Cydon-IA

Battaglie Pokémon 1 vs. 1 alla portata di tutti

Docente

Fabio Palomba

Studenti

Leonardo Schiavo e Emmanuel De Luca

Marzo 2024

Università degli Studi di Salerno

Documentazione del progetto sviluppato per il corso di "Fondamenti di Intelligenza Artificiale"

https://github.com/KLS-01/Cydon-IA

Indice

1 Introduzione

Pokémon è un franchise giapponese di proprietà di **The Pokémon Company** creato nel 1996 da Satoshi Tajiri. La sua nascita coincide con la pubblicazione di due videogiochi: *Pokémon Versione Rossa* e *Pokémon Versione Verde*.

Ricordate tutte le battaglie estenuanti passate piegati sul proprio GameBoy?
Abbiamo deciso di ovviare al superamento di questa difficoltà proponendo un modello di intelligenza artificiale che aiuta il giocatore consigliando la migliore mossa possibile in battaglie Pokémon 1 vs. 1 di 1ª generazione.

2 Definizione del problema

2.1 Obiettivi

L'obiettivo preposto da *Cydon-IA* è di aiutare il giocatore a vincere una battaglia Pokémon 1 vs. 1 suggerendo la mossa più opportuna.

2.2 Specifica P.E.A.S.

Qui riportiamo la nostra specifica P.E.A.S.:

- **Performance**: le misure di prestazione scelte sono le seguenti:
  - **Tasso di vittoria**: Questa misura indica la percentuale di battaglie vinte dall'agente rispetto al totale delle battaglie affrontate.
  - **Efficacia delle mosse suggerite**: Questa misura indica quanto le mosse suggerite dall'agente portano a risultati positivi.
  - **Adattabilità**: Questa misura indica quanto bene l'agente si comporta in scenari variabili.
- **Environment**: l'ambiente in cui opera l'agente è rappresentato dallo stato attuale della battaglia (nome dei Pokémon in campo, salute, status, malus).
- **Actuators**: l'agente interviene nell'ambiente suggerendo la mossa più conveniente per vincere la battaglia.
- **Sensors**: la percezione dell'ambiente da parte dell'agente è data da un insieme di dati che costituiscono una rappresentazione dello stato corrente della battaglia.

2.2.1 Caratteristiche dell'ambiente

- **Singolo agente**: la singola AI a cui sarà affidato il compito di suggerire le mosse al giocatore.
- **Completamente osservabile**: l'agente ha accesso a tutte le informazioni riguardanti la battaglia. Può vedere lo stato attuale dei Pokémon in campo, la loro salute, gli stati alterati e altre informazioni rilevanti.
- **Stocastico e simil black box**: l'ambiente è influenzato da fattori non completamente prevedibili, quali le mosse degli avversari, che potrebbero variare anche in base a fattori casuali.
- **Sequenziale**: La battaglia procede in turni, con l'agente e l'avversario che si alternano nell'eseguire le mosse.
- **Statico**: l'ambiente non cambia mentre l'agente sta operando all'interno della battaglia. Tuttavia, le mosse degli avversari possono variare in base alle decisioni dell'agente stesso.
- **Discreto**: l'ambiente ha un numero finito di stati distinti (seppur elevato). Le azioni possibili includono le mosse dei Pokémon, che sono anch'esse discrete.

2.3 Analisi del problema

Dato l'obiettivo di sviluppare una IA in grado di assistere il giocatore nel prendere decisioni strategiche durante battaglie Pokémon 1 vs. 1, esaminiamo i concetti chiave correlati al problema:

Battaglie Pokémon 1 vs. 1: Le battaglie sono combattimenti tra due Pokémon, uno per parte. L'obiettivo è sconfiggere il Pokémon avversario riducendo la sua salute a zero.

Mosse dei Pokémon: Ogni Pokémon ha un insieme di mosse che può utilizzare durante la battaglia. Le mosse hanno diversi effetti, come infliggere danno, alterare gli stati o fornire benefici.

Strategia e decisioni: Il giocatore deve scegliere attentamente le mosse da utilizzare in base alla situazione. La strategia coinvolge la conoscenza dei tipi di Pokémon, delle loro statistiche e delle mosse disponibili.

Completamente osservabile: L'IA deve avere accesso a tutte le informazioni sulla battaglia, inclusi i dettagli dei Pokémon, la loro salute, gli stati alterati e le mosse disponibili.

