Lecture 9: Deep Q-learning

Zhi Wang & Chunlin Chen

Department of Control and Systems Engineering
Nanjing University

Nov. 26th, 2020

Contents and Goals

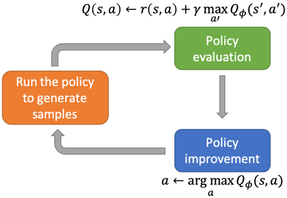
- How we can make Q-learning work with deep networks
  - Use replay buffers, separate target networks
- Tricks for improving Q-learning in practice
  - Double Q-learning, multi-step Q-learning
- Continuous Q-learning methods
- Goals
  - Understand how to implement Q-learning so that it can be used with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

Table of Contents

Review: Fitted Q-iteration (FQI)

- Full fitted Q-iteration algorithm. Loop:
  1. collect dataset $\{(s_i, a_i, r_i, s_i')\}$ using behavior policy π
     loop for K iterations:
     2. set $y_i \leftarrow r_i + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$
     3. set $\phi \leftarrow \arg\min_\phi \sum_i ||Q_\phi(s_i, a_i) - y_i||^2$

- Online fitted Q-iteration algorithm. Loop:
  1. observe one sample (s_i, a_i, r_i, s_i') using behavior policy π
  2. set $y_i \leftarrow r_i + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$
  3. set $\phi \leftarrow \phi - \alpha \frac{\mathrm{d}Q_\phi(s_i,a_i)}{\mathrm{d}\phi}(Q_\phi(s_i, a_i) - y_i)$



Problem 1: Correlated samples in MDPs

- Online fitted Q-iteration algorithm. Loop:
  1. take some action a_i observe (s_i, a_i, r_i, s_i')
  2. set $y_i \leftarrow r_i + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$
  3. set $\phi \leftarrow \phi - \alpha \frac{\mathrm{d}Q_\phi(s_i,a_i)}{\mathrm{d}\phi}(Q_\phi(s_i, a_i) - y_i)$

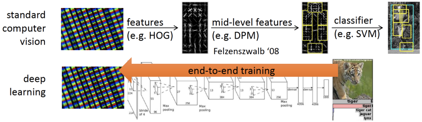
- these samples are correlated!

- Fitted Q-iteration is not gradient descent!

$$\phi \leftarrow \phi - \alpha \frac{\mathrm{d}Q_\phi(s_i,a_i)}{\mathrm{d}\phi}\left(Q_\phi(s_i,a_i) - \underbrace{\left(r_i + \gamma \max_{a_i'} Q_\phi(s_i', a_i')\right)}_{\text{no gradient through target value!}}\right)$$

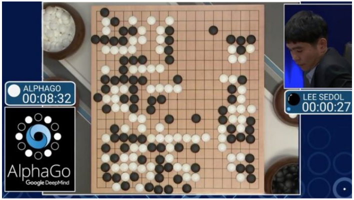
Supervised learning

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



Reinforcement learning

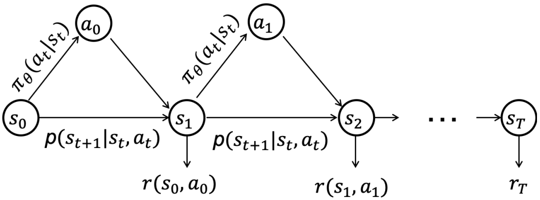
- Samples are not i.i.d., temporally correlated
- Given an initial state, find a sequence of optimal actions



Correlated samples in online Q-learning

- Online fitted Q-iteration algorithm. Loop:
  1. take some action a_i, observe (s_i, a_i, r_i, s'_i)
  2. set $\phi \leftarrow \phi - \alpha \frac{\mathrm{d}Q_\phi(s_i,a_i)}{\mathrm{d}\phi}\left(Q_\phi(s_i,a_i) - \left(r_i + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i)\right)\right)$

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



Correlate samples

- Full fitted Q-iteration algorithm. Loop:
  1. collect dataset $\{(s_i, a_i, r_i, s_i')\}$ using behavior policy π

     loop for K iterations:

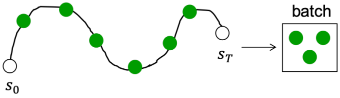
     2. set $y_i \leftarrow r_i + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$
     3. set $\phi \leftarrow \arg\min_\phi \sum_i ||Q_\phi(s_i, a_i) - y_i||^2$

- Online fitted Q-iteration. Loop:
  1. take some action a_i, observe (s_i, a_i, r_i, s_i')
  2. set $y_i = r_i + \gamma \max_{a_i'} Q_\phi(s_i', a_i')$
  3. set $\phi \leftarrow \phi - \alpha \frac{\mathrm{d}Q_\phi(s_i, a_i)}{\mathrm{d}\phi}(Q_\phi(s_i, a_i) - y_i)$

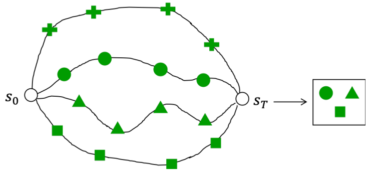
special case with $K = 1$, and one gradient step

How to reduce the correlation between samples?

