OTTIMIZZAZIONE DEL PUNTEGGIO CON DDQN IN ATARI BREAKOUT

Giovanni Scorziello
Email: g.scorziello5@studenti.unisa.it

Abstract. Questo report descrive la realizzazione di un agente basato su Dueling Deep Q-Learning per il videogioco Atari Breakout, spiegando perché un approccio Q-Learning tradizionale non sia praticabile a causa delle dimensioni elevate dello spazio degli stati e della natura ritardata delle ricompense. Vengono presentati i passaggi chiave dell'implementazione: la riduzione dei frame (pre-processing), l'utilizzo di una Dueling Deep Q-Network, l'impiego della replay memory per ridurre la correlazione temporale, e il soft update dei pesi per stabilizzare l'apprendimento. Inoltre, si illustrano le scelte relative agli iperparametri e le motivazioni per la loro selezione, concludendo che tali tecniche permettono all'agente di apprendere strategie efficaci in un ambiente di gioco complesso.

1 INTRODUZIONE

In questo report viene descritta l'analisi e l'implementazione di un agente intelligente progettato per giocare al videogioco arcade *Atari Breakout*. Il progetto si basa sull'utilizzo di tecniche di Reinforcement Learning (RL), con un focus particolare sull'algoritmo di *Deep Q-Learning*.

L'obiettivo principale è stato quello di sviluppare un agente capace di apprendere autonomamente strategie efficaci per massimizzare il punteggio nel gioco, attraverso l'interazione continua con l'ambiente e l'ottimizzazione delle decisioni in tempo reale. Il report illustra le scelte metodologiche adottate, le fasi di sviluppo, i risultati ottenuti e le considerazioni finali sull'efficacia dell'agente nell'ambito del gioco.

2 OBBIETTIVI DEL PROGETTO

Gli obiettivi del progetto riguardano l'implementazione di un sistema avanzato di apprendimento per rinforzo in grado di giocare a **Atari Breakout**, massimizzando il punteggio e definendo azioni precise per muovere il paddle e anticipare i movimenti della pallina.
Il gioco termina quando vengono perse tutte le vite (una vita si perde se la pallina non viene intercettata dal paddle ed esce dallo schermo), per cui l'obiettivo primario consiste nell'insegnare all'agente a colpire regolarmente la pallina, imparandone la traiettoria e adottando strategie efficaci per abbattere i blocchi.

3 METODOLOGIE

In questa sezione vengono illustrati i passaggi fondamentali e i punti chiave che hanno contribuito al raggiungimento degli obiettivi prefissati.

3.1 Il Gioco

La prima fase relativa allo sviluppo dell'agente ha riguardato la comprensione del gioco e del suo funzionamento.

Lo scopo del gioco è semplice, muovere un paddle per riuscire a colpire una palla che a sua volta dovrà rompere un muro di mattoni presente nella parte superiore dello schermo.

3.2 L'Ambiente

Per l'implementazione dell'Ambiente è stata utilizzata la libreria **Gymnasyum**. Essa mette a disposizione un ambiente predefinito per Atari Breakout, contenente tutte le strutture necessarie per la gestione delle dinamiche di gioco e delle azioni disponibili, in particolare per l'implementazione dell'agente si è deciso di utilizzare l'ambiente *Breakout-v4*.

3.2.1 Azioni

Lo spazio delle azioni definito all'intero dell'ambiente utilizzato è di 4 azioni, definite come segue:

0 (NOOP): nessuna operazione

1 (FIRE): esplosione di un blocco

2 (RIGHT): muovere il paddle verso sinistra

3 (LEFT): muovere il paddle verso destra

3.2.2 Spazio delle osservazioni

Lo spazio delle osservazioni dell'ambiente non è altro che l'immagine **RGB** che viene mostrata al giocatore umano, l'ambiente restituisce quindi i frame di gioco. Sul frame restituito sono state effettuate delle operazioni di preprocessing per ridurre la complessità dei frame da dare in pasto al modello.

3.2.3 Reward

Di default, l'ambiente assegna una reward ogni volta che viene distrutto un blocco del muro di mattoni, con un valore variabile in base al colore del blocco. Tuttavia, per ottimizzare il processo di addestramento del modello, questi meccanismi di reward sono stati riadattati.

In particolare, è stata introdotta una penalità pari a *-1* ogni volta che viene rilevata la perdita di una vita, al fine di scoraggiare comportamenti che aumentano il rischio di fallimento.

Inoltre, la reward per la distruzione di un blocco è stata uniformata a un valore fisso di +1, indipendentemente dal colore del blocco. Le motivazioni alla base di queste scelte verranno approfondite nel paragrafo dedicato al Pre-Processing.

3.2.4 Fine Episodio

L'episodio termina nel momento in cui vengono perse tutte le 5 vite di gioco.

