Machine Learning

Lecture 13: Value-Based Deep Reinforcement Learning

Nevin L. Zhang
lzhang@cse.ust.hk

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology

This set of notes is based on the references listed at the end and internet resources.

Outline

Motivation

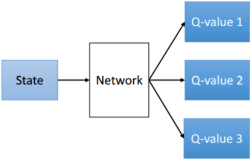
Q-learning:

- **Repeat**
  - Choose a for the state s (ϵ-greedy with $\arg\max_a Q(s,a)$)
  - Take action a, observe r and s'
  - Update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$
$$s \leftarrow s'$$

- Difficult to represent $Q(s,a)$ as a lookup table when the number of states is large.
  - Atari games: A state is an image with 210×160 pixels and each pixel has 128 possible color values.
  - The total number of states is: $(210 \times 160)^{128}$.
- Cannot determine actions for states never visited (such states are many).

Deep Q-Networks

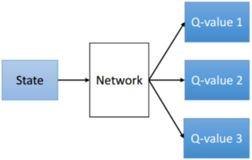


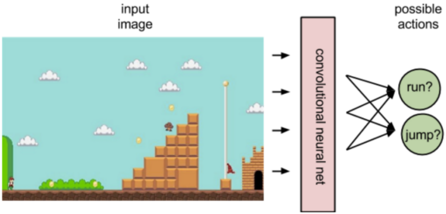
- In DQN, we aim to approximate the optimal state-action value function $Q^*(s, a)$ using a neural network with parameters θ:

$$Q(s, a; \theta) \approx Q^*(s, a)$$

- A function is defined over all possible states (images) without enumerating them.
- There is information for picking actions at any states.

Deep Q-Networks





- In DQN, we aim to approximate the optimal state-action value function $Q^*(s, a)$ using a neural network with parameters θ:

$$Q(s, a; \theta) = Q^*(s, a)$$

- **Key question**: How to learn θ from experiences with the environment?

$$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots$$

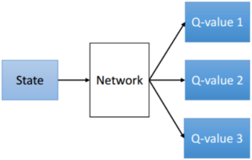
Outline

Learning DQN

Q-learning:

- **Repeat**
  - Choose a for the state s (ϵ-greedy with $\arg\max_a Q(s,a)$)
  - Take action a, observe r and s'
  - Update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$
$$s \leftarrow s'$$

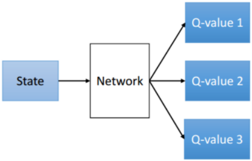


- We will learn θ iteratively, just as $Q(s,a)$ is learned iteratively in Q-learning.
- As the first step toward a DQN learning algorithm, we can change the the Q-value update step into a θ update step:

$$\theta \leftarrow \text{update}(\theta; s, a, s', r)$$

- To do so, we need come up with a objective function and update θ using its gradient.

Learning from experience tuple $e = (s, a, s', r)$



- We want to change θ so that $Q(s, a; \theta)$ gets closer to $Q^*(s, a)$.
- The problem is that we do not know the true target $Q^*(s, a)$.
- So, we use the **TD target**:

$$y = r(s, a) + \gamma \max_{a'} Q(s', a'; \theta)$$

  - Let Q' be the results if we run one value iteration on $Q(s, a; \theta)$. It is closer to Q^* than Q.
  - As explained in the previous lecture, y is an unbiased estimation of Q' based on the experience tuple.

Learning from experience tuple $e = (s, a, s', r)$

- Our target is now to push $Q(s, a; \theta)$ toward the TD target. So, we use the following loss function:

$$L(\theta) = (y - Q(s, a; \theta))^2 = ([r(s, a) + \gamma \max_{a'} Q(s', a'; \theta)] - Q(s, a; \theta))^2 \quad (1)$$

- The update rule for θ is:

$$\theta \leftarrow \theta - \alpha \nabla_\theta ([r(s, a) + \gamma \max_{a'} Q(s', a'; \theta)] - Q(s, a; \theta))^2 \quad (2)$$

where α is the learning rate.