- Samples in a single episode:
  - temporally correlated



- Samples from different episodes:
  - i.i.d



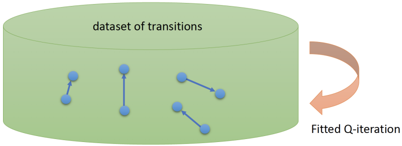
Replay buffers: store the data/transitions

- Full fitted Q-iteration algorithm. Loop:
  1. collect dataset $\{(s_i, a_i, r_i, s'_i)\}$ using behavior policy π

     loop for K iterations:

     2. set $y_i \leftarrow r_i + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i)$
     3. set $\phi \leftarrow \arg\min_\phi \sum_i ||Q_\phi(s_i, a_i) - y_i||^2$

- any behavior policy π will work!
- just load data from a buffer here
- still use $K = 1$ and one gradient step

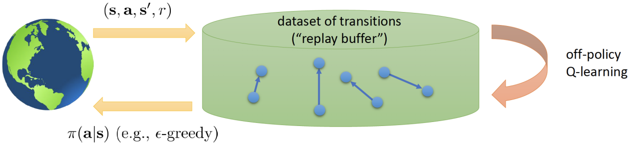


Q-learning with a replay buffer

- Loop:
    1. sample a batch $\{(s_j, a_j, r_j, s'_j)\}$ from buffer $\mathcal{B}$
    2. $\phi \leftarrow \phi - \alpha \sum_i \frac{\mathrm{d}Q_\phi(s_j,a_j)}{\mathrm{d}\phi} \left(Q_\phi(s_j, a_j) - \left(r_j + \gamma \max_{a'_j} Q_\phi(s'_j, a'_j) \right) \right)$

- Step 1: samples are no longer correlated if they come from different episodes
- Step 2: use multiple samples in the batch for low-variance gradient

- **Question**: Where does the data come from?
    - Need to periodically feed the replay buffer



Full Q-learning with a replay buffer

- Loop:
  1. collect dataset $\{(s_i, a_i, r_i, s'_i)\}$ using behavior policy, add it to $\mathcal{B}$

     loop for K iterations:

     2. sample a batch $\{(s_j, a_j, r_j, s'_j)\}$ from buffer $\mathcal{B}$
     3. $\phi \leftarrow \phi - \alpha \sum_j \frac{\mathrm{d}Q_\phi(s_j, a_j)}{\mathrm{d}\phi}\left(Q_\phi(s_j, a_j) - \left(r_j + \gamma \max_{a'_j} Q_\phi(s'_j, a'_j)\right)\right)$

- $K = 1$ is common, though larger K is more efficient

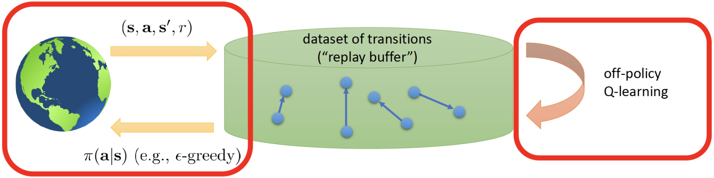


Table of Contents

Problem 2: Moving target in the Bellman equation

- Online fitted Q-iteration algorithm. Loop:
  1. take some action a_i observe (s_i, a_i, r_i, s'_i)
  2. set $y_i \leftarrow r_i + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i)$
  3. set $\phi \leftarrow \phi - \alpha \frac{\mathrm{d}Q_\phi(s_i,a_i)}{\mathrm{d}\phi}(Q_\phi(s_i, a_i) - y_i)$

- Samples are correlated: solved by a replay buffer

- Fitted Q-iteration is not gradient descent!
  - Target value changes when the Q-network ϕ is updated!

$$\phi \leftarrow \phi - \alpha \frac{\mathrm{d}Q_\phi(s_i, a_i)}{\mathrm{d}\phi} \left(Q_\phi(s_i, a_i) - \underbrace{\left(r_i + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i) \right)}_{\text{no gradient through target value!}} \right)$$