3.3 Sviluppo dell'Agente

Partendo dall'analisi della documentazione fornita da *Gymnasyum* per l'ambiente, l'utilizzo di un algoritmo di **Q-Learning** standard è sembrato improbabile fin dall'inizio per diversi motivi.

3.3.1 Spazio delle Osservazioni

Dalla documentazione di Gymnasium risulta che lo spazio delle osservazioni di Atari Breakout è un `array` di forma (210, 160, 3), dove 210 è il numero di pixel in altezza, 160 il numero di pixel in larghezza e 3 rappresenta i tre canali di colore (rosso, verde e blu).
Di conseguenza, ogni stato dell'ambiente è un singolo fotogramma che può essere visto come un `array` tridimensionale in cui ciascun pixel presenta 256 possibili valori per canale, generando uno spazio di stati estremamente grande $256^{210\times160\times3}$). Ciò rende impraticabile l'utilizzo di un *Q-Learning* tabellare, in quanto sarebbe impossibile gestire o memorizzare una Q-table che copra tutti i possibili stati derivanti dai diversi fotogrammi del gioco, rendendo pertanto necessario ricorrere a metodologie più avanzate come il **Deep Q-Learning**.

3.3.2 Ricompensa Ritardata

A ciò si aggiunge il fatto che, in questo ambiente, la ricompensa è ritardata rispetto all'azione che l'ha effettivamente generata: la ricompensa viene assegnata ogni volta che un blocco viene distrutto, ma l'azione decisiva (spostare il paddle per colpire la pallina) è avvenuta diversi frame prima. Di conseguenza, ricompense e penalità non vengono fornite a ogni frame, bensì in ritardo, rendendo più complesso correlare in modo diretto azioni e ricompense. Le reti neurali profonde consentono di apprendere feature che permettono di cogliere relazioni di lungo periodo e associare azioni distanti alle relative ricompense, cosa molto più complicata da gestire con un approccio di Q-Learning classico.

La soluzione identificata è stata quindi quella di utilizzare un'algoritmo di Deep Q-Learning, una variante del Q-Learning che si basa sull'utilizzo di reti neurali per approssimare la funzione Q.
Tra le varie tecniche, quella identificata come migliore, e quindi quella selezionata, è il **Dueling Deep Q-Network (DDQN)**. Questa architettura permette di separare esplicitamente la rappresentazione della *value function* di uno stato dalla *advantage function* di un'azione, così che il modello possa apprendere quali sono gli stati con un valore maggiore senza dover per forza apprendere l'effetto di ogni azione per ogni stato.
Nel paragrafo dedicato alla soluzione proposta saranno forniti maggiori dettagli sul perché questa sia una funzione fondamentale per l'apprendimento nel gioco Atari Breakout.

3.4 Pre-Processing

Per aumentare l'efficienza delle operazioni di apprendimento, sono state apportate modifiche all'ambiente Atari Breakout di Gymnasium e sono state effettuate operazioni di pre-processing sui frame restituiti: la funzione `step` è stata regolata in modo che l'agente compia un'azione ogni 4 frame (frame skipping), poiché non è necessario reagire a ogni singolo frame, dato che l'unica differenza tra due frame consecutivi è lo spostamento di qualche pixel della pallina. Inoltre, per ridurre la complessità dei dati da fornire alla rete, i frame sono stati ridimensionati da 210×160 a 84×84 pixel e convertiti da immagini RGB (tre canali) a immagini in scala di grigi (un solo canale). Di conseguenza, non essendo più in grado di distinguere i colori, la ricompensa per la distruzione di ogni blocco è stata uniformata a 1.

4 SOLUZIONE PROPOSTA

4.1 Dueling Deep Q-Learning

L'utilizzo di una **Dueling Deep Q-Network** si è rivelato la soluzione ottimale per conseguire gli obiettivi prefissati, grazie alle caratteristiche peculiari di tale architettura: essa scorpora la stima del valore di uno stato (*value function*) da quella del vantaggio (*advantage*) di intraprendere una determinata azione. In questo modo, la rete neurale suddivide il suo ultimo strato in due "rami": uno dedicato alla stima della *value function* dello stato S e l'altro all'*advantage function* per ciascuna azione; i due output vengono poi ricombinati, ottenendo la stima finale di Q. A differenza di una DQN classica, dove invece Q viene approssimata direttamente, questo approccio permette di imparare quanto sia buono uno stato anche quando le azioni disponibili hanno effetti molto simili tra loro, come avviene spesso in Atari Breakout, gran parte dei reward sono infatti pari a 0, poiché talvolta il movimento della pallina non produce eventi significativi. Di conseguenza, la capacità di distinguere il valore dello stato dal vantaggio delle azioni accelera l'apprendimento e migliora la stabilità dei risultati.

