

DEEP Q-NETWORKS : DQN

Published in 2013 - 2015 Nature

TERMINOLOGY

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

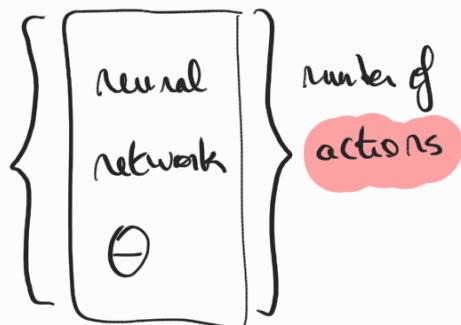
$$\begin{array}{c} \log \left(\frac{P}{1-P} \right) \\ \curvearrowright \\ P \end{array}$$
$$\frac{\exp(l)}{\exp(l)+1}$$
$$\curvearrowleft l$$

Question: What if S is infinite?

↪ change of representations

Deterministic policy

dimension
of
observation
space



$$q_\theta: S \times A \rightarrow \mathbb{R}$$

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

Θ : set of parameters

Stochastic policy

dimension
of
observation
space



$$p_\theta: S \times A \rightarrow \mathbb{R}$$

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

apply softmax here
to get probabilities

Formulation:

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

loss

Update: ↘ parameters

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

↳ includes "gradient ascent algorithms"

Typical approach : **stochastic gradient descent**

Sketch:

(1) **Batch Sampling :**

using the current policy, find SETS of trajectories

(2) **Batch update :**

using the batch, update the parameters

KEY IDEA BEHIND Q-LEARNING

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

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

$r, s' \sim \Delta(s, a)$

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

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

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

fixed

Implementation details:

- (prioritised) experience replay
- reward clipping

DQN algorithm

initialise two models : prediction model Θ
target model Θ'

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

Iterate:

First step: Batch Sampling (B)

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

→ we break them down into B steps:

$$(s, a, r, s')$$

Constant: batch size

Second step: Update the networks

- * at every iteration, update the prediction model:

compute $\hat{\mathcal{L}}(\theta)$

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

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

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

if you have a neural network / ML model parametrised by θ , you have functions for

computing ∇_θ and for updating:

$$\hat{\mathcal{L}}(\theta) \xrightarrow{\text{gradient}} \nabla_\theta \hat{\mathcal{L}}(\theta) \xrightarrow{\text{optimisation Step}} \theta_{\text{new}}$$

- * every N iterations, update the target model:

$$\theta' \leftarrow \theta \quad \begin{matrix} \leftarrow \\ \text{parameters of the} \\ \text{prediction model} \end{matrix}$$

\uparrow
parameters of
the target
model

$N \approx 10$