The moving target

- Full Q-learning with a replay buffer. Loop:
  1. collect dataset $\{(s_i, a_i, r_i, s'_i)\}$ using behavior policy, add it to $\mathcal{B}$
     loop for K iterations:
     2. sample a batch $\{(s_j, a_j, r_j, s'_j)\}$ from buffer $\mathcal{B}$
     3. $\phi \leftarrow \phi - \underbrace{\alpha \sum_j \frac{\mathrm{d}Q_\phi(s_j, a_j)}{\mathrm{d}\phi} \left(Q_\phi(s_j, a_j) - \left(r_j + \gamma \max_{a'_j} Q_\phi(s'_j, a'_j) \right) \right)}_{\text{one gradient step, moving target}}$

- Full fitted Q-iteration algorithm. Loop:
  1. collect dataset $\{(s_i, a_i, r_i, s'_i)\}$ using behavior policy π
     loop for K iterations:
     2. set $y_i \leftarrow r_i + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i)$
     3. set $\underbrace{\phi \leftarrow \arg\min_\phi \sum_i ||Q_\phi(s_i, a_i) - y_i||^2}_{\text{perfectly well-defined, stable regression}}$

Solution 2: Target networks

- Idea: use another Q-network and fix it in the inner loop
  - Targets don't change in the inner loop

Q-learning with replay buffer and target network. Loop:

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

   loop for N iterations:

   2. collect dataset $\{(s_i, a_i, r_i, s_i')\}$ using behavior policy, add it to $\mathcal{B}$

      loop for K iterations:

      3. sample a batch $\{(s_j, a_j, r_j, s_j')\}$ from buffer $\mathcal{B}$
      4. $\phi \leftarrow \phi - \alpha \sum_j \frac{\mathrm{d}Q_\phi(s_j, a_j)}{\mathrm{d}\phi} \left(Q_\phi(s_j, a_j) - \left(r_j + \gamma \max_{a_j'} Q_{\phi'}(s_j', a_j') \right) \right)$

"Classic" deep Q-network (DQN)

Q-learning with replay buffer and target network. Loop:

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

   loop for N iterations:

   2. collect dataset $\{(s_i, a_i, r_i, s_i')\}$ using behavior policy, add it to $\mathcal{B}$

      loop for K iterations:

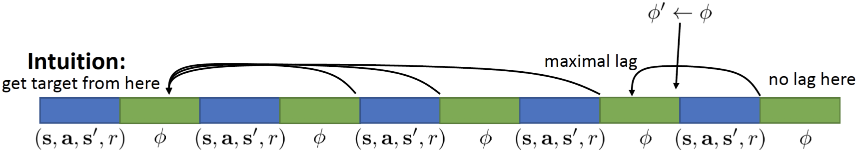
      3. sample a batch $\{(s_j, a_j, r_j, s_j')\}$ from buffer $\mathcal{B}$
      4. $\phi \leftarrow \phi - \alpha \sum_j \frac{\mathrm{d}Q_\phi(s_j, a_j)}{\mathrm{d}\phi} \left(Q_\phi(s_j, a_j) - \left(r_j + \gamma \max_{a_j'} Q_{\phi'}(s_j', a_j') \right) \right)$

- Classic deep Q-learning with $K = 1$. Loop:
  1. take some action a_i and observe (s_i, a_i, s_i', r_i), add it to $\mathcal{B}$
  2. sample mini-batch $\{(s_j, a_j, r_j, s_j')\}$ from $\mathcal{B}$ uniformly
  3. compute $y_j = r_j + \gamma \max_{a_j'} Q_{\phi'}(s_j', a_j')$ using target network $Q_{\phi'}$
  4. $\phi \leftarrow \phi - \alpha \sum_j \frac{\mathrm{d}Q_\phi(s_j, a_j)}{\mathrm{d}\phi} (Q_\phi(s_j, a_j) - y_j)$
  5. update ϕ': copy ϕ every N steps

Alternative target network

- Classic deep Q-learning with $K = 1$. Loop:
  1. take some action a_i and observe (s_i, a_i, s'_i, r_i), add it to $\mathcal{B}$
  2. sample mini-batch $\{(s_j, a_j, r_j, s'_j)\}$ from $\mathcal{B}$ uniformly
  3. compute $y_j = r_j + \gamma \max_{a'_j} Q_{\phi'}(s'_j, a'_j)$ using target network $Q_{\phi'}$
  4. $\phi \leftarrow \phi - \alpha \sum_j \frac{\mathrm{d}Q_\phi(s_j, a_j)}{\mathrm{d}\phi} (Q_\phi(s_j, a_j) - y_j)$
  5. update ϕ': copy ϕ every N steps

- **Problem**: In one inner loop, time lags for different steps are different!