4.2 Replay Memory

Per l'addestramento della rete è stata utilizzata una **Replay Memory**, nella quale vengono memorizzate tutte le esperienze dell'agente a ogni step per tutti gli episodi.
In altre parole, vi sono raccolte le transizioni (stato, azione, ricompensa, ecc.) generate durante l'interazione con l'ambiente da parte dell'agente. Tale buffer ha un limite di 1.000.000 elementi; da questo insieme di "esperienze" si estraggono in modo casuale i campioni necessari per addestrare la rete. Questa strategia risulta essenziale, in quanto contribuisce a ridurre la correlazione temporale dei dati.
La rete non viene così addestrata su sequenze consecutive di frame, ma su esperienze passate prese in modo random, migliorando l'efficacia dell'apprendimento. Tali caratteristiche si rivelano fondamentali per l'addestramento di una *DDQN* in un ambiente complesso come `Atari Breakout`.

4.3 Architettura dell'Agente

L'architettura del modello è composta da tre strati di *Rete Neurale Convoluzionale*, responsabili dell'estrazione delle feature dai frame in ingresso.
Essi inizialmente presentano un solo canale, in quanto in scala di grigi, e arrivano a 64 canali nell'ultimo livello convoluzionale. Il risultato di queste convoluzioni viene poi appiattito e inoltrato a un livello completamente connesso, suddiviso in due rami: uno dedicato alla predizione di un singolo valore V, che indica quanto sia buono lo stato, e l'altro impegnato a calcolare il vettore di vantaggi A (uno per ciascuna azione). Entrambe le sezioni contengono tre strati lineari: nella parte che stima il valore di stato l'uscita finale è un singolo valore, mentre nella sezione dedicata alle azioni l'uscita finale fornisce un numero di valori pari al numero di azioni possibili. Infine, per combinare il valore di stato e il vantaggio in un'unica stima di Q, si sfrutta la formula:

$$Q(s,a;\theta,\alpha,\beta) = V(s,\theta,\beta) + (A(s,a,\theta,\alpha)) - \frac{1}{|A|}\sum_{a'} A(s,a,\theta,\alpha))$$

che garantisce una separazione esplicita tra la "bontà" intrinseca dello stato e la differenza di valore tra le varie azioni, e risulta particolarmente utile quando più azioni forniscono reward simili, riducendo l'indeterminazione e accelerando l'apprendimento.

4.4 Soft Update

Per l'aggiornamento dei pesi durante l'addestramento è stata utilizzata la tecnica del **Soft Update**, che permette di mantenere allineato il *target network* con la rete principale in modo graduale, evitando di sostituirne completamente i pesi a intervalli regolari. Questo approccio rende più affidabile la propagazione di quanto appreso dalla rete, riducendo le oscillazioni dei valori Q.
A ogni passo di training, i pesi del target network vengono modificati come combinazione tra i pesi attuali e quelli della rete principale, secondo un coefficiente τ (tau) che bilancia il peso dell'aggiornamento.
Questo coefficiente ha un ruolo chiave, in quanto controlla quanto velocemente i pesi della rete principale influenzano i pesi della target network.
Un τ di valore piccolo, come 0.001 in questo caso, rende il processo molto graduale e stabilizza l'apprendimento; se invece τ fosse più grande, la rete target si allineerebbe più velocemente ai pesi della rete principale, con il rischio di introdurre maggiore instabilità.

4.5 Addestramento dell'Agente

Durante la fase di addestramento dell'agente sono state testate diverse combinazioni di iperparametri per ottimizzare le performance; i valori che hanno prodotto i risultati migliori sono i seguenti:

Batch Size: impostato a 64

Learning Rate: impostato a $1 \times 10^{-}4$

Epsilon: Utilizzata per regolare il tasso di esplorazione del modello durante l'addestramento, indica la probabilità con la quale il modello sceglierà un'azione casuale piuttosto che un'azione ritenuta ottimale. Essa è stata inizialmente impostata a 1.0. per essere poi gradualmente ridotta utilizzando il *simulated annealing*. Ciò ha permesso all'agente di esplorare durante la fase iniziale dell'addestramento, per poi stabilizzare le proprie decisioni verso la fine.

Warmup Steps: Indica un numero di steps iniziale in cui l'agente interagisce con l'ambiente senza aggiornare la rete, così da popolare la replay memory con un campione di transizioni sufficientemente vario prima di iniziare l'apprendimento vero e proprio. Il numero di step è stato impostato a 10.000, per via dell'elevata dimensione dello spazio delle osservazioni dell'ambiente.

Minimum Epsilon (ε_{min}): Fissata a 0.05, rappresenta il valore minimo raggiungibile da ε (epsilon) durante l'addestramento.