Moving Target

- The TD target $y = r(s, a) + \gamma \max_{a'} Q(s', a'; \theta)$ depends on the parameters θ.
- Every time the parameters are updated, the target is also changed.
- So, learning θ with update rule (2) is like chasing a moving target.
- This makes learning unstable.

$$\theta_0$$

$$\theta_1 \leftarrow \theta_0 - \alpha \nabla_\theta (\underline{y_{\theta_0}} - Q(s_1, a_1; \theta))^2$$

$$\theta_2 \leftarrow \theta_1 - \alpha \nabla_\theta (\underline{y_{\theta_1}} - Q(s_2, a_2; \theta))^2$$

......

Target Network

- To deal with the moving target issue, create a **target network** that has the same structure as the DQN. Let θ^- be its parameters.
- Use the target network to compute the TD target y
- Set $\theta^- = \theta$ once in a while (every C steps).

$$\theta \leftarrow \theta - \alpha \nabla_\theta ([r(s,a) + \gamma \max_{a'} Q(s', a'; \theta^-)] - Q(s,a;\theta))^2 \tag{3}$$

$C=10$

t	θ	θ^-
1	θ_0	θ_0
2	θ_1	θ_0
:		:
10	θ_9	θ_0
11	θ_{10}	θ_{10}
:		θ_{10}

Deep Q-Learning with Target Network

Repeat:

- Take action a in current state s, observe r and s'
- Update the parameters:

$$\begin{aligned} \theta &\leftarrow \theta - \alpha \nabla_\theta([r(s,a) + \gamma \max_{a'} Q(s', a'; \theta^-)] - Q(s, a; \theta))^2 \\ s &\leftarrow s' \end{aligned}$$

- $\theta^- \leftarrow \theta$ in every C steps.

Outline

Experience Replay

- In Deep Q-Learning with target network
  - Only the latest experience tuple is used for parameter update.
- The idea of **experience replay** is:
  - Store experience tuples in a buffer
  - Use a random minibatch experience tuples for parameter update.

Deep Q-Learning with Target Network and Experience Replay

Repeat:

- Take action a in current state s, observe r and s'; add experience tuple (s, a, s', r) to a buffer D; $s \leftarrow s'$
- Sample a minibatch $B = \{s_j, a_j, s'_j, r_j\}$ from D.
- Update the parameters

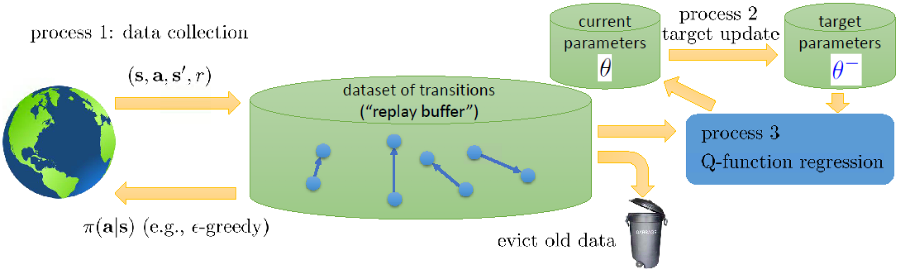
$$\theta \quad \leftarrow \quad \theta - \alpha \nabla_\theta \sum_j ([r(s_j, a_j) + \gamma \max_{a'_j} Q(s'_j, a'_j; \theta^-)] - Q(s_j, a_j; \theta))^2$$

- $\theta^- \leftarrow \theta$ in every C steps.

Advantages of Experience Replay

- With experience replay, each experience tuple is potentially used multiple times. Hence better data efficiency.
- With experience replay, each update improves $Q(s, a; \theta)$ using signals from multiple points (experience tuples). Hence faster convergence.
- Without experience replay, experiences tuples used in consecutive parameter updates are strongly correlated. Experience replay breaks the correlations and hence reduces variance of the updates.
- Experience replay avoids oscillations or divergence in the parameters.

