② Deep Q-learning DQN

Published in 2013 - 2015 Nature

Approach: Q-learning → what if q-values are represented by a neural network ?

Bellman equations: Q^* is the only solution to that equation

$$Q^*(s,a) = \mathbb{E}_{\substack{r,s' \\ \sim \Delta(s,a)}}\left[r + \gamma \max_{a'\in A} Q^*(s',a')\right]$$

$$\mathcal{L}(\theta) = \frac{1}{2}\left(\mathbb{E}_{\substack{(s,a,r,s') \\ \sim \sigma_\theta}}\left[\overbrace{r + \gamma \max_{a'\in A} Q_{\theta'}(s',a')}^{\substack{\theta' \\ \text{target model}}} - \overbrace{Q_\theta(s,a)}^{\substack{\theta \\ \text{prediction model}}}\right]\right)^2$$

two occurrences of θ

if $\mathcal{L}(\theta)=0$ then Q_θ satisfies Bellman equatio

$\Rightarrow Q_\theta = Q_*$

Key idea: use two networks!

↳ similarly to off-policy learning
↳ similarly to the maximisation bias

$$\mathcal{L}(\theta) = \frac{1}{2}\left(\underset{(s,a,r,s') \sim \sigma_\theta}{\mathbb{E}}\left[\underbrace{r + \gamma \max_{a' \in A} Q_{\theta'}(s',a')}_{\text{fixed}} - Q_{\theta}(s,a)\right]\right)^2$$

Implementation details:

- (prioritised) experience replay
- reward clipping

DQN algorithm

initialise two models : prediction model θ
target model θ'

$$\theta \hookrightarrow \sigma_\theta(s) = \underset{a \in A}{\arg\max}\; q_\theta(s,a)$$

Iterate:

constant: batch size

First step: Batch Sampling (B)

First step: Batch Sampling (B)

Simulate the environment using ε-greedy from σ_Θ giving full trajectories

→ we break them down into B steps:

(s, a, r, s')

Second step: Update the networks

- at every iteration, update the prediction model:

Compute $\hat{\mathcal{L}}(\Theta)$

$$\mathcal{L}(\Theta) = \frac{1}{2}\left(\mathbb{E}_{\substack{(s,a,r,s') \\ \sim \sigma_\Theta}}\left[\underbrace{r + \gamma \max_{a' \in A} Q_{\Theta'}(s',a')}_{\text{fixed}} - Q_\Theta(s,a)\right]\right)^2$$

$\hat{\mathcal{L}}(\Theta) = \frac{1}{2}$ average over steps from the batch:

(s,a,r,s'): $r + \gamma \max_{a' \in A} Q_{\Theta'}(s',a') - Q_\Theta(s,a)$

if you have a neural network / ML model parametrised by Θ, you have functions for computing ∇_Θ and for updating:

$$\hat{\mathcal{L}}(\Theta) \xrightarrow{\text{gradient}} \nabla_\Theta \hat{\mathcal{L}}(\Theta) \xrightarrow{\text{optimisation step}} \Theta_{new}$$

* every N iterations, update the target model :

$\theta' \leftarrow \theta$

parameters of the prediction model

parameters of the target model

$N \simeq 10$