Reinforcement Learning in Lunar Lander

Botta Michele 534058
Github: `https://github.com/bottamichele/RLLunarLander`

Sommario. Nel seguente report viene discusso l'uso del Reinforment Learning su Lunar Lander e dei relativi risultati ottenuti con tale approccio.

1 Introduzione

Il **Reinforcement Learning** è uno dei tre paradigmi del Machine Learning e, a differenza degli altri due paradigmi, pone come obiettivo di addestrare un **agente** su un determinato **ambiente** (o **enviroment**) compiendo **azioni** e ottenendo delle **ricompense** (o **reward**) da tali azioni con lo scopo di imparare una determinata strategia che decide quale azione ritenuta adatta a compiere e questa strategia viene chiamata **policy**.

Nel presente report, il caso di studio preso in esame per l'uso del reinforcement learning, è stato scelto di usare Lunar Lander. **Lunar Lander** è un ambientazione del suolo lunare con la presenza di una navicella spaziale e lo scopo è di far atterrare la navicella sul suolo nel modo più sicuro possibile.

Il seguente report è organizzato come segue. La sezione 2 viene riportato le tecnologie utilizzate per il progetto. La sezione 3 viene descritto l'algoritmo di apprendimento utilizzato. La sezione 4 viene mostrato i risultati ottenuti dal modello addestrato.

2 Tecnologie utilizzate

Il progetto è stato sviluppato in Python utilizzando le seguenti librerie: Gymnasium e PyTorch.

Gymnasium è una libreria dedicata per il Reinforcement Learning che mette a disposizione diversi enviroment pronti utilizzati comunemente e tra quelli che possiamo trovare sono i giochi atari e Lunar Lander, che è oggetto del seguente report.

PyTorch è una libreria di Machine Learning e Deep Learning che permette di addestrare modelli in vari task con l'uso di reti neurali e da la possibiltà anche che i modelli possono essere addestrati su GPU in modo da accelerare il processo di addestramento. Il modello implementato è stato una rete neurale addestrata su GPU in modo che i tempi di addestramento non risultassero troppo lunghi.

3 Deep Q-Networks

Prima di descrivere l'algoritmo di apprendimento utilizzato, è utile introdurre qualche nozione.

In generale, un task del Reinforcement Learning è composto da un enviroment con un insieme di azioni possibili da compiere A, da un insieme di osservazioni S e da ricompense e da un agente che può compiere azioni e ricevere osservazioni e ricompsense da tali azioni. Un'**osservazione** s_i definisce lo stato corrente dell'ambiente, dell'agente o di entrambi. Un'azione x_i definisce un determinato comportamento dell'agente che la compie.

Ora passiamo alla descrizione dell'algoritmo di apprendimento utilizzato per il progetto.

Deep Q-Networks [[1]](o **DQN**) è una rete neurale convoluzionara che prende in input un'osservazione che è composta da un'immagine e da in output un valore per ogni azione che indica un grado di ottimalità di quell'azione. L'architettura di una DQN è costituita: da l'input layer, seguita dai strati nascosti composta da 3 layer convolutivi e da un layer fully connected, ed infine l'output layer. Inoltre, utilizza la funzione di attivazione ReLu. La rete DQN viene addestrata minimizzando la seguente funzione di loss:

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}[(y_i - Q(s,a;\theta_i))^2]$$

dove θ_i rappresenta i parametri della rete, la funzione Q rappresenta il grado di valutazione per la coppia corrente s e a e

$$y_i = \mathbb{E}_{s'\sim\varepsilon}[r + \gamma \max Q^*(s',a')|s,a]$$

dove γ rappresenta il **discount factor**.

Viene utilizzato anche un tecnica chiamata **experience replay** che tiene traccia di ciò che l'agente ha fatto e questo si traduce a create un **memory buffer** che memorizza le ultime N esperienze. Un'**esperienza** e_i è composta come (s_i, a_i, r_i, s_{i+1}). Inoltre, viene utilizzato anche la tecnica **ϵ-greedy policy** che consiste nel scegliere un'azione ottimale o un'azione casuale da compiere.

La versione del DQN utilizzata nel progetto è stato lasciato inalterato il funzionamento ed è stato modificato solo la sua architettura per adattarlo al task in esame. L'architettura della rete neurale che è stata modificata è stata la seguente: l'input layer avente numero di nomi pari alla dimensione dell'osservazione, seguita da due strati nascosti di fully connected con ognuno composto da 256 nodi e, infine, l'output layer con numero di nodi pari al numero di azioni possibili da compiere del task; inoltre, è stato utilizzato la funzione di attivazione ReLu eccetto tra il secondo strato nascosto e l'output dove viene eseguita una semplice linearizzazione.

Il funzionamento dell'algoritmo di apprendimento è sintentizzato in Algorithm 1.

Algorithm 1 Schema dell'algoritmo

Inizializza il memory buffer D di dimensioni N
Inizializza i parametri θ della rete neurale
$t = 1$
for episodio $= 1, M$ **do**
    while **do**episodio corrente non è terminato
        Con probabilità ϵ scegli l'azione a_t in modo casuale, altrimenti scegli $a_t = \max_a Q^*(s,a;\theta)$
        L'azione a_t viene compiuta e viene restituita la ricompensa r_t e l'osservazione s_{t+1}
        Memorizza (s_t, a_t, r_t, s_{t+1}) in D
        Costruisci un minibatch estrando casualmente esperienze da D
        $y_j = \begin{cases} r_j & \text{se } s_{j+1} \text{ terminale} \\ r_j + \gamma \max Q^*(s',a') & \text{altrimenti} \end{cases}$
        Esegue la discesa del gradiente $y_j - Q(s,a;\theta)$ sul minibatch costruito
        $t \leftarrow t + 1$
    end while
end for

4 Risultati

Il modello è stato addestrato utilizzato i seguenti iperparametri: dimensione del minibatch pari a 64, dimensione del memory buffer pari a 1000000, $\gamma = 0.99$, numero di episodi pari a 600, learning rate pari a 0.001 e ϵ iniziale pari a 1 che diminusce con l'avanzamento dell'addestramento fino al valore minimo pari a 0.1. Inoltre, l'ottimizzatore utilizzato per l'addestramento del modello è stato Adam.

Il modello addestrato è stato testato su un numero di episodi pari a 100 mostrando ottimi risultati totalizzando un punteggio medio pari a 253.48 con un deviazione stardard pari a 44.12. Questo significa che l'agente ha imparato bene di eseguire lo scopo del task preso in esame.

5 Conclusioni

L'uso del Deep Q-Networks per addestrare un agente per Lunar Lander è stato ottimo fornendo ottimi risultati.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing atari with deep reinforcement learning". In: **arXiv preprint arXiv:1312.5602** (2013).