A More General View of DQN

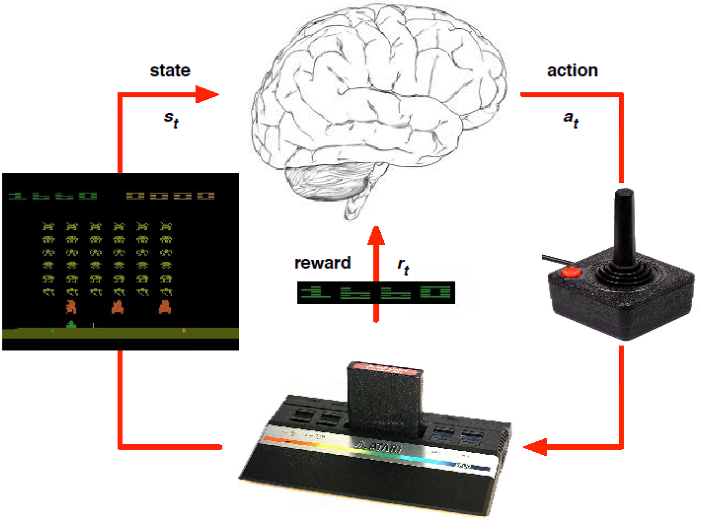


- Processes 1 and 3 run at the same pace, while Process 2 is slower.
- Old experiences are discarded when buffer is full.

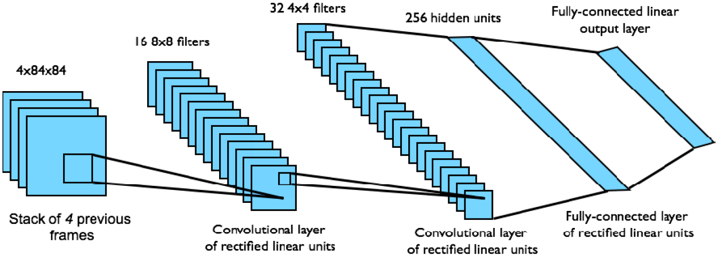
Outline

Atari Games

Space Invaders

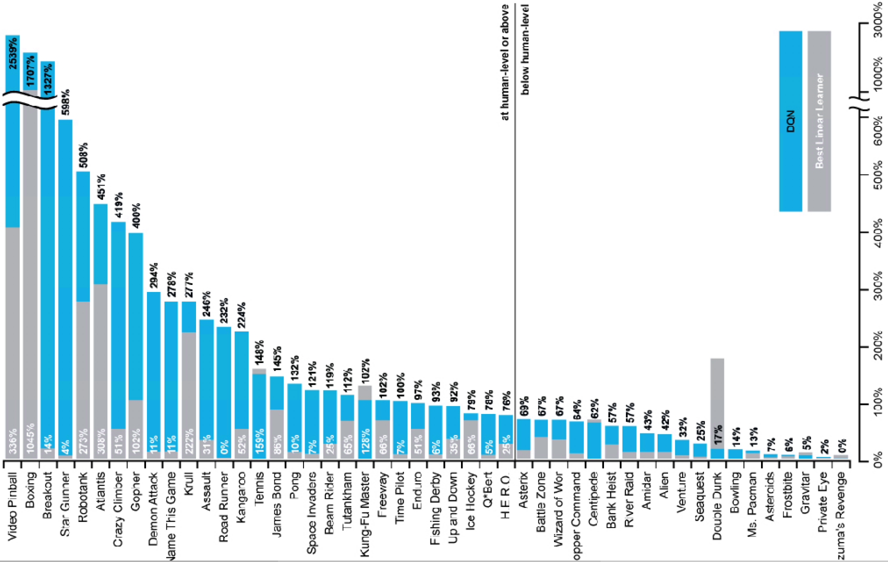


DAN in Atari



- End-to-end learning of values $Q(s; a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s; a)$ for 18 joystick/button positions
- Reward is change in score for that step
- Network architecture and hyperparameters fixed across all games

Comparable with or superior to human at majority of games



Results on Breakout

https://www.youtube.com/watch?v=V1eYniJ0Rnk

Outline

Overestimate of Value function

- In (3), we estimate TD Target as follows:

$$y_Q = r(s,a) + \gamma \max_{a'} Q(s', a'; \theta^-)$$

The subscript Q indicates that y_Q is the estimate used in Q-learning (DQN).