Alternative target network

- Classic deep Q-learning with $K = 1$. Loop:
  1. take some action a_i and observe (s_i, a_i, s'_i, r_i), add it to $\mathcal{B}$
  2. sample mini-batch $\{(s_j, a_j, r_j, s'_j)\}$ from $\mathcal{B}$ uniformly
  3. compute $y_j = r_j + \gamma \max_{a'_j} Q_{\phi'}(s'_j, a'_j)$ using target network $Q_{\phi'}$
  4. $\phi \leftarrow \phi - \alpha \sum_j \frac{\mathrm{d}Q_\phi(s_j,a_j)}{\mathrm{d}\phi} (Q_\phi(s_j, a_j) - y_j)$
  5. update ϕ': copy ϕ every N steps

- Feels weirdly uneven, can we always have the same lag?
- Popular alternative updating for the target network:

$$5.\ \text{update } \phi' : \phi' \leftarrow \tau\phi' + (1-\tau)\phi$$

  - $\tau = 0.99$ works well

Deep Q-learning and fitted Q-iteration

Deep Q-learning ($N = 1, K = 1$). Loop:

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

   loop for N iterations:

   2. collect M transitions $\{(s_i, a_i, r_i, s_i')\}$ using behavior policy, add them to $\mathcal{B}$

      loop for K iterations:

      3. sample a batch $\{(s_j, a_j, r_j, s_j')\}$ from buffer $\mathcal{B}$
      4. $\phi \leftarrow \phi - \alpha \sum_j \frac{\mathrm{d}Q_\phi(s_j,a_j)}{\mathrm{d}\phi} \left(Q_\phi(s_j, a_j) - \left(r_j + \gamma \max_{a_j'} Q_{\phi'}(s_j', a_j') \right) \right)$

Fitted Q-iteration (written similarly as above). Loop:

1. collect M transitions $\{(s_i, a_i, r_i, s_i')\}$ using behavior policy, add them to $\mathcal{B}$

   loop for N iterations:

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

      loop for K iterations:

      3. sample a batch $\{(s_j, a_j, r_j, s_j')\}$ from buffer $\mathcal{B}$
      4. $\phi \leftarrow \phi - \alpha \sum_j \frac{\mathrm{d}Q_\phi(s_j,a_j)}{\mathrm{d}\phi} \left(Q_\phi(s_j, a_j) - \left(r_j + \gamma \max_{a_j'} Q_{\phi'}(s_j', a_j') \right) \right)$

A more general view

Deep Q-learning ($N = 1, K = 1$). Loop:

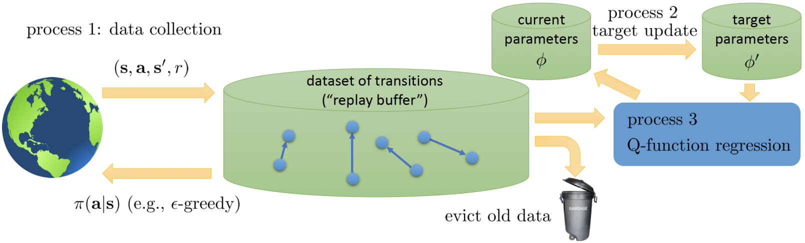
1. save target network parameters: $\phi' \leftarrow \phi$

   loop for N iterations:

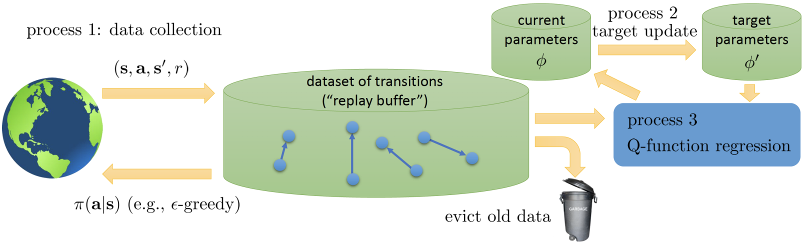
   2. collect M transitions $\{(s_i, a_i, r_i, s_i')\}$ using behavior policy, add them to $\mathcal{B}$

      loop for K iterations:

      3. sample a batch $\{(s_j, a_j, r_j, s_j')\}$ from buffer $\mathcal{B}$
      4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi(s_j, a_j)}{d\phi} \left(Q_\phi(s_j, a_j) - \left(r_j + \gamma \max_{a_j'} Q_{\phi'}(s_j', a_j') \right) \right)$



A more general view



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

Table of Contents

What's the problem with continuous actions?

