3.5 TEMPORAL DIFFERENCE LEARNING

I *Temporal Difference* (TD) sono degli algoritmi di *Reinforcement Learning* che combinano le caratteristiche della Programmazione Dinamica (DP) e del Montecarlo (MC) e vengono considerati da (Sutton et al., 2018) come uno degli strumenti fondamentali del *Reinforcement Learning*.

TD e MC sono entrambi *Model-free* e apprendono le informazioni necessarie dall'esperienza. Tuttavia, i TD non devono attendere la fine di un episodio per effettuare l'aggiornamento della *Value Function*, tanto è vero che questo viene effettuato dopo ogni singolo *timeframe*, proprio come avviene nel DP.

Di seguito vengono brevemente illustrate le due fasi dei TD:

Policy Evaluation

La differenza sostanziale rispetto a MC consiste nell'utilizzare una funzione di aggiornamento contenente l'equazione di *Bellman*. Di seguito viene mostrata la formula:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{3.31}$$

Nella quale:

- $R_{t+1} + \gamma V(S_{t+1})$ è chiamato *TD Target*.
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ è chiamato *TD error*.

Policy Evaluation

Per quanto riguarda questa fase viene utilizzata la ϵ-*greedy* illustrata in precedenza (Sezione 3.4.2).

3.5.1 On e Off Policy

Si ritiene necessario evidenziare l'esistenza di due approcci di apprendimento differenti all'interno dei TD:

- *On-Policy Learning*, nel quale si apprende una *policy* π dall'esperienza ricavata con *policy* π.
- *Off-Policy Learning* dove si apprende una *policy* π dall'esperienza ricavata con una *policy* differente, ossia μ.

Di seguito vengono descritti i due principali algoritmi utilizzati per risolvere i problemi di RL.

3.5.2 Sarsa

L'algoritmo *Sarsa* presentato in (Rummery A. e Niranjan, 1994) segue un approccio di tipo *On-Policy* per la risoluzione del problema di RL.

Nella fase di *Policy Evaluation*, come visto in precedenza, risulta più utile stimare la *Action-Value Function* rispetto alla *State-Value Function*. Di seguito viene mostrata la formula del suo aggiornamento:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{3.32}$$

Dove: $\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)$ è chiamato *TD error*.

In Figura 3.4 viene mostrato lo pseudocodice di *Sarsa* presente in (Sutton et al., 2018).

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Figura 3.4: Pseudocodice *Sarsa*

Funzionamento
In Figura 3.5 viene mostrato il funzionamento di *Sarsa*: sulla base dell'azione A_t dello stato S_t e la *reward* ottenuta R_{t+1} viene scelta un'azione A_{t+1} dello stato S_{t+1}. Queste informazioni vengono utilizzate in seguito per effettuare l'aggiornamento di Q(s,a). Questo processo si ripete partendo dall'azione A_{t+1}

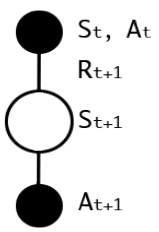

Figura 3.5: Funzionamento *Sarsa*

3.5.3 Q-Learning

L'algoritmo *Q-Learning* presentato in (Watkins, 1989) segue invece un approccio di tipo *Off-Policy* per la risoluzione del problema di RL.

A differenza di *Sarsa*, il *Q-Learning* effettua l'aggiornamento della *Action-Value Function* scegliendo l'azione che ritiene più promettente e per farlo utilizza un'approccio *greedy*. Di seguito viene mostrata la formula dell'aggiornamento:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, A_t) - Q(S_t, A_t)] \tag{3.33}$$

In Figura 3.6 viene mostrato lo pseudocodice del *Q-Learning* presente in (Sutton et al., 2018).

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figura 3.6: Pseudocodice *Q-Learning*

Funzionamento
In Figura 3.7 viene mostrato il funzionamento del *Q-Learning*: sulla base dell'azione A_t dello stato S_t e la *reward* ottenuta R_{t+1}, a differenza di ciò che avviene in *Sarsa*, viene scelta l'azione con il massimo *Q-Value* nello stato S_{t+1}. Queste informazioni vengono utilizzate in seguito per effettuare l'aggiornamento di Q(s,a). Il processo si ripete partendo dall'azione A_{t+1}

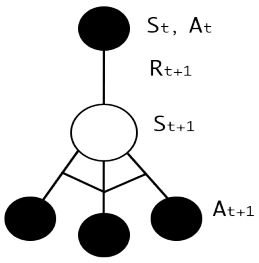

Figura 3.7: Funzionamento *Q-Learning*