control through deep reinforcement learning**: Q-learning with convolutional networks for playing Atari.
  - Van Hasselt, Guez , Silver. (2015). **Deep reinforcement learning with double Q-learning**: a very effective trick to improve performance of deep Q-learning.
  - Lillicrap et al. (2016). **Continuous control with deep reinforcement learning**: continuous Q-learning with actor network for approximate maximization.
  - Wang, Schaul , Hessel, van Hasselt, Lanctot , de Freitas (2016). **Dueling network architectures for deep reinforcement learning**: separates value and advantage estimation in Q-function.
  - Z. Ren, et al., **Self-Paced Prioritized Curriculum Learning With Coverage Penalty in Deep Reinforcement Learning**, *TNNLS*, 2018.

# THE END