- Full fitted Q-iteration algorithm. Loop:
  1. collect dataset $\{(s_i, a_i, r_i, s'_i)\}$ using behavior policy π
     loop for K iterations:
     2. set $y_i \leftarrow r_i + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i)$
     3. set $\phi \leftarrow \arg\min_\phi \sum_i ||Q_\phi(s_i, a_i) - y_i||^2$

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

The target value involves the max operator

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

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

- target value $y_j = r_j + \gamma \max_{a'_j} Q_{\phi'}(s'_j, a'_j)$
  - particularly problematic, need another inner loop of optimization
  - **Question**: how to perform the optimization, i.e., the max operator?

Option 1: Stochastic optimization

- The action space is typically low-dimensional
  - What about stochastic optimization?

The simplest solution: uniform sampling

- $\max_a Q(s,a) \approx \max\{Q(s,a_1),...,Q(s,a_n)\}$
- $(a_1,...,a_n)$ sampled from the some distribution (e.g., uniform)

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

More accurate solution: Cross-entropy method (CEM)

Simple **iterative** stochastic optimization:

1. Draw a sample from a probability distribution
2. Minimize the cross-entropy between this distribution and a target distribution to produce a better sample in the next iteration

works OK, for up to about 40 dimensions

A simple example of maximizing $f(\boldsymbol{x})$. Loop:

1. Obtain N samples: $\boldsymbol{X} \sim \text{SampleGaussian}(\mu, \sigma^2; N)$
2. Evaluate objective function $f(\boldsymbol{X})$ at sampled points
3. Sort $\boldsymbol{X}$ by $f(\boldsymbol{X})$ in descending order: $\boldsymbol{X} \leftarrow \text{sort}(\boldsymbol{X}, f)$
4. Update sampling distribution by the top M elites:
   $\mu \leftarrow \text{mean}(\boldsymbol{X}(1:M)), \quad \sigma^2 \leftarrow \text{var}(\boldsymbol{X}(1:M))$

Objective:

$$x^* = \arg\max_{\boldsymbol{x}} f(\boldsymbol{x})$$
$$\Downarrow$$
$$a^* = \arg\max_{\boldsymbol{a}} Q(s, \boldsymbol{a})$$

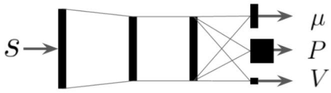
Many stochastic optimization solutions...

- Covariance matrix adaptation evolution strategy (CMA-ES)
  - an evolutionary algorithm for difficult non-linear non-convex black-box optimization problems in continuous domain
- Many more solutions...

Option 2: Easily maximizable Q-functions

- Use function class that is easy to optimize
  - e.g., the quadratic function

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



NAF Architecture.

- **NAF**: **N**ormalized **A**dvantage **F**unctions

$$\arg\max_a Q_\phi(s,a) = \mu_\phi(s)$$

$$\max_a Q_\phi(s,a) = V_\phi(s)$$

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

Table of Contents

Option 3: learn an approximate maximizer

- Lillicrap et al., "Continuous control with deep reinforcement learning," ICLR 2016.
  - Deep deterministic policy gradient (DDPG)
  - Really approximate **deep Q-learning** in the continuous action domain
- $\max_a Q_\phi(s,a) = Q_\phi(s, \arg\max_a Q_\phi(s,a))$
- **idea**: train another network $\mu_\theta(s)$ such that

$$\mu_\theta(s) \approx \arg\max_a Q_\phi(s,a)$$

- **Question**: how to optimize this deterministic "actor" $\mu_\theta(s)$?

Q-learning with continuous actions

- **idea**: train another network $\mu_\theta(s)$ such that

$$\mu_\theta(s) \approx \arg\max_a Q_\phi(s, a)$$

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

$$\frac{\mathrm{d}Q_\phi(s, \mu_\theta(s))}{\mathrm{d}\theta} = \frac{\mathrm{d}Q_\phi}{\mathrm{d}a} \cdot \frac{\mathrm{d}a}{\mathrm{d}\theta} = \frac{\mathrm{d}Q_\phi}{\mathrm{d}\mu_\theta(s)} \cdot \frac{\mathrm{d}\mu_\theta(s)}{\mathrm{d}\theta}$$

- new target

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

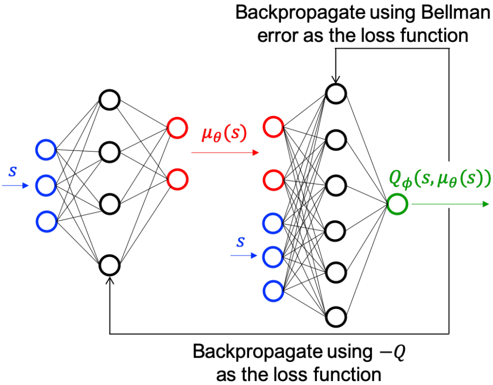
DDPG network architecture

- Backpropagate the critic:

$$\nabla_\phi = \frac{\mathrm{d}Q_\phi(s,a)}{\mathrm{d}\phi}\left(Q_\phi(s,a) - y\right)$$

