

# DEEP Q-NETWORKS : DQN

Published in 2013 - 2015 Nature

## TERMINOLOGY

probability  
logit

$p$   
 $\log\left(\frac{p}{1-p}\right)$

$$(0, 1)$$

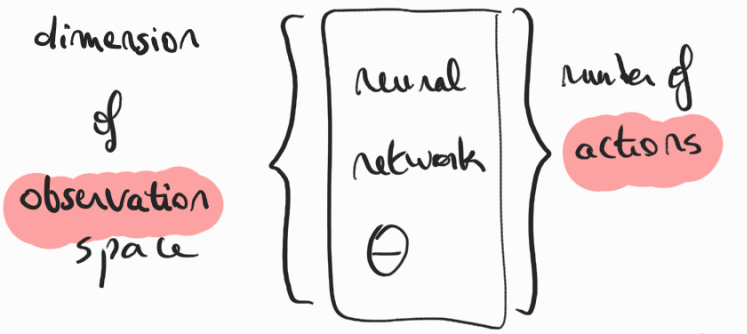
$$(-\infty, +\infty)$$

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

Question: What if  $S$  is infinite?

↳ change of representation

### Deterministic policy



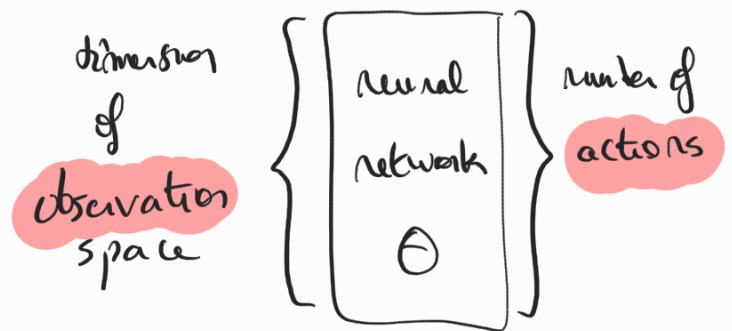
$$q_{\Theta}: S \times A \rightarrow \mathbb{R}$$

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

← value

$\Theta$ : set of parameters

### Stochastic policy



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

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

logit

apply softmax here to get probabilities

Formulation:

$$\min_{\theta} \underbrace{\mathcal{L}(\theta)}_{\text{loss}}$$

Update: parameters

$$\begin{bmatrix} \text{NEW} = \text{OLD} + \alpha (\text{CURRENT} - \text{OLD}) \\ \hookrightarrow \text{includes "gradient ascent algorithms"} \end{bmatrix}$$

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

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

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

if  $\mathcal{L}(\theta) = 0$  then  $Q_\theta$  satisfies Bellman equation  
 $\Rightarrow Q_\theta = Q_*$

Key idea: use two networks!

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

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

## Implementation details:

- (prioritized) experience replay
- reward clipping

## DQN algorithm

initialise two models : prediction model  $\theta$   
target model  $\theta'$

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

Iterate:

First step: Batch Sampling ( $B$ )

constant: batch size

Simulate the environment using  $\epsilon$ -greedy from  $\sigma_{\theta}$   
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  $\hat{\mathcal{L}}(\theta)$

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

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

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

if you have a neural network / ML model parametrised by  $\theta$ , you have functions for

computing  $\nabla_\theta$  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:

$$\theta' \leftarrow \theta$$

parameters of the prediction model

parameters of the target model

$$N \approx 10$$