- The Q-values used are not accurate, and are influenced by random factors denoted as ω^-. So, we can write:

$$y_Q = r(s,a) + \gamma \max_{a'} Q(s', a'; \theta^-, \omega^-)$$

- If we run the process many times and take average of the y values, we get

$$E_{\omega^-}[y_Q] = r(s,a) + \gamma E_{\omega^-}[\max_{a'} Q(s', a'; \theta^-, \omega^-)]$$

Overestimate of Value function

- Ideally, we would like to first get a robust estimate of Q by take average over many run, i.e.,

$$E_{\omega^-}[Q(s', a'; \theta^-, \omega^-)]$$

and then use it to estimate the TD target:

$$\bar{y} = r(s, a) + \gamma \max_{a'} E_{\omega^-}[Q(s', a'; \theta^-, \omega^-)]$$

- On average, the y_Q overestimates the ideal value $\bar{y}$ because

$$\begin{aligned} E_{\omega^-}[y_Q] &= r(s, a) + \gamma E_{\omega^-}[\max_{a'} Q(s', a'; \theta^-, \omega^-)] \\ &\geq r(s, a) + \gamma \max_{a'} E_{\omega^-}[Q(s', a'; \theta^-, \omega^-)] \\ &= \bar{y} \end{aligned}$$

Note that. for any two random variables X_1 and X_2, $E[\max(X_1, X_2)] \geq \max(E[X_1], E[X_2])$

Double DQN

- $a^* = \arg\max_{a'} Q(s', a'; \theta, \omega)$

$$y_{DoubleQ} = r(s,a) + \gamma Q(s', a^*; \theta^-, \omega^-)$$

- Taking average, we get

$$E_{\omega^-}[y_{DoubleQ}] = r(s,a) + \gamma E_{\omega^-}[Q(s', a^*; \theta^-, \omega^-)]$$

This is closer to

$$\bar{y} = r(s,a) + \gamma \max_{a'} E_{\omega^-}[Q(s', a'; \theta^-, \omega^-)]$$

than $$E_{\omega^-}[y_Q] = r(s,a) + \gamma E_{\omega^-}[\max_{a'} Q(s', a'; \theta^-, \omega^-)]$$

An Analogy

- TD Target in DQN:

$$y_Q = r(s,a) + \gamma \max_{a'} Q(s', a'; \theta^-)$$

The second term estimates future optimal value of s' in two steps:

- Picks next action a', and
- Evaluate Q(s', a')

In both steps, we use θ^-

An Analogy

- Task: Estimate highest exam score of a class given current situation s'.
- Method 1: Ask one teacher θ^- to
  - Estimate the score of each student a': $Q(s', a'; \theta^-)$.
  - Pick best student: $a^* = \arg\max_{a'} Q(s', a'; \theta^-)$
  - Report $Q(s', a^*; \theta^-)$.

  Problem: Best student picked by teach θ^- might not the best after all. Score for a^* is overestimated, and hence max score of class is overestimated.
- Method 2:
  - Ask one teacher θ to pick best student: $a^* = \arg\max_{a'} Q(s', a'; \theta)$
  - Ask one teacher θ^- to estimate her score: $Q(s', a^*; \theta^-)$

Double DQN

- Update rule for DQN:

$$\theta \leftarrow \theta - \alpha \nabla_\theta([r(s,a) + \gamma \max_{a'} Q(s', a'; \theta^-)] - Q(s,a;\theta))^2$$

Or, equivalently:

$$\theta \leftarrow \theta - \alpha \nabla_\theta([r(s,a) + \gamma Q(s', a^*; \theta^-)] - Q(s,a;\theta))^2$$