Stocastico e simil black box: L'ambiente di battaglia è influenzato da fattori casuali e decisioni nascoste dell'avversario. L'IA non ha piena conoscenza di come l'avversario prenderà decisioni.

3 Soluzione al problema

Il primo passo da seguire per capire come risolvere il problema scelto è capire e comprendere quale tipologia di IA, tra le varie viste, possa essere correttamente utilizzata per dare una soluzione. Il problema proposto offre un ambiente complesso e che non permette di avere una soluzione fissa o una risposta che sia totalmente corretta. Quindi, a monte delle osservazioni più specifiche, il primo approccio a cui potremmo pensare sarebbe quello di usarne uno di Reinforcement Learning, ma osserviamo in dettaglio anche altri.

Esplorandoi vari approcci, usandone uno non informato ad esempio, come la ricerca in profondità, lo spazio degli stati risulterebbe così immenso che non sarebbe completo e nemmeno ottimo, la complessità temporale resta esponenziale e così anche quella spaziale, infine, l'ambiente così complesso non darebbe modo di avere una strategia ottimale. Usando la ricerca informata; invece, la definizione di una giusta euristica potrebbe essere molto complesso, e sarebbe ostacolata dall'ampia dinamicità del turno stesso. Un approccio A*, ad esempio, sarebbe ostacolato dal grande numero di stati da rappresentare, che genererebbe problemi di spazio e di esplorazione, rendendo la computazione infinita.

Un approccio basato sulla teoria dei giochi troverebbe difficoltà nel trovare una strategia razionale, dato che l'approccio stesso ha bisogno di una comprensione dettagliata delle strategie avversarie, e ha anche bisogno di partire dall'assunzione che entrambi gli attuatori siano razionali, cosa che nel contesto del problema scelto potrebbe non essere sempre verificato. Inoltre, un approccio del genere non permetterebbe all'agente di apprendere, facendo in modo tale da pregiudicare un adattamento stesso dell'IA a diverse strategie di gioco.

Considerando invece un approccio basato sugli algoritmi genetici, il concetto stesso del problema esplorato non si limita al voler trovare una situazione iniziale ed ottimizzarla, ma trovare soluzioni adattive, efficaci, a tempo reale e nate dall'apprendimento.

Proprio per i vari motivi elencati, un approccio basato sul Reinforcement Learning, che fa utilizzo del Q-Learning, rappresenta l'approccio migliore, che permette all'AI di imparare strategie possibili, esplorarle e migliorarle, e non solo di dare quella che, dopo una sola computazione, sembra essere quella migliore. In conclusione, in un ambiente così complesso, è ottimo usare un approccio basato sull'apprendimento e sullo sviluppo di strategie adattive, interagendo direttamente con l'ambiente stesso, e quindi usare un approccio per Rinforzo, più in particolare usare il Q-Learning risulta essere comunque l'approccio migliore.

3.1 Algoritmo di Q-Learning

Di seguito sono riportati i vari passi caratteristici dell'**algoritmo di Q-Learning**:

- Crea una matrice Q_0, ossia composta da tutti zero.
- Sceglie uno stato casuale dallo spazio degli stati in caso di apprendimento batch, oppure usa lo stato corrente passato al modello in caso di apprendimento online.
- Sceglie un'azione casuale tra quelle disponibili nell'ambiente.
- Esegue l'azione, osserva l'evoluzione dell'ambiente in base ad essa e ne calcola il reward associato.

- Normalizza il valore usando la formula tipica del Q-learning:
  $Q(s_t, a_t) \leftarrow Q(s_t, a_t)(1 - \alpha_t(s_t, a_t)) + \alpha_t(s_t, a_t)[R_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})]$
- Inserisce il reward nella matrice Q, composta da N righe ed M colonne, nella posizione $[i][j]$, dove sulle righe vengono rappresentati gli stati e sulle colonne vengono rappresentate le azioni e i rappresenta la i-esima riga e la j rappresenta la j-esima colonna.
- Continua l'addestramento sullo stato successivo.
- La best move sarà il massimo per la colonna e la riga all'interno della matrice Q.

