Advanced algorithms for learning Q-functions

Presentation of Sergey Levine
adapted by Damien Ernst
Plan

1. How can we make Q-learning work better in practice?
2. A generalized view of Q-learning algorithms
3. Tricks for improving Q-learning in practice
4. 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
Recap: Learning Q-functions

**full fitted Q-iteration algorithm:**

1. collect dataset \( \{(s_i, a_i, s'_i, r_i)\} \) using some policy

2. set \( y_i \leftarrow r(s_i, a_i) + \gamma \max_{a'_i} Q_{\phi}(s'_i, a'_i) \)

3. set \( \phi \leftarrow \arg\min_{\phi} \frac{1}{2} \sum_i \| Q_{\phi}(s_i, a_i) - y_i \|^2 \)

**online Q iteration algorithm:**

1. take some action \( a_i \) and observe \( (s_i, a_i, s'_i, r_i) \)

2. \( y_i = r(s_i, a_i) + \gamma \max_{a'_i} Q_{\phi}(s'_i, a'_i) \)

3. \( \phi \leftarrow \phi - \alpha \frac{dQ_{\phi}}{d\phi}(s_i, a_i)(Q_{\phi}(s_i, a_i) - y_i) \)

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

\[ a = \arg\max_a Q_{\phi}(s, a) \]
What’s wrong?

online Q iteration algorithm:

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

these are correlated!

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

Q-learning is *not* gradient descent!

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

no gradient through target value
Correlated samples in online Q-learning

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

- sequential states are strongly correlated
- target value is always changing
Another solution: replay buffers

online Q iteration algorithm:
1. take some action \( a_i \) and observe \((s_i, a_i, s'_i, r_i)\)
2. \( \phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i)]) \)

full fitted Q-iteration algorithm:
1. collect dataset \( \{(s_i, a_i, s'_i, r_i)\} \) using some policy
2. set \( y_i \leftarrow r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i) \)
3. set \( \phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2 \)

special case with \( K = 1, \) and one gradient step

any policy will work! (with broad support)
just load data from a buffer here
still use one gradient step

Fitted Q-iteration
Q-learning with replay buffer

Q-learning with a replay buffer:

1. sample a batch \((s_i, a_i, s'_i, r_i)\) from \(\mathcal{B}\)

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

+ 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...

\[
(s, a, s', r) \quad \Rightarrow \quad \text{dataset of transitions ("replay buffer")}
\]

\[
\pi(a|s) \ (\text{e.g., } \epsilon\text{-greedy}) \quad \Rightarrow \quad \text{off-policy Q-learning}
\]
Putting it together

full Q-learning with replay buffer:

1. collect dataset \( \{(s_i, a_i, s'_i, r_i)\} \) using some policy, add it to \( B \)

2. sample a batch \( (s_i, a_i, s'_i, r_i) \) from \( B \)

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

\( K = 1 \) is common, though larger \( K \) more efficient
What is still wrong?

online Q iteration algorithm:

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

Q-learning is *not* gradient descent!

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

**These are correlated!**

use replay buffer

This is still a problem!
Why has fitted-Q iteration still an edge over Q-learning with replay buffer?

full Q-learning with replay buffer:
1. collect dataset \( \{(s_i, a_i, s'_i, r_i)\} \) using some policy, add it to \( B \)
2. sample a batch \( (s_i, a_i, s'_i, r_i) \) from \( B \)
3. \( \phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i)]) \)

one gradient step, moving target

full fitted Q-iteration algorithm:
1. collect dataset \( \{(s_i, a_i, s'_i, r_i)\} \) using some policy
2. set \( y_i \leftarrow r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i) \)
3. set \( \phi \leftarrow \underset{\phi}{\operatorname{arg\,min}} \frac{1}{2} \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2 \)

perfectly well-defined, stable regression
Q-learning algorithm with replay buffer and target network

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

Q-learning with replay buffer and target network:

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

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

3. sample a batch $(s_i, a_i, s'_i, r_i)$ from $B$

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

process 1: data collection

process 2: target update

process 3: Q-function regression

$\pi(a|s)$ (e.g., $\epsilon$-greedy)

dataset of transitions ("replay buffer")

evict old data

current parameters $\phi$

target parameters $\phi'$

process 2

Q-function regression
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 and process 3 run at the same speed, process 2 is slow
- Fitted Q-iteration: process 3 and process 2 are combined in a single process. But variants of FQI with target networks could exist.
Overestimation in Q-learning

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

this last term is the problem

imagine we have two random variables: \( X_1 \) and \( X_2 \)

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

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

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

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

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

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:

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

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

if the two Q’s are noisy in different ways, there is no problem
Double Q-learning in practice

where to get two Q-functions?

just use the current and target networks!

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

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!
Q-learning with continuous actions

What’s the problem with continuous actions?

\[ \pi(a_t|s_t) = \begin{cases} 1 & \text{if } a_t = \text{arg max}_{a_t} Q_{\phi}(s_t, a_t) \\ 0 & \text{otherwise} \end{cases} \]

target value \( y_j = r_j + \gamma \text{max}_{a_j'} Q_{\phi'}(s_j', a_j') \)

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:

\[
\max_a Q(s, a) \approx \max \{Q(s, a_1), \ldots, Q(s, a_N)\}
\]

\((a_1, \ldots, a_N)\) 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
  works OK, for up to about 40 dimensions

• Covariance Matrix Adaptation Evolution Strategy (CMA-ES)
  • substantially less simple iterative stochastic optimization
Easily maximizable Q-functions

Option 2: use function class that is easy to optimize

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

**NAF: Normalized Advantage Functions**

\[ \arg \max_a Q_\phi(s, a) = \mu_\phi(s) \]
\[ \max_a Q_\phi(s, a) = V_\phi(s) \]

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

Gu, Lillicrap, Sutskever, L., ICML 2016
Q-learning with continuous actions

Option 3: learn an approximate maximizer

DDPG (Lillicrap et al., ICLR 2016)  “deterministic” actor-critic
really approximate Q-learning

\[
\max_a Q_{\phi}(s, a) = Q_{\phi}(s, \arg \max_a Q_{\phi}(s, a))
\]

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

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

\[
\frac{dQ_{\phi}}{d\theta} = \frac{da}{d\theta} \frac{dQ_{\phi}}{da}
\]

new target \(y_j = r_j + \gamma \max_{a'_j} Q'_{\phi'}(s'_j, \mu_\theta(s'_j))\)
Q-learning with continuous actions

Option 3: learn an approximate maximizer

DDPG:

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

![Graphs of Pong, Breakout, Video Pinball, and Venture](image)

**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

• Double Q-learning helps *a lot* in practice, simple and no downsides

• Start with high exploration (epsilon) and gradually reduce

Slide partly borrowed from J. Schulman
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. ‘15
- 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
Q-learning suggested readings

• Classic papers

• Deep reinforcement learning Q-learning papers
  • Mnih et al. (2013). Human-level control through deep reinforcement learning: Q-learning with convolutional networks for playing Atari.
  • Lillicrap et al. (2016). Continuous control with deep reinforcement learning: continuous Q-learning with actor network for approximate maximization.