DEEP Q-NETWORKS : DQN

Published in 2013 - 2015 Nature

TERMINOLOGY

probability	p	$(0, 1)$
logit	$\log\left(\frac{p}{1-p}\right)$	$(-\infty, +\infty)$

$$
\begin{tikzcd}
p \arrow[r, bend left, "\log\left(\frac{p}{1-p}\right)"] & \ell \arrow[l, bend left, "\frac{\exp(\ell)}{\exp(\ell)+1}"]
\end{tikzcd}
$$

Question: What if S is infinite?

↳ Change of representation

Deterministic policy

dimension of observation space — neural network Θ — number of actions

Θ: set of parameters

$$q_\Theta : S \times A \to \mathbb{R}$$

← value

$$\sigma_\Theta(s) = \operatorname*{argmax}_{a \in A} q_\Theta(s,a)$$

Stochastic policy

dimension of observation space — neural network Θ — number of actions

$$p_\Theta : S \times A \to \mathbb{R}$$

← logit

$$\sigma_\Theta(s) = \text{distribution}\underbrace{(a \mapsto p_\Theta(s,a))}$$

apply softmax here to get probabilities

Formulation:

$$\min_{\theta} \mathcal{L}(\theta)$$

loss

Update:

params

$$\text{NEW} = \text{OLD} + \alpha\,(\text{CURRENT} - \text{OLD})$$

↳ includes "gradient ascent algorithms"

Typical approach: stochastic gradient descent

Sketch:

(1) Batch sampling:
using the current policy, find <u>SETS</u> of trajectories

(2) Batch update:
using the batch, update the parameters

KEY IDEA BEHIND Q-LEARNING

Bellman equation:

Q^* is the only solution to that equation

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

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

two occurrences of θ

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

$\Rightarrow Q_\theta = Q_*$

Key idea: use two networks!

↳ similarly to off-policy learning

↳ similarly to the maximisation bias

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

Implementation details:

- (prioritised) experience replay
- reward clipping

DQN algorithm

initialise two models : prediction model Θ
target model Θ'

$$\Theta \hookrightarrow \quad \sigma_\Theta(s) = \arg\max_{a \in A} q_\Theta(s,a)$$

Iterate:

constant: batch size

<u>First step</u>: 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 $\hat{\mathcal{L}}(\theta)$

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

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

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

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

$N \simeq 10$