Deep RL with Q-Functions

CS 285

Instructor: Sergey Levine
UC Berkeley
Recap: Q-learning

full fitted Q-iteration algorithm:
1. collect dataset \( \{(s_i, a_i, s'_i, r_i)\} \) using some policy
2. set \( y_i \leftarrow r(s_i, a_i) + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i) \) \( K \times \)
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. \( \phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - y_i) \)

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

fit a model to estimate return
generate samples (i.e. run the policy)
improve the policy
\[ a = \arg\max_a Q_\phi(s, a) \]
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. \(\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a')]])\)

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

synchronized parallel Q-learning
asynchronous parallel Q-learning

get \((s, a, s', r)\) update \(\phi\)
get \((s, a, s', r)\) update \(\phi\)

\(t\) \(\theta\)
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. $\phi \leftarrow \phi - \alpha \frac{dQ_{\phi}}{d\phi}(s_i, a_i)(Q_{\phi}(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'_i} Q_{\phi}(s'_i, a'_i)])$

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

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

dataset of transitions

Fitted $Q$-iteration
Another solution: replay buffers

Q-learning with a replay buffer:

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

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...

\((s, a, s', r)\)

dataset of transitions ("replay buffer")

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

K = 1 is common, though larger K more efficient

\( (s, a, s', r) \)

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

dataset of transitions ("replay buffer")

off-policy Q-learning
Target Networks
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. $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - y_i) \quad \text{these are correlated! use replay buffer}$

Q-learning is *not* gradient descent!

$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s'_i, a'_i)]) \quad \text{no gradient through target value}$$

This is still a problem!
Q-Learning and Regression

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

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

targets don’t change in inner loop!
“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 \( \{(s_i, a_i, s'_i, r_i)\} \) using some policy, add it to \( \mathcal{B} \)
3. sample a batch \( (s_i, a_i, s'_i, r_i) \) from \( \mathcal{B} \)
4. \( \phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, a_i)(Q_\phi(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'} Q_{\phi'}(s'_i, a'_i)]) \)

“classic” deep Q-learning algorithm:
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 \( \{s_j, a_j, s'_j, r_j\} \) 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. \( \phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(s_j, a_j)(Q_\phi(s_j, a_j) - y_j) \)
5. update \( \phi' \): copy \( \phi \) every \( N \) steps

Mnih et al. ‘13

You’ll implement this in HW3!
Alternative target network

“classic” deep Q-learning algorithm:

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 $y_j = r_j + \gamma \max_{a'_j} Q_{\phi'}(s'_j, a'_j)$ using target network $Q_{\phi'}$
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. 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
A General View of Q-Learning
Fitted Q-iteration and Q-learning

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

DQN: \( N = 1, \ K = 1 \)

Fitted Q-learning (written similarly as above):
1. collect \( M \) datapoints \( \{(s_i, a_i, s'_i, r_i)\} \) using some policy, add them to \( B \)
2. save target network parameters: \( \phi' \leftarrow \phi \)
3. sample a batch \( (s_i, a_i, s'_i, r_i) \) from \( B \)
4. \( \phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi} (s_i, a_i)(Q_\phi(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'} Q_\phi'(s'_i, a'_i)]) \)

\} \quad \text{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 $\{(s_i, a_i, s'_i, r_i)\}$ 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

Q-function regression
A more general view

- Online Q-learning (last lecture): 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 in the inner loop of process 2, which is in the inner loop of process 1
Improving Q-Learning
Are the Q-values accurate?

As predicted Q increases, so does the return
Are the Q-values accurate?
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:

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

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

if the two Q’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: $y = r + \gamma Q_{\phi'}(s', \text{arg max}_{a'} Q_{\phi'}(s', a'))$

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

just use current network (not target network) to evaluate action
still use target network to evaluate value!
Multi-step returns

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

where does the signal come from?

Q-learning does this: max bias, min variance

remember this?

\[
\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \left( r(s_{i,t}, a_{i,t}) + \gamma \hat{V}_\theta^*(s_{i,t+1}) - \hat{V}_\theta^*(s_{i,t}) \right)
\]

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

Policy gradient:

\[
\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \left( \sum_{t'=t}^{T} \gamma^{t'-t} r(s_{i,t'}, a_{i,t'}) - b \right)
\]

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

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

\[
y_{j,t} = \sum_{t'=t}^{t+N-1} \gamma^{t-t'} r_{j,t'} + \gamma^N \max_{a_{j,t+N}} Q_{\phi'}(s_{j,t+N}, a_{j,t+N})
\]

N-step return estimator
Q-learning with N-step returns

\[
y_{j,t} = \sum_{t'=t}^{t+N-1} \gamma^{t-t'} r_{j,t'} + \gamma^N \max_{a_{j,t+N}} Q^{\phi}(s_{j,t+N}, a_{j,t+N})
\]

this is supposed to estimate \(Q^\pi(s_{j,t}, a_{j,t})\) for \(\pi\)

\[
\pi(a_t|s_t) = \begin{cases} 
1 & \text{if } a_t = \arg\max_a Q^{\phi}(s_t, a) \\
0 & \text{otherwise} \end{cases}
\]

why?

we need transitions \(s_{j,t'}, a_{j,t'}, s_{j,t'+1}\) 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

+ less biased target values when Q-values are inaccurate
+ typically faster learning, especially early on
- only actually correct when learning on-policy

For more details, see: “Safe and efficient off-policy reinforcement learning.” Munos et al. ‘16
Q-Learning with Continuous Actions
Q-learning with continuous actions

What’s the problem with continuous actions?

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

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

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

• 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

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

**NAF: Normalized Advantage Functions**

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

+ 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
(real 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 $y_j = r_j + \gamma Q_\phi'(s'_j, \mu_\theta(s'_j)) \approx r_j + \gamma Q_\phi'(s'_j, \arg\max_{a'} Q_\phi(s'_j, a'_j))$
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 $y_j = r_j + \gamma Q_{\phi'}(s'_j, \mu_{\theta'}(s'_j))$ 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. $\theta \leftarrow \theta + \beta \sum_j \frac{d\mu}{d\theta}(s_j) \frac{dQ_{\phi}}{da}(s_j, \mu(s_j))$
6. update $\phi'$ and $\theta'$ (e.g., Polyak averaging)
Implementation Tips and Examples
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

• 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

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

Slide partly borrowed from J. Schulman
Advanced tips for Q-learning

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

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

• Double Q-learning helps \textit{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

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

stored data from all past experiments $\{(s_i, a_i, s'_i)\}_i$

live data collection

training buffers
- off-policy $(s, a, s', r)$
- on-policy $(s, a, s', r)$
- labeled $(s, a, Q_T(s, a))$

Bellman updaters
compute $Q_T(s, a) = r + \max_{a'} Q_\theta(s', a')$

training threads
$\min_\theta ||Q_\theta(s, a) - Q_T(s, a)||^2$

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

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

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

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