Non-linear Value Function Approximation: Double Deep Q-Networks

Alina Vereshchaka

CSE4/510 Reinforcement Learning
Fall 2019

avereshc@buffalo.edu

October 8, 2019

*Slides are based on Deep Reinforcement Learning: Q-Learning by Garima Lalwani, Karan Ganju, Unnat Jain. Illinois
1 Recap: DQN

2 Double Deep Q Network
Recap: Deep Q-Networks (DQN)

- Represent value function by deep Q-network with weights $w$
  \[ Q(s, a, w) \approx Q^\pi(s, a) \]

- Define objective function
  \[
  L(w) = \mathbb{E} \left[ \left( r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right)^2 \right]
  \]

- Leading to the following Q-leaning gradient
  \[
  \frac{\partial L(w)}{\partial w} = \mathbb{E} \left[ \left( r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right) \frac{\partial Q(s, a, w)}{\partial w} \right]
  \]

- Optimize objective end-to-end by SGD, using $\frac{\partial L(w)}{\partial w}$
Deep Q-Networks

DQN provides a stable solution to deep value-based RL

1. Use experience replay
2. Freeze target Q-network
3. Clip rewards or normalize network adaptive to sensible range
Deep Q-Network (DQN) Architecture

Naive DQN

- Q-value
- Network
- State
- Action

Optimized DQN used by DeepMind

- Q-value 1
- Q-value 2
- Q-value n
- Network
- State
DQN in Atari

1) Input:
4 images = current frame + 3 previous

2) Output: $Q(s, a_1)$
$Q(s, a_2)$
$Q(s, a_3)$
$\ldots$
$Q(s, a_{18})$
Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

For episode $= 1, M$ do
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1, T$ do
    With probability $\epsilon$ select a random action $a_t$
    otherwise select $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$
    Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j + 1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters $\theta$
    Every $C$ steps reset $\hat{Q} = Q$
  End For
End For
Table of Contents

1 Recap: DQN

2 Double Deep Q Network
Double Q learning

One (Estimator) Isn’t Good Enough?
One (Estimator) Isn’t Good Enough?

https://pbs.twimg.com/media/C5ymV2IVMAYtAev.jpg
Double Q-learning

Two estimators:

- **Estimator** $Q_1$: Obtain best actions
- **Estimator** $Q_2$: Evaluate $Q$ for the above action
Double Q-learning

Two estimators:

- Estimator $Q_1$: Obtain best actions
- Estimator $Q_2$: Evaluate $Q$ for the above action

What is the main motivation?
Double Q-learning

Two estimators:

- **Estimator $Q_1$:** Obtain best actions
- **Estimator $Q_2$:** Evaluate $Q$ for the above action

Chances of both estimators overestimating at same action is lesser
Double Q-learning

Two estimators:

- **Estimator $Q_1$:** Obtain best actions
- **Estimator $Q_2$:** Evaluate $Q$ for the above action

$$Q_1(s, a) \leftarrow Q_1(s, a) + \alpha (\text{Target} - Q_1(s, a))$$

**Q Target:** $r(s, a) + \gamma \max_{a'} Q_1(s', a')$
Double Q-learning

Two estimators:

- **Estimator \( Q_1 \):** Obtain best actions
- **Estimator \( Q_2 \):** Evaluate \( Q \) for the above action

\[
Q_1(s, a) \leftarrow Q_1(s, a) + \alpha (\text{Target} - Q_1(s, a))
\]

**Q Target:** \( r(s, a) + \gamma \max_{a'} Q_1(s', a') \)

**Double Q Target:** \( r(s, a) + \gamma Q_2(s', \arg \max_{a'} (Q_1(s', a'))) \)
Algorithm 1 Double Q-learning

1: Initialize $Q^A, Q^B, s$
2: repeat
3: Choose $a$, based on $Q^A(s, \cdot)$ and $Q^B(s, \cdot)$, observe $r, s'$
4: Choose (e.g. random) either UPDATE(A) or UPDATE(B)
5: if UPDATE(A) then
6: Define $a^* = \arg \max_a Q^A(s', a)$
7: $Q^A(s, a) \leftarrow Q^A(s, a) + \alpha(s, a)(r + \gamma Q^B(s', a^*) - Q^A(s, a))$
8: else if UPDATE(B) then
9: Define $b^* = \arg \max_a Q^B(s', a)$
10: $Q^B(s, a) \leftarrow Q^B(s, a) + \alpha(s, a)(r + \gamma Q^A(s', b^*) - Q^B(s, a))$
11: end if
12: $s \leftarrow s'$
13: until end
Two estimators:

- Estimator $Q_1$: Obtain best actions
- Estimator $Q_2$: Evaluate $Q$ for the above action
Algorithm 1: Double Q-learning (Hasselt et al., 2015)

Initialize primary network $Q_\theta$, target network $Q_\theta'$, replay buffer $D$, $\tau << 1$

for each iteration do

for each environment step do

Observe state $s_t$ and select $a_t \sim \pi(a_t, s_t)$

Execute $a_t$ and observe next state $s_{t+1}$ and reward $r_t = R(s_t, a_t)$

Store $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $D$

for each update step do

sample $e_t = (s_t, a_t, r_t, s_{t+1}) \sim D$

Compute target Q value:

$$Q^*(s_t, a_t) \approx r_t + \gamma Q_\theta(s_{t+1}, \text{argmax}_{a'} Q_{\theta'}(s_{t+1}, a'))$$

Perform gradient descent step on $(Q^*(s_t, a_t) - Q_\theta(s_t, a_t))^2$

Update target network parameters:

$$\theta' \leftarrow \tau \cdot \theta + (1 - \tau) \cdot \theta'$$
Are the Q-values accurate?