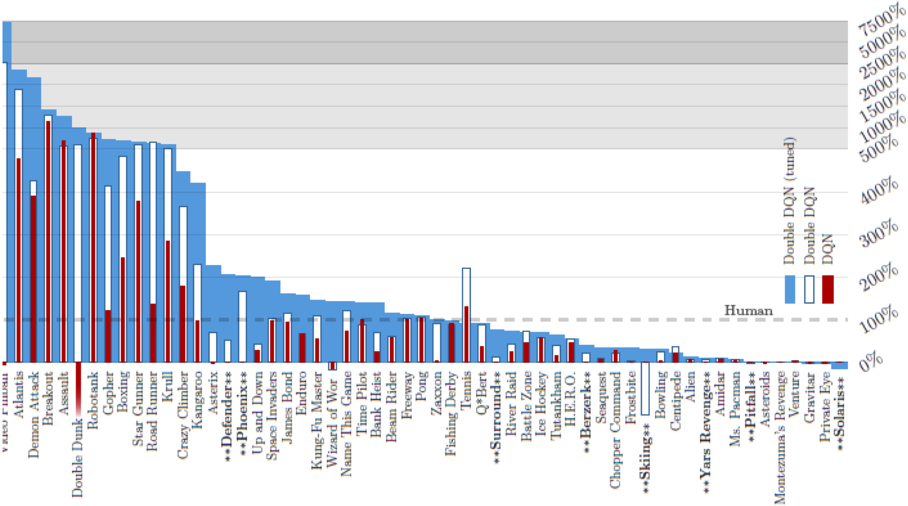
where $a^* = \arg\max_{a'} Q(s', a'; \theta^-)$.

- Update rule for Double DQN:

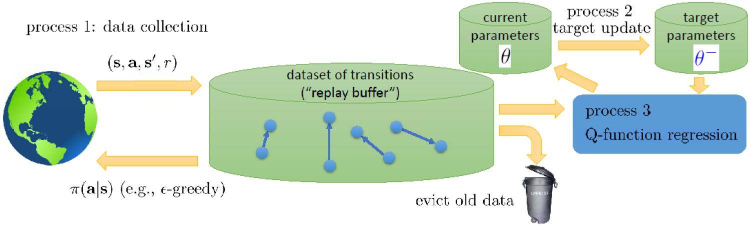
$$\theta \leftarrow \theta - \alpha \nabla_\theta([r(s,a) + \gamma Q(s', a^*; \theta^-)] - Q(s,a;\theta))^2 \tag{4}$$

where $a^* = \arg\max_{a'} Q(s', a'; \theta)$.

DQN versus Double DQN



Why Prioritized Experience Replay?



- In DQN and Double DQN, experience transitions are uniformly sampled from a replay memory for the purpose of Q-function regression. (Process 3)
- This approach does not take the significance of the experience transitions into consideration.

Prioritized Experience Replay

- The significance of a experience tuple (s, a, s', r) is measured using the **TD error**:

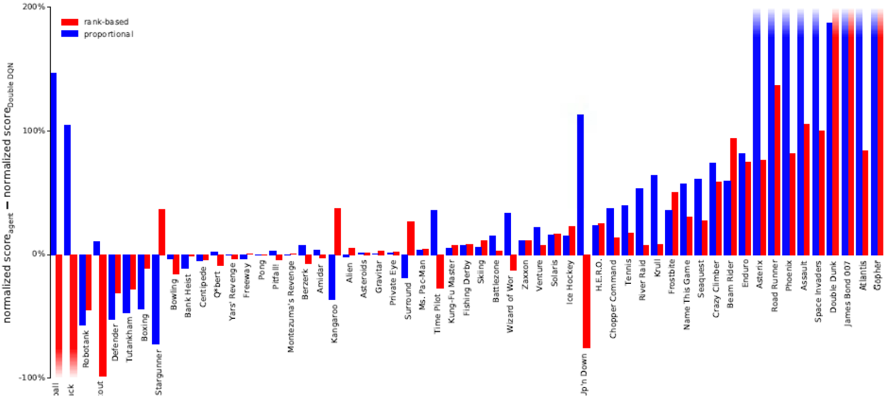
$$\delta = [r(s, a) + \gamma \max_{a'} Q(s', a'; \theta)] - Q(s, a; \theta)$$

- Let $\{s_i, a_i, s'_i, r_i)\}$ be the collection of experience tuples in the replay memory.
- For each update of θ, experience tuples are sampled using the following distribution:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

- p_i is a function of δ_i: $p_i = |\delta_i| + \epsilon$ or $p_i = \frac{1}{rank(i)}$
- α determines how much prioritization is used.$\alpha = 0$ amounts to uniform sampling.

