Practical Deep RL Implementation Techniques

CS 224R
Reminders

Today:  
Homework 1 due, Homework 2 out

Wed next week:  
Project proposal due
The Plan

Recap & finish Q learning

Q learning tricks

Improving Q learning

Case studies: games, robotics

Key learning goals:

• Practical Q learning implementation tricks
• Understanding the landscape of Q learning algorithms
The Plan

Recap & finish Q learning

Q learning tricks

Improving Q learning

Case studies: games, robotics
Value-Based RL

Value function: \( V^\pi(s_t) = ? \)

Q function: \( Q^\pi(s_t, a_t) = ? \)

Advantage function: \( A^\pi(s_t, a_t) = ? \)

Reward = 1 if I can play it in a month, 0 otherwise

Current \( \pi(a_2|s) = 1 \)

How can we improve the policy?
Policy Iteration

policy iteration algorithm:
1. evaluate $A^\pi(s, a)$
2. set $\pi \leftarrow \pi'$

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

as before: $A^\pi(s, a) = r(s, a) + \gamma E[V^\pi(s')] - V^\pi(s)$

Slide adapted from Sergey Levine
Value Iteration

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

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

\[
\arg\max_{a_t} A^\pi(s_t, a_t) = \arg\max_a Q^\pi(s_t, a_t)
\]

\[
Q^\pi(s, a) = r(s, a) + \gamma E[V^\pi(s')] \quad \text{(a bit simpler)}
\]

Skip the policy and compute values directly!

Value iteration algorithm:
1. set \( Q(s, a) \leftarrow r(s, a) + \gamma E[V(s')] \)
2. set \( V(s) \leftarrow \max_a Q(s, a) \)

Policy iteration algorithm:
1. evaluate \( Q^\pi(s, a) \)
2. set \( \pi \leftarrow \pi' \)

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

\[
\arg\max_a Q(s, a) \rightarrow \text{policy}
\]

approximates the new value!

Fit a model to estimate return

Generate samples (i.e. run the policy)

Improve the policy

\[
V^\pi(s) \leftarrow \max_a Q^\pi(s, a)
\]

Slide adapted from Sergey Levine
Q learning

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

Value iteration algorithm:
1. set \( Q(s, a) \leftarrow r(s, a) + \gamma E[V(s')] \)
2. set \( V(s) \leftarrow \max_a Q(s, a) \)

Fitted Q iteration algorithm:
1. set \( y_i \leftarrow r(s_i, a_i) + \gamma E[V_\phi(s'_i)] \) approximate \( E[V(s'_i)] \approx \max_{a'} Q_\phi(s'_i, a'_i) \)
2. set \( \phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2 \) doesn’t require simulation of actions!

\[ Q^*(s_t, a_t) = E_{s_{t+1} \sim p(s_{t+1}|s_t, a_t)} \left[ r(s_t, a_t) + \gamma \max_{a'} Q^*(s_{t+1}, a') \right] \]
Value-Based RL: Definitions

\[ Q^\pi(s_t, a_t) = \sum_{t'=t}^T E_{\pi_\theta} [r(s_{t'}, a_{t'})|s_t, a_t] \]: total reward from taking \( a_t \) in \( s_t \) \hfill \text{"how good is a state-action pair"}

\[ V^\pi(s_t) = E_{a_t \sim \pi_\theta(a_t|s_t)} [Q^\pi(s_t, a_t)] \]: total reward from \( s_t \) \hfill \text{"how good is a state"}

If you know \( Q^\pi \), you can use it to improve \( \pi \).

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

For the optimal policy \( \pi^* \):

\[ Q^*(s_t, a_t) = E_{s_{t+1} \sim p(s_{t+1}|s_t, a_t)} \left[ r(s_t, a_t) + \gamma \max_{a'} Q^*(s_{t+1}, a') \right] \]

Bellman equation
Value-Based RL

Value function: $V^\pi(s_t) = ?$
Q function: $Q^\pi(s_t, a_t) = ?$
Q* function: $Q^*(s_t, a_t) = ?$
Value* function: $V^*(s_t) = ?$

Reward = 1 if I can play it in a month, 0 otherwise

Current $\pi(a_2|s) = 1$
Fitted Q-iteration Algorithm

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

Algorithm hyperparameters

- dataset size \( N \)
- collection policy
- iterations \( K \)
- gradient steps \( S \)

Result: get a policy \( \pi(a|s) \) from \( \arg\max_a Q_\phi(s, a) \)

Important notes:

- We can reuse data from previous policies!
- An off-policy algorithm using replay buffers
Q learning animation
Q-learning

Bellman equation: \[ Q^*(s_t, a_t) = E_{s_{t+1} \sim p(s_{t+1}|s_t, a_t)} \left[ r(s_t, a_t) + \gamma \max_{a'} Q^*(s_{t+1}, a') \right] \]

Pros:
+ More sample efficient than on-policy methods
+ Can incorporate off-policy data (including a fully offline setting)
+ Can updates the policy even without seeing the reward
+ Relatively easy to parallelize