4 Implementazione del sistema

Compreso l'algoritmo e motivata la scelta dietro all'approccio per Rinforzo, compresa la strategia da attuare, è possibile ora individuare un'implementazione dell'algoritmo e integrarlo con il sistema scelto. L'algoritmo è rappresentato da 4 fasi:

- Estrazione delle informazioni dalla battaglia
- Rappresentazione dello stato
- Esecuzione dell'algoritmo di Q-learning
- Mostrare a schermo il suggerimento dell'IA

4.1 Estrazione delle informazioni dalla battaglia

Prima di poter addestrare il modello sullo stato corrente dobbiamo fare raccolta delle informazioni relative allo stato corrente, questo è necessario per rappresentare correttamente lo stato come parte dello spazio degli stati del nostro ambiente, in questo modo possiamo eseguire correttamente l'algoritmo, inoltre, quante più informazioni riusciamo a raccogliere e rappresentare attraverso lo stato corrente, quanto più precise e coerenti saranno le mosse eseguite dall'IA e tanto saranno corrette le informazioni rappresentative dell'ambiente.

4.1.1 Informazioni relative alla battaglia

Il modello prende in input uno stato corrente, rappresentato da una tupla formata dalle informazioni raccolte nella fase di estrazione. Per quanto sia molto complesso un turno Pokémon, per motivi implementativi si è optato per lo scegliere le informazioni più importanti e rappresentative di una battaglia di questo genere.

4.2 Rappresentazione dello stato

Per la rappresentazione dello stato corrente è fondamentale dare informazioni sul Pokémon corrente, il Pokémon nemico, sulla vita, situazione di status, di boost di entrambi i Pokémon, e inoltre anche informazioni sulla possibilità al cambio del nostro Pokémon, in modo tale da dare una visione generale e quanto più accurata possibile dello stato.
Le informazioni vengono estrapolate dal file HTML della pagina dove viene hostata la battaglia; quindi, la cosa più congeniale possibile è stata creare un'estenzione per chrome, in modo tale da facilitare, tramite l'utilizzo di javascript, la lettura inamica della pagina stessa, in cui sono raccolte tutte le informazioni necessarie alla rappresentazione corretta stato per stato. Per collegare il modello e l'estensione si è optato per l'utilizzo di Flask, per hostare un server locale che fa la chiamata al modello inviando i dati dello stato corrente, e l'utilizzo di JavaScript, per la realizzazione dell'estensione e della chiamata asincrona. I dati vengono inviati come JSON, dato che viene utilizzata la REST API; quindi, è stato necessario convertire il JSON in data FLASK, e successivamente creare la tupla rappresentante dello stato corrente. Di seguito verrà riportato un esempio di composizione della tupla stessa.

```
('Charmander', 'Poliwrath', 1.5, 1.5, True, False, True, False, True)
```

4.3 Esecuzione dell'algoritmo di Q-learning

Adesso illustreremo come viene eseguito l'algoritmo di Q-learning.

4.3.1 Definire il vantaggio del turno

Per definire se un turno risulta vantaggioso o meno per il Pokémon attuale viene usata una funzione euristica che valuta la situazione attuale tramite una formula basata sui danni causati da noi e dai danni causati dal Pokémon avversario, per valutare se in quel turno il delta è favorevole a noi o meno, dove il delta viene rappresentato da un booleano e indica se in quel tSurno l'ambiente è favorevole o meno a noi.

Di seguito la formula per calcolare il delta del turno, indicato come *damage_advantage*:

$$damage\_advantage = damage\_dealt\_by\_me \geq damage\_dealt\_to\_me$$

dove:

- $damage\_dealt\_by\_me = \frac{(atk1+spatk1)\cdot first\_mon\_type\_advantage\cdot my\_buff\cdot opponent\_status}{de2\cdot hp2}$
- $damage\_dealt\_to\_me = \frac{(atk2+spatk2)\cdot second\_mon\_type\_advantage\cdot opponent\_buff\cdot my\_status}{de1\cdot hp1}$

A questo punto il ragionamento su quale sia la mossa migliore da fare viene abbastanza semplice, dato che un'azione aggressiva in un turno con delta negativo potrebbe compromettere la vittoria della partita. Tuttavia, ciò non è sempre vero, dato che l'algoritmo riesce a vedere oltre il singolo turno, grazie proprio al concetto di epoche introdotto dall'approccio scelto. Un turno sarà simulato per N epoche, e in queste N epoche saranno rappresentati tutti i risvolti che azioni casuali portano all'ambiente.
Quindi il risultato finale sarà dato dal massimo valore che la funzione di reward assume nelle varie epoche iterate su quel singolo stato.

