Machine Learning - Sistemi Intelligenti per Internet

Reinforcement Learning

Q-Learning | [Double] Deep Q-Learning

Gianmaria Del Monte - Omar Elsayed

Markov Decision Process

Un *Markov Decision Process* (MDP) descrive un ambiente utile per il reinforcement learning, dove un ambiente è completamente osservabile.

Uno stato S_t è detto *Markoviano* se $P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, \ldots, S_t]$.

In altre parole, dato uno stato attuale S_t, lo stato successivo S_{t+1} è dipendente solo dallo stato precedente S_t e non dalla storia passata.

Formalmente, un MDP è una quintupla (S, A, P, R, γ):

- S è un insieme finito di *stati*;
- A è un insieme finito di *azioni*;
- P è una *matrice di probabilità di transizioni di stato*, in cui ogni elemento è del tipo $P^a_{ss'} = P[S_{t+1} = s' \mid S_t = s, A_t = a]$;
- R è la *funzione di reward* $R^a_s = E[R_{t+1} \mid S_t = s, A_t = a]$;
- γ è il *discount factor*, $\gamma \in [0, 1]$.

Il *return value* G_t è la somma totale dei reward scontata per il fattore γ dallo step t:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Si definisce policy π una distribuzione di probabilità delle azioni dati gli stati:

$$\pi(a|s) = P[A_t = a \mid S_t = s].$$

Si definisce *state-value function* $v_\pi(s)$ il valore di return atteso partendo dallo stato s, seguendo la policy π:

$$v_\pi(s) = E_\pi[G_t \mid S_t = s].$$

La state-value function può essere decomposta in due parti, secondo l'approccio di *Bellman*:

1. reward immediato R_{t+1}
2. valore *scontato* dello stato successivo $\gamma v_\pi(s_{t+1})$

$$v_\pi(s) = E[R_{t+1} + \gamma v_\pi(s_{t+1}) \mid S_t = s].$$

L'obiettivo in un MDP è quindi quello di scegliere una sequenza di azioni che massimizzino il reward totale.

Si definisce *action-value function* $q_\pi(s,a)$ il valore di return atteso partendo dallo stato s, scegliendo un'azione a e seguendo la policy π:

$$q_\pi(s,a) = E_\pi[G_t \mid S_t = s,\ A_t = a].$$

L'obiettivo in un MDP è quello di trovare la migliore sequenza di azioni, in modo tale da massimizzare la somma totale dei reward.

Si definiscono, quindi, l'*optimal state-value function* $v_*(s)$ il massimo delle *value function* tra tutte le policy, $v_*(s) = \max_\pi v_\pi(s)$, e l'*optimal action-value function* $q_*(s,a)$ il massimo delle *action-value function* tra tutte le policy, $q_*(s,a) = \max_\pi q_\pi(s,a)$.

È chiaro che la conoscenza di q_* ci permette di definire il comportamento ottimale in un MDP, per cui è possibile scegliere la migliore azione. La policy ottimale si può ottenere, infatti, scegliendo l'azione a che massimizza $q_*(s,a)$:

La funzione q_* può essere espressa anch'essa secondo la forma di Bellman:

$$q_*(s,a) = R^a{}_s + \gamma \sum_{s' \in S} P^a{}_{ss'} \max_{a'} q_*(s',a')$$

Quest'ultima è un'equazione non lineare, per cui non è risolvibile usando tecniche dell'algebra lineare, per cui si usano metodi iterativi per approssimare via via il valore di q_*:

- value iteration
- q-learning
- sarsa

Q-Learning

L'algoritmo di apprendimento per rinforzo *Q-Learning* utilizza una tabella chiamata *Q-table* dove vengono memorizzati i valori Q associati ad ogni coppia stato-azione. Le righe della tabella sono gli stati e le colonne le azioni.

L'obiettivo di questo algoritmo è quello di poter apprendere questi valori con l'esperienza, andando a modificarli in base ai reward ottenuti dopo aver svolto una certa azione in un certo stato.

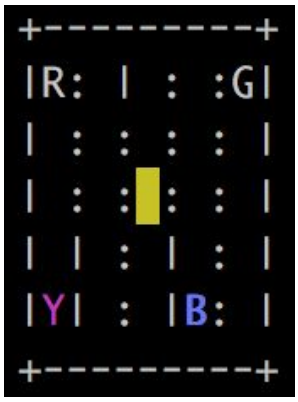
La formula usata per aggiornare la Q-table è la seguente:

$$Q(s_t,\ a_t) \leftarrow (1 - \alpha) * Q(s_t,\ a_t) + \alpha * (R_{t+1} + \gamma * max\ Q(s_{t+1},\ a_{t+1}))$$

dove:

- α è il tasso di apprendimento
- γ è il discount factor
- R_{t+1} è il reward.

Taxi Driver

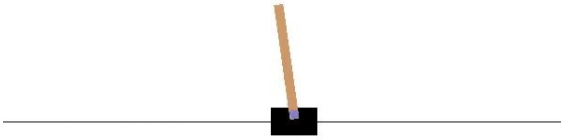


In questo Gym all'agente è richiesto di muovere un taxi in un ambiente a griglia, di raccogliere passeggeri da alcuni punti predefiniti e di portarli alla loro destinazione. Data la dimensione della griglia (5 x 5) e il numero di azioni limitato (6, 4 di movimento, uno per caricare e un altro per scaricare un passeggero) e il numero di posizioni dove può trovarsi il passeggero (5), il numero di stati è 500. Questo rende possibile l'utilizzo di una tabella dove salvare i valori di Q per ogni stato.