Cons:
- Lots of “tricks” to make it work
- Potentially could be harder to learn than just a policy
The Plan

Recap

Q learning tricks

Improving Q learning

Case studies: games, robotics
Q-learning

fitted Q iteration algorithm:

1. set $y_i \leftarrow r(s_i, a_i) + \gamma \max_{a'_i} Q_\phi(s'_i, a'_i)$
2. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \| Q_\phi(s_i, a_i) - y_i \|^2$

Questions:

• Is this a gradient descent algorithm?
• Is this algorithm off or on policy?
• What could be potential problems with it?
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. set $\phi \leftarrow \arg\min_{\phi} \frac{1}{2} \sum_i \| Q_{\phi}(s_i, a_i) - y_i \|^2$

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

online Q iteration algorithm:

1. take some action $a_i$ and observe $(s_i, a_i, s'_i, r_i)$
2. set $\phi \leftarrow \arg \min_\phi \frac{1}{2} \sum_i \| Q_\phi(s_i, a_i) - y_i \|^2$

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

any policy will work!
just load data from a buffer here
Solution: replay buffers

Q-learning with a replay buffer:

1. sample a batch \((s_i, a_i, s'_i, r_i)\) from \(B\)
2. \(\phi \leftarrow \arg \min_{\phi} \frac{1}{2} \sum_i \| Q_\phi(s_i, a_i) - y_i \|^2 \)

but where does the data come from?

need to periodically feed the replay buffer...

+ samples are no longer correlated
+ multiple samples in the batch (low-variance gradient)
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. set \( \phi \leftarrow \arg \min \phi \frac{1}{2} \sum_i \|Q_\phi(s_i, a_i) - y_i\|^2 \)

\[ K = 1 \text{ is common, though larger } K \text{ more efficient} \]
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. set $\phi \leftarrow \arg \min_{\phi} \frac{1}{2} \sum_i \|Q_{\phi}(s_i, a_i) - y_i\|^2$

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

these are correlated!

use replay buffer
Q-Learning and Regression

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

Moving targets!

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 $B$
3. sample a batch $(s_i, a_i, s'_i, r_i)$ from $B$
4. set $\phi \leftarrow \text{arg min}_\phi \frac{1}{2} \sum_i \|Q_\phi(s_i, a_i) - [r(s_i, a_i) + \gamma \max_{a'} Q_{\phi'}(s'_i, a'_i)]\|^2$

targets don’t change in inner loop!
“Classic” deep Q-learning algorithm (DQN)

“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. set $\phi \leftarrow \arg \min_{\phi} \frac{1}{2} \sum_{j} ||Q_{\phi}(s_j, a_j) - y_j||^2$
5. update $\phi'$: copy $\phi$ every $N$ steps

Mnih et al. ‘13
The Plan

Recap

Q learning tricks

Improving Q learning

Case studies: games, robotics
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!

where does the signal come from?

Q-learning does this: max bias, min variance

remember this?

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

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

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

+ 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

\[ \begin{align*}
& \text{• Does it still work off-policy?} \\
\end{align*} \]
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 $$\pi$$

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

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
Aside: exploration with Q-learning

online Q iteration algorithm

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

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

3. set $\phi \leftarrow \arg \min \phi \frac{1}{2} \sum_i \|Q_\phi(s_i, a_i) - y_i\|_2$

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

- Why could that be a bad idea?

- Epsilon greedy

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

- Why could that be a bad idea?
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*
Advanced tips for Q-learning

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

Slide partly borrowed from J. Schulman
The Plan

Recap

Q learning tricks

Improving Q learning

Case studies: games, robotics
Example: Deep Q Learning (DQN) Applied to Atari

- Human-level control through deep RL, Mnih et. al, 2013
- Uses target network and replay buffer
- One step back-up (no n-step returns)
- Became a popular benchmark since
Example: DQN
Example: Q-learning Applied to Robotics

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

Continuous action space?
Simple optimization algorithm ->
Cross Entropy Method (CEM)
QT-Opt: Q-learning at Scale

In-memory 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 jobs
$\min_\theta ||Q_\theta(s, a) - Q_T(s, a)||^2$

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

Slide adapted from D. Kalashnikov
QT-Opt: MDP Definition for Grasping

State: over the shoulder RGB camera image, no depth

Action: 4DOF pose change in Cartesian space + gripper control

Reward: binary reward at the end, if the object was lifted. Sparse. No shaping

Automatic success detection:

Slide adapted from D. Kalashnikov
QT-Opt: Setup and Results

7 robots collected 580k grasps  
Unseen test objects

96% test success rate!
Recap

Key learning goals:

• Practical Q learning implementation tricks
• Understanding the landscape of Q learning algorithms

Q learning implementation:

• Replay buffer & target networks
• Double Q-learning & n-step returns

Landscape of Q learning:

• Q learning w/ continuous actions
• Examples
Next

Any other way to learn a policy?

What about the dynamics of the environment?

Model-based RL