4.3.2 Elaborazione della funzione di reward

Dopo aver utilizzato la funzione per trovare il delta del turno, sarà facile valutare quindi le azioni, in base al delta stesso. In particolare, se osserviamo il suggerimento di una mossa aggressiva allora il modello prima di valutare il reward stesso andrà a valutare se il delta risulta positivo, se osserviamo il suggerimento di una mossa difensiva si andrà a valutare il delta come negativo, successivamente dev'essere valutata l'azione compiuta, dando un valore arbitrario 1 nel caso in cui l'azione viene valutata coerente al delta, -1 nel caso di una mossa non coerente al delta e 100 nel caso di mossa che porta allo stato finale, ossia lo stato in cui la vita dell'avversario è 0.

4.4 Mostrare a schermo il suggerimento dell'IA

Il suggerimento, ritenuto dal modello ottimale per il dato stato, verrà poi passato al sistema chiamante tramite JSON risposta della chiamata REST effettuata all'applicazione FLASK tramite AJAX. L'output possibile del modello sarà quindi 0 - ATTACCA oppure 1 - DIFENDI.

4.5 Obiettivi

L'obiettivo principale risulta non ottenere sempre la mossa che mi dà vincita assicurata, ma la mossa, che nella propagazione delle epoche, mi dà reward complessivo maggiore, quindi la mossa che risulta massimo per quella posizione.

4.6 Valutazione

Una volta terminato lo sviluppo del modello si ha avuto modo di verificare l'efficacia e l'ottimizzazione temporale del modello, in particolare abbiamo testato il tempo di addestramento su 5000 epoche per diversi stati, testando le prestazioni del modello e i tempi di risposta.

4.6.1 Precisione

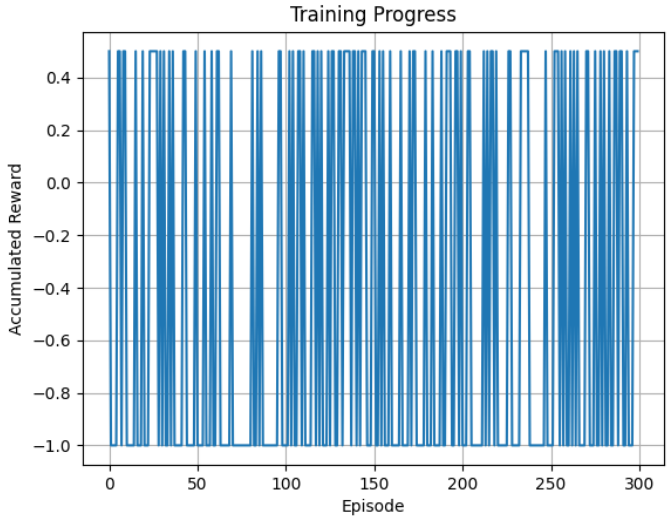
Per la valutazione del modello abbiamo usato due approcci:

- la valutazione del modello su situazioni in un ambiente reale, ossia una simulazione di una battaglia.
- la valutazione su dati in un dataset.

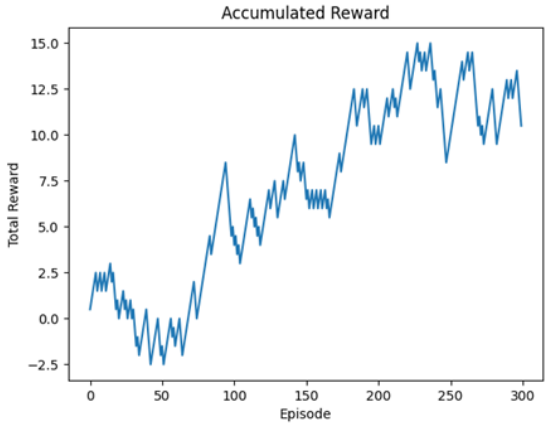
Per la valutazione del modello in N simulazioni di una battaglia reale Pokémon, abbiamo ottenuto risultati discordanti. Dato che il modello per alcune situazioni, che aveva già affrontato diverse volte nel passato, risultava essere molto pronto ed efficiente, invece, per situazioni sporadiche, di cui l'agente non aveva già delle epoche di addestramento ed esperienza, risultava suggerire mosse poco ottimali ed efficienti. Questo risultato mostra la difficoltà nel trovare dei pattern e delle strategie specifiche e uguali per questo tipo di ambiente così complesso.