Questo tipo di problemi, con pochi stati, è facilmente risolvibile con Q-Learning, infatti siamo riusciti in pochissimi istanti ad avere una Q-table ottima che permette di risolvere il problema in pochissimi passi.

Cart-Pole

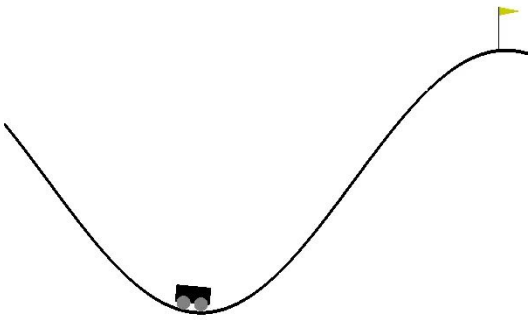
Il Cart-Pole è un problema in cui bisogna mantenere in equilibrio un pendolo verticale spostando un carrello a sinistra o a destra. Essendo lo spazio di osservazione di questo Gym continuo, si ha un numero infinito di stati, rendendo così necessario discretizzare lo l'osservazione per poter utilizzare la Q-table.



La discretizzazione, però, non rende possibile apprendere correttamente i valori di Q, andando di fatti ad introdurre degli errori che portano l'agente a non riuscire a raggiungere un punteggio decente.

Mountain Car

In questo Gym bisogna portare sopra una collina una macchina sotto potenziata, per poterlo fare bisogna riuscire a sfruttare la gravità per aumentare l'accelerazione. Anche in questo Gym, come Cart-Pole, gli stati sono infiniti e bisogna discretizzare l'osservazione per poter creare la Q-table. L'ambiente di questo Gym ha due colline, una dove è presente l'obiettivo ed un'altra da poter usare per prendere lo slancio.



L'agente che abbiamo addestrato riesce a raggiungere sempre l'obiettivo dondolando nella valle per poter salire la collina e raggiungere l'obiettivo.

Deep Q-Learning

Nel Deep Q-Learning si utilizza una rete neurale profonda per approssimare la funzione di Bellman, dove l'input è lo stato osservato e l'output sono le Q-value associate ad ogni azione.

Cart-Pole

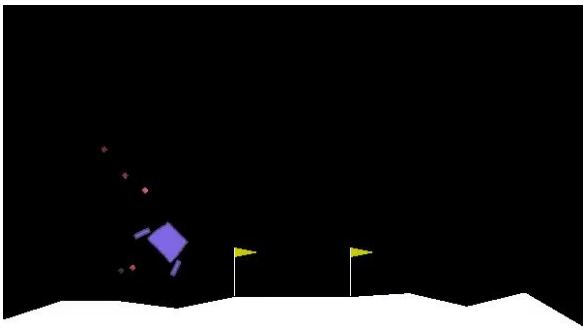
Cart-Pole con il DQL riesce a convergere abbastanza velocemente ottenendo ottimi risultati, raggiunge una percentuale di successo del 100%.

Mountain Car

Anche Mountain Car con DQL migliora notevolmente, raggiunge ottimi risultati già dopo 2000 episodi. Una differenza che si può notare rispetto al Q-Learning classico è che l'agente impara a salire leggermente la collina di sinistra per poter poi raggiungere l'obiettivo, di fatti portando a termine il compito in minor tempo.

Lunar Lander

Nel Gym Lunar Lander bisogna far atterrare un lander sulla superficie lunare senza arrecargli danni. Il modulo di atterraggio ha 3 motori da utilizzare, 2 laterali e uno in basso, e deve imparare ad utilizzarli nel modo corretto nel punto indicato (sempre di coordinate [0,0]). Una particolarità del Gym è la randomizzazione del terreno, che impedisce di far apprendere all'agente la conformazione del terreno.



Con DQL l'agente non riesce ad imparare ad atterrare correttamente, bensì tende a volare via o schiantarsi.

Double Deep Q-Learning

Questo algoritmo è una variante del DQL con la differenza che vengono usate due reti neurali. Ogni rete viene usata per addestrare l'altra, vengono chiamate solitamente *train* e *target*. La rete target viene usata per scegliere l'azione da eseguire e la rete train viene usata per valutare l'azione, dopo un certo numero di episodi si copiano i pensi della rete train in quella target, in questo modo stabilizzando l'addestramento.

Mountain Car

Mountain Car con DDQL riesce a raggiungere risultati ottimali già dopo 500 episodi, aggiornando la rete target ogni 30 episodi. Anche in questo caso ci mette minor tempo di Q-Learning classico a raggiungere l'obiettivo.

Lunar Lander

L'agente per questo Gym, grazie alla stabilità fornita dall'uso di due reti, riesce ad imparare velocemente ad atterrare correttamente nel punto obiettivo, infatti in pochi episodi già è possibile osservarlo atterrare sano e salvo. Per favorire l'esplorazione degli stati, essendo infiniti anche in questo Gym, questo agente è stato spinto ad usare azioni casuali per 300 episodi, questo gli ha permesso di conoscere meglio l'ambiente in cui si muove.

Riferimenti e link utili

- Repository github con il codice dei gym risolti
- https://github.com/gmgigi96/reinforcement-learning
- Lezioni di Reinforcement Learning - David Silver
  https://www.youtube.com/watch?v=2pWv7GOvuf0
- *Playing Atari with Deep Reinforcement Learning* (Mnih, Kavukcuoglu, Silver, Graves, Antonoglou, Wierstra, Riedmiller)
- Tutorial sull'utilizzo di OpenAi Gym
  https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/
- https://openai.com/
- https://keras.io/