Epsilon Decay: Il tasso di decadimento di epsilon è stato calcolato mediante la formula

$$\frac{(\varepsilon - \varepsilon_{min})}{warmupsteps}$$

Gamma: Impostata a 0.99, indica il fattore di sconto per le future ricompense. Con un fattore impostasto molto vicino ad 1, attribuiamo una maggiore importanza alle ricompense future.

Memory Size: La calacità massima della *Replay Memory* utilizzata dall'agente, impostata ad un valore di 1

milione.

Come ottimizzatore è stato adottato **AdamW**. In un ambiente così vasto, il numero di episodi per l'addestramento si è rivelato cruciale: un primo ciclo di 2.000 episodi portava a un punteggio massimo di circa 5, evidenziando l'incapacità dell'agente di comprendere pienamente l'ambiente di gioco. Un secondo ciclo di 5.000 episodi ha mostrato i primi miglioramenti, con punteggi attorno a 10. Dopo un *fine tuning* di altre 2.000 epoche, abbassando il learning rate a 1×10^5e fissando ε a 0.3, si è notato un leggero miglioramento, ma è stato l'incremento a 20.000 episodi a consentire all'agente di raggiungere punteggi superiori a 15. Gli iperparametri finali qui descritti hanno offerto i risultati più soddisfacenti, anche in considerazione delle risorse computazionali limitate rispetto a quelle impiegate nei progetti di riferimento; in tale contesto, le prestazioni ottenute rappresentano un buon compromesso tra la soluzione ottimale e le risorse disponibili.

4.6 I Risultati Ottenuti

I migliori modelli ottenuti dalle diverse fasi di addestramento effettuate sono 2:

Breakout-Player-v1.0: Si tratta dell'agente sul quale è stato effettuato prima un training su 5000 epoche con gli iperparametri definiti precedentemente, e successivamente altre 2000 epoche di **fine tuning** abbassando la learning rate a 1×10^5, ε a 0.5 e ε_{min} a 0.01

Breakout-Player-v2.0: L'agente sul quale è stato effettuato un training di 20000 epoche con i parametri descritti bel paragrafo relativo all'Addestramento dell'Agente.

La valutazione delle performance degli agenti risultanti è stata effettuata tramite una fase di testing, nella quale ogni agente ha effettuato un totale di 50 partite, durtante le quali si è tenuto traccia del punteggio raggiunto dagli agenti per ogni partita.
Questi dati sono stati successivamente utilizzati per verificare il **punteggio massimo** ottenuto dall'agente durante le partite, in modo da verificare quale fosse il miglior risultato possibile raggiungibile dall'agente durante una partita, il **punteggio medio** e la **varianza** tra i punteggi tra le varie partite, che servono invece per verificare se l'agente è effettivamente riuscito ad imparare a giocare abbastanza bene da raggiungere un risultato soddisfacente per quanto riguarda il punteggio in tutte le partite effettuate, così da verificare anche se un alto punteggio massimo fosse solo un caso isolato o fosse effettivamente merito dell'apprendimento effettuato. La valutazione delle performance degli agenti risultanti è stata effettuata tramite una fase di testing. Questa fase prevedeva che ogni agente giocasse 50 partite, durante le quali è stato registrato il punteggio di ciascuna sessione. Successivamente, per valutare le prestazioni, sono stati analizzati il **punteggio massimo**, il **punteggio medio** e la **varianza** dei punteggi ottenuti. Il **punteggio massimo** indica il miglior risultato raggiungibile in una singola partita, evidenziando il potenziale massimo dell'agente; il **punteggio medio** descrive la prestazione tipica dell'agente, consentendo di capire quanto regolarmente l'agente ottenga buoni risultati; infine, la **varianza** offre indicazioni sulla stabilità delle performance, rivelando se l'agente è in grado di mantenere risultati costanti o se, invece, dipende da episodi isolati. I risultati ottenuti sono i seguenti:

4.6.1 Breakout-Player-v1.0

Punteggio Massimo: 9

Punteggio Medio: 6.6

Varianza: 6.1

4.6.2 Breakout-Player-v2.0

Punteggio Massimo: 18

Punteggio Medio: 8.6

Varianza: 6.6

Dai risultati emerge che l'agente **Breakout-Player-v2.0** riesce a raggiungere un punteggio massimo maggiore (18 rispetto a 9) e mantiene anche un punteggio medio più elevato (8.6 contro 6.6), indicando un miglioramento complessivo delle prestazioni rispetto alla versione precedente.
La varianza leggermente superiore (6.6 contro 6.1) suggerisce che le performance del modello potrebbero presentare una distribuzione dei punteggi leggermente più ampia, ma, considerando l'aumento significativo del punteggio medio e di quello massimo, ciò indica comunque una capacità complessivamente più elevata di ottenere punteggi alti, anche se con una certa variabilità tra le partite.

References

[1] Github Repo: ddqn-atari-breakout