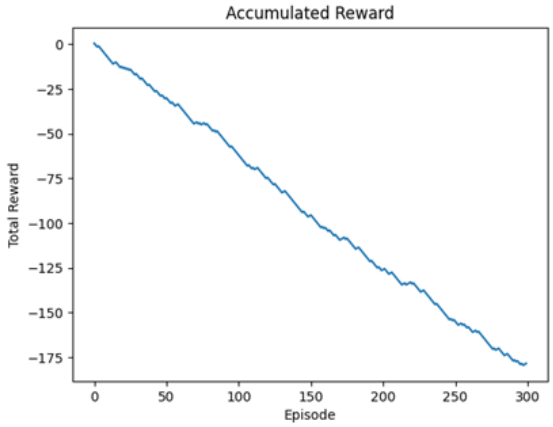
Tuttavia, i risultati non sono tutti da scartare, e molte volte il modello ha effettivamente suggerito mosse giuste, come visto anche dall'approccio con il dataset, volendo stimare in percentuale il 75% dei casi il modello stimava mosse giuste.

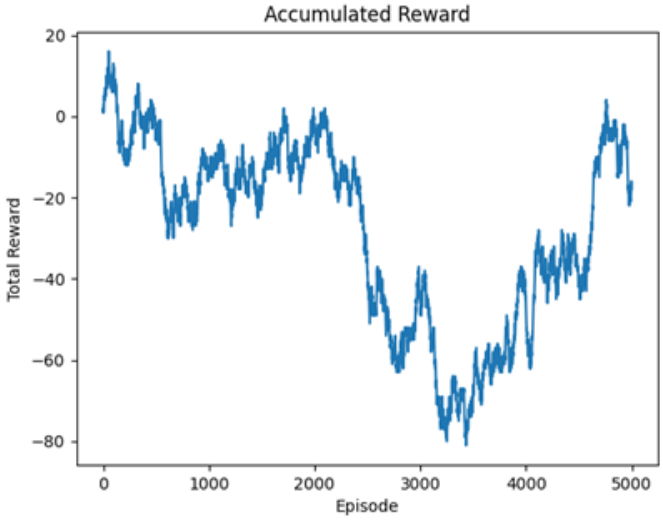
Inoltre, dal grafico delle distribuzioni delle ricompense si può proprio osservare la natura stocastica dell'ambiente. In generale, comunque, un grafico discontinuo nella curva di apprendimento può essere normale per diverse ragioni, una di queste è propria dell'apprendimento per rinforzo: l'apprendimento per rinforzo è un processo stocastico, il che significa che il comportamento dell'agente e le ricompense ottenute possono variare casualmente da un episodio all'altro.



Dopo N epoche le ricompense sono distribuite in modo più lineare, e in generale possiamo avere due tipi di grafici, uno per addestramento su stato favorevole e uno per addestramento su stato sfavorevole. Di seguito sono rappresentati a sinistra un grafico per apprendimento favorevole, a destra un grafico per situazioni sfavorevoli, in basso il grafico in situazione di parità.



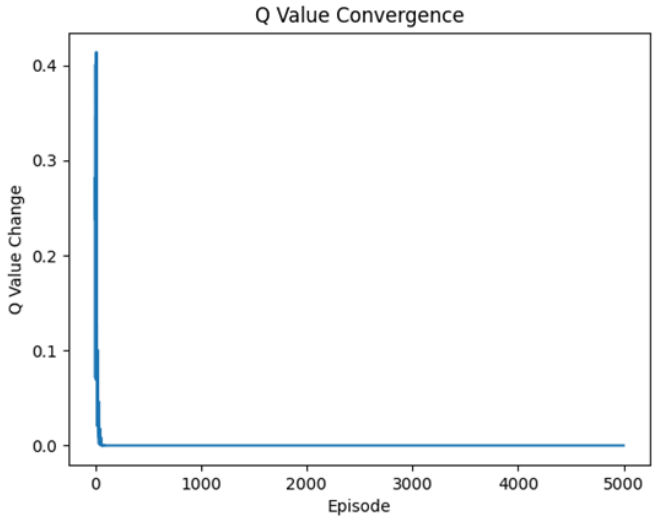