- Backpropagate the actor:

$$\nabla_\theta = \frac{\mathrm{d}Q_\phi}{\mathrm{d}\mu_\theta(s)} \cdot \frac{\mathrm{d}\mu_\theta(s)}{\mathrm{d}\theta}$$



Deep deterministic policy gradient (DDPG)

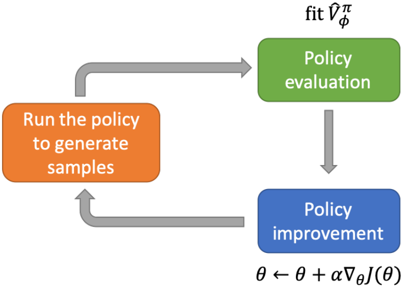
- Loop:
  1. take some action a_i and observe (s_i, a_i, s_i', r_i), add it to $\mathcal{B}$
  2. sample mini-batch $\{(s_j, a_j, r_j, s_j')\}$ from $\mathcal{B}$ uniformly
  3. compute $y_j = r_j + \gamma Q_{\phi'}(s_j', \mu_{\theta'}(s_j'))$ by *target* networks $Q_{\phi'}$ and $\mu_{\theta'}$
  4. $\phi \leftarrow \phi - \alpha \sum_j \frac{\mathrm{d}Q_\phi(s_j,a_j)}{\mathrm{d}\phi}\left(Q_\phi(s_j, a_j) - y_j\right)$
  5. $\theta \leftarrow \theta + \beta \sum_j \frac{\mathrm{d}Q_\phi}{\mathrm{d}\mu_\theta(s_j)} \frac{\mathrm{d}\mu_\theta(s_j)}{\mathrm{d}\theta}$
  6. update ϕ', θ': $\phi' \leftarrow \tau\phi' + (1-\tau)\phi$, $\theta' \leftarrow \tau\theta' + (1-\tau)\theta$

- The behavior policy π:
  - The target greedy policy is $\pi^*(s) = \mu_\theta(s)$, actually
  - Add some exploration noise to the target greedy policy, just like ϵ-greedy in tabular Q-learning

$$\pi(a|s) \sim \mathcal{N}(\mu_\theta(s), \sigma^2)$$

Review: Actor-critic algorithms

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



DDPG vs. Actor-critic

- DDPG. Loop:
  1. take some action a_i and observe (s_i, a_i, s_i', r_i), add it to $\mathcal{B}$
  2. sample mini-batch $\{(s_j, a_j, r_j, s_j')\}$ from $\mathcal{B}$ uniformly
  3. compute $y_j = r_j + \gamma Q_{\phi'}(s_j', \mu_{\theta'}(s_j'))$ by *target* networks $Q_{\phi'}$ and $\mu_{\theta'}$
  4. $\phi \leftarrow \phi - \alpha \sum_j \frac{\mathrm{d}Q_\phi(s_j,a_j)}{\mathrm{d}\phi}(Q_\phi(s_j, a_j) - y_j)$
  5. $\theta \leftarrow \theta + \beta \sum_j \frac{\mathrm{d}Q_\phi}{\mathrm{d}\mu_\theta(s_j)} \frac{\mathrm{d}\mu_\theta(s_j)}{\mathrm{d}\theta}$
  6. update ϕ', θ': $\phi' \leftarrow \tau\phi' + (1-\tau)\phi$, $\theta' \leftarrow \tau\theta' + (1-\tau)\theta$

- Actor-critic. Loop:
  1. sample $\{s_i, a_i, r_i, s_i'\}$ from $\pi_\theta(a|s)$ (run it on the robot)
  2. policy evaluation: fit $\hat{V}_\phi^\pi(s)$ to sampled reward sums
  3. evaluate $\hat{A}^\pi(s_i, a_i) = r_i + \gamma \hat{V}_\phi^\pi(s_i') - \hat{V}_\phi^\pi(s_i)$
  4. policy improvement: $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(a_i|s_i)\hat{A}^\pi(s_i, a_i)$
  5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Review: Q-learning vs. SARSA

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

- Q-learning approximates the optimal action-value function for an optimal policy, $Q \approx Q_* = Q_{\pi_*}$
  - The target policy is greedy w.r.t Q, $\pi(a|s) = \arg\max_a Q(s, a)$
  - The behavior policy can be others, e.g., $b(a|s) = \varepsilon$-greedy

---

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

- SARSA approximates the action-value function for the behavior policy, $Q \approx Q_\pi = Q_b$
  - The target and the behavior policy are the same, e.g., $\pi(a|s) = b(a|s) = \varepsilon$-greedy

