3.6 DEEP Q-LEARNING

Nel presente operato, ci si concentrerà sulla risoluzione di un problema di *Reinforcement Learning* utilizzando la tecnica appena presentata, chiamata *Q-Learning*. Di seguito vengono forniti alcuni concetti utili a comprendere il funzionamento di una sua evoluzione che combina i concetti del *Deep Learning* e quelli del *Reinforcement Learning* dando vita al cosiddetto *Deep Q-Learning*.

3.6.1 Tabular Action-Value

La *Action-Value Function* $Q(S_t, A_t)$ durante l'esecuzione dell'algoritmo *Q-Learning* viene stimata e aggiornata in modo incrementale. Per ogni coppia stato S_t e azione A_t il valore della *Action-Value* viene memorizzato in una tabella, detta *Tabular Action-Value*, all'interno della quale le colonne contengono le azioni, mentre le righe i possibili stati (Figura 3.8).

	A_1	A_2	---	A_n
S_1	$Q(S_1, A_1)$	$Q(S_1, A_2)$	---	$Q(S_1, A_n)$
S_2	$Q(S_2, A_1)$	$Q(S_2, A_2)$	---	$Q(S_2, A_n)$
---	---	---	---	---
S_n	$Q(S_n, A_1)$	$Q(S_n, A_2)$	---	$Q(S_n, A_n)$

Figura 3.8: Esempio di *Q-Table*

Tuttavia, questo approccio risulta applicabile solamente nel caso in cui vi siano un numero ridotto di stati e azioni, poiché qualora siano presenti molteplici stati o questi siano continui, si riscontrano due problemi: il primo riguarda l'elevata memoria richiesta per memorizzare la *Q-table*, mentre il secondo consiste nelle elevate tempistiche necessarie per stimare le coppie stato-azione in modo accurato.

3.6.2 Funzioni di approssimazione

Per risolvere questi problemi è necessario rappresentare in forma compatta la *Q-value* in modo tale che sia utilizzabile in più casistiche, ottenendo quindi una generalizzazione dell'esperienza acquisita. La soluzione consiste nello stimare la *Action-Value Function* utilizzando una funzione di approssimazione, nella quale θ sono i parametri della funzione.

$$\tilde{q}(s, a; \theta) \approx q_\pi(s, a) \tag{3.34}$$

3.6.3 Deep Q-Network

In passato venivano utilizzati modelli di *Machine Learning* per la funzione di approssimazione descritta in precedenza (Sezione 3.6.2). Tuttavia, attualmente vengono preferite le reti neurali. In Figura 3.9 viene mostrato il funzionamento delle reti neurali come approssimatore.

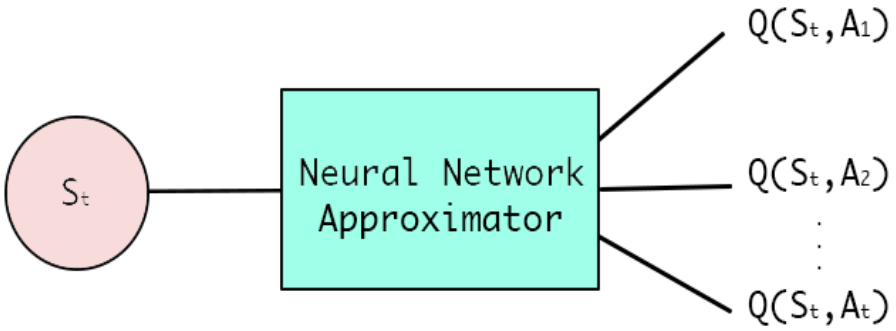

Figura 3.9: Funzionamento *Deep Q-Network*

Ciononostante, la funzione di approssimazione può risultare non stabile a causa di dati non stazionari, non indipendenti e identicamente distribuiti.

In (Mnih et al., 2015) vengono proposti due metodi che permettono la risoluzione di questo problema.

Experience Replay

In questo primo metodo viene salvata in memoria l'esperienza acquisita al tempo t indicata con $e_t = (S_t, A_t, R_t, S_{t+1})$ e ottenuta dall'interazione tra agente e ambiente. L'aggiornamento dei pesi della rete neurale non avviene dopo ogni singola transizione, bensì dopo un sottoinsieme di transizioni definito *mini-batch*.

Target Model

Nel secondo metodo durante l'aggiornamento dei pesi della rete si utilizzano in simultanea due funzioni di approssimazione, ovvero due reti uguali ma con pesi differenti: l'errore viene calcolato dalla differenza tra esse.

Infine, analogamente al *Sarsa* e al *Q-Learning*, viene illustrato in Figura 3.10 lo pseudocodice del *Deep Q-Learning* presentato in (Mnih et al., 2015).

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Figura 3.10: Pseudocodice *Deep Q-Learning*