Prioritized Experience Replay Improves Double DQN in Most Cases

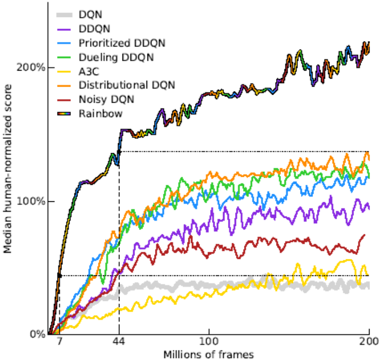


Code Example

github.com/vmayoral/basic_reinforcement_learning/blob/master/tutorial6/README.md

RAINBOW on Atari Games [1]

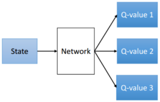
RAINBOW = Double DQN + Prioritized Replay + Dueling Networks + Multi-step Learning + Distributional RL:



[1] Hessel *et al.* 2017: Rainbow: Combining Improvements in Deep Reinforcement Learning

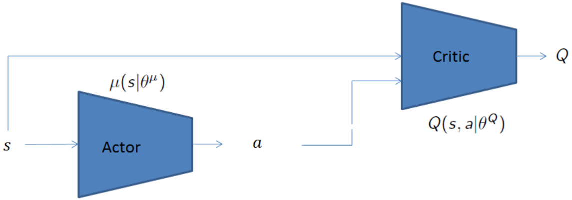
Deep Deterministic Policy Gradient (DDPG) [2]

- DQN using a neural network to represent $Q(s, a)$



$$\pi(s) = \arg\max_a Q(s, a)$$

- Cannot deal with continuous actions.
- DDPG: One network for $Q(s, a)$: **Critic** $Q(s, a|\theta^Q)$
  another network for $\pi(s) = \arg\max_a Q(s, a)$: **Actor** $\mu(s|\theta^\mu)$



[2] Lillicrap *et al.* 2016, CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING

Deep Deterministic Policy Gradient (DDPG)

for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $\boxed{y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})}$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\boxed{\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}}$$

- DQN: $y = r(s,a) + \gamma \max_{a'} Q(s', a'; \theta)$
- Actor update: $\max_{\theta^\mu} J = Q(s, \mu(s|\theta^\mu)|\theta^Q)$

Deep Deterministic Policy Gradient (DDPG)

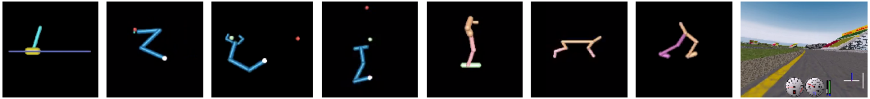


Figure 1: Example screenshots of a sample of environments we attempt to solve with DDPG. In order from the left: the cartpole swing-up task, a reaching task, a gasp and move task, a puck-hitting task, a monoped balancing task, two locomotion tasks and Torcs (driving simulator). We tackle all tasks using both low-dimensional feature vector and high-dimensional pixel inputs. Detailed descriptions of the environments are provided in the supplementary. Movies of some of the learned policies are available at https://goo.gl/J4PIAz.

- **Twin Delayed DDPG (TD3)**: Substantially improved version of DDPG https://spinningup.openai.com/en/latest/algorithms/td3.html

References

- Sergey Levine. Deep Reinforcement Learning, http://rail.eecs.berkeley.edu/deeprlcourse/, 2018.
- David Silver. Tutorial: Deep Reinforcement Learning, http://hunch.net/∼beygel/deep_rl_tutorial.pdf, 2016.
- Tambet Matiisen. Demystifying Deep Reinforcement Learning. https://ai.intel.com/demystifying-deep-reinforcement-learning/, 2015.
- Watkins. (1989). Learning from delayed rewards: introduces Q-learning
- Riedmiller. (2005). Neural fitted Q-iteration: batch-mode Q-learning with neural networks
- 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.
- Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized experience replay. In ICLR, 2016.
- Wang, Schaul, Hessel, van Hasselt, Lanctot, de Freitas (2016). Dueling network architectures for deep reinforcement learning: separates value and advantage estimation in Q-function.