DDPG vs. Actor-critic

- DDPG
  - The actor: approximate the optimal policy $a^* \mu_\theta(s) = \arg\max_a Q_\phi(s, a)$
  - The critic: approximate the optimal action-value function Q^*_ϕ
  - Off-policy, more sample efficient
- Actor-critic
  - The actor: approximate the current policy $a \sim \pi_\theta(a|s)$
  - The critic: approximate the state-value function V^π_ϕ for given policy π
  - On-policy, at least converge to a local optimum

Table of Contents

Overestimation in Q-learning

- target value $y_j = r_j + \gamma \underbrace{\max_{a'_j} Q_{\phi'}(s'_j, a'_j)}_{\text{this is the problem}}$

- Imagine we have two random variables: x_1 and x_2

$$\mathbb{E}[\max(x_1, x_2)] \geq \max(\mathbb{E}[x_1], \mathbb{E}[x_2])$$

  - $Q_{\phi'}(s', a')$ is not perfect – it looks "noisy"
  - hence $\max_{a'} Q_{\phi'}(s', a')$ **overestimates** the next value!

- note that $\max_{a'} Q_{\phi'}(s', a') = Q_{\phi'}(s', \arg\max_{a'} Q_{\phi'}(s', a'))$
  - action selected according to $Q_{\phi'}$
  - value also comes from $Q_{\phi'}$

Double Q-learning

- $\mathbb{E}[\max(x_1, x_2)] \geq \max(\mathbb{E}[x_1], \mathbb{E}[x_2])$
- note that $\max_{a'} Q_{\phi'}(s', a') = Q_{\phi'}(s', \arg\max_{a'} Q_{\phi'}(s', a'))$
  - action selected according to $Q_{\phi'}$
  - value also comes from $Q_{\phi'}$
  - if the noise in the two parts 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

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

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

  - if the two Q-networks, Q_{ϕ_A} and Q_{ϕ_B}, 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: $y = r + \gamma Q_{\phi'}(s', \arg\max_{a'} Q_{\phi'}(s', a'))$
- double Q-learning: $y = r + \gamma Q_{\phi'}(s', \arg\max_{a'} Q_{\phi}(s', a'))$
  - just use current network (not target network) to evaluate action
  - still use target network to evaluate value

Table of Contents

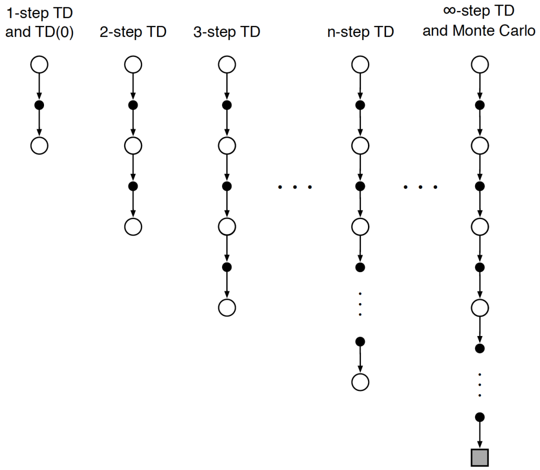
n-step bootstrapping: Combine MC and one-step TD

- 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	$n = \infty$
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**

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

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

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

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

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

n-step TD update rule

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

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

- 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 $V(S_{t+n})$

$$\begin{aligned} V(S_t) &\leftarrow V(S_t) + \alpha[G_{t:t+n} - V(S_t)] \\ &= V(S_t) + \alpha[R_{t+1} + \gamma R_{t+2} + ... + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) - V(S_t)] \end{aligned}$$

Deep Q-learning with n-step bootstrapping

- Q-learning target: $y_{j,t} = r_{j,t} + \gamma \max_{a'_{j,t+1}} Q_{\phi'}(s'_{j,t+1}, a'_{j,t+1})$
  - these are the only values that matter if $Q_{\phi'}$ is bad!
  - these values are important if $Q_{\phi'}$ is good

- Construct multi-step targets, N-step return estimator:

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

Table of Contents

Simple practical tips for Q-learning

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

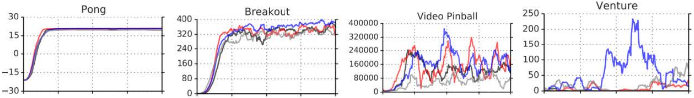


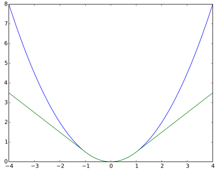
Figure: From T. Schaul, J. Quan, I. Antonoglou, and D. Silver. "Prioritized experience replay". *arXiv preprint arXiv:1511.05952* (2015), Figure 7

- Large replay buffers help improve stability
  - Looks more like fitted Q-iteration
- It tasks time, be patient - might be no better than random for a while
- Start with high exploration and gradually move to high exploitation

Advanced tips for Q-learning

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

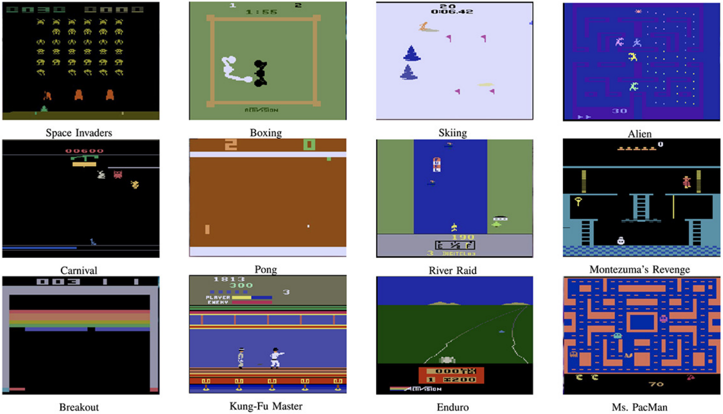
$$\mathcal{L}(x) = \begin{cases} x^2/2 & \text{if } |x| \le \delta \\ \delta|x| - \delta^2/2 & \text{otherwise} \end{cases}$$



- 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

Q-learning with convolutional networks

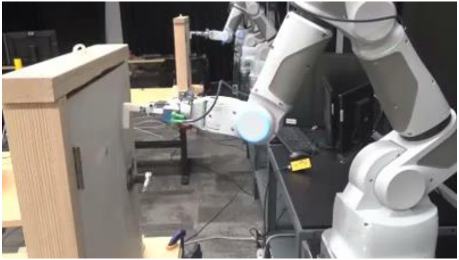
- Mnih et al., "Human-level control through deep reinforcement learning," 2013.
- Use replay buffer and target network
- One-step backup, one gradient step
- Can be improved a lot with double Q-learning (and other tricks)



Space Invaders, Boxing, Skiing, Alien, Carnival, Pong, River Raid, Montezuma's Revenge, Breakout, Kung-Fu Master, Enduro, Ms. PacMan

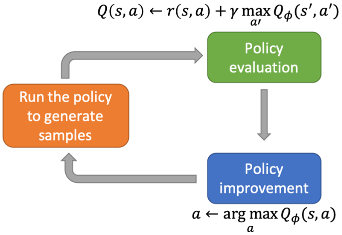
Q-learning on a real robot

- Gu et al., "Robot manupulation with deep reinforcement learning and ...," 2017.
- Continuous actions with NAF (quadratic in actions)
- Use replay buffer and target network
- One-step backup, four gradient steps per simulator step for efficiency
- Parallelized across multiple robots



Review

- Q-learning with deep neural networks
  - Replay buffers
  - Target networks
- Generalized fitted Q-iteration
- Deep deterministic policy network
  - Deep Q-learning for continuous action space
  - Another network for approximating optimal policy
  - Off-policy
- Extensions
  - Double Q-learning
  - Multi-step Q-learning



Learning objectives of this lecture

- You should be able to...
  - Use deep neural networks to approximate Q-functions, be able to implement deep Q-learning with replay buffers and target networks
  - Use deep deterministic policy gradient for continuous actions

  - Know double Q-learning for addressing the overestimation problem
  - Know deep Q-learning with n-step returns

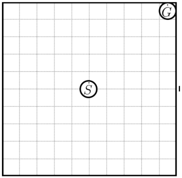
Deep Q-learning suggested readings

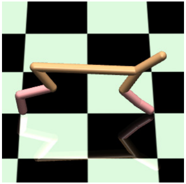
- Lecture 8 of CS285 at UC Berkeley, **Deep Reinforcement Learning, Decision Making, and Control**
  - `http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-8.pdf`

- 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.

Homework 5

- Study the DDPG algorithm in detail
- Implement the DDPG algorithm on problems 1 & 2
  - Problem 1: the point maze navigation, continuous state-action space ($s, a \in \mathbb{R}^2$, $s \in [-0.5, 0.5]^2, a \in [-0.1, 0.1]^2$)
  - Problem 2: the MuJoCo HalfCheetah, make the robot run forward
  - Compare DDPG with policy gradient and actor-critic algorithms
- Write a report introducing the algorithms and your experimentation
  - Explanations, steps, evaluation results, visualizations...
  - Submit the code and the report to *yuanyang@smail.nju.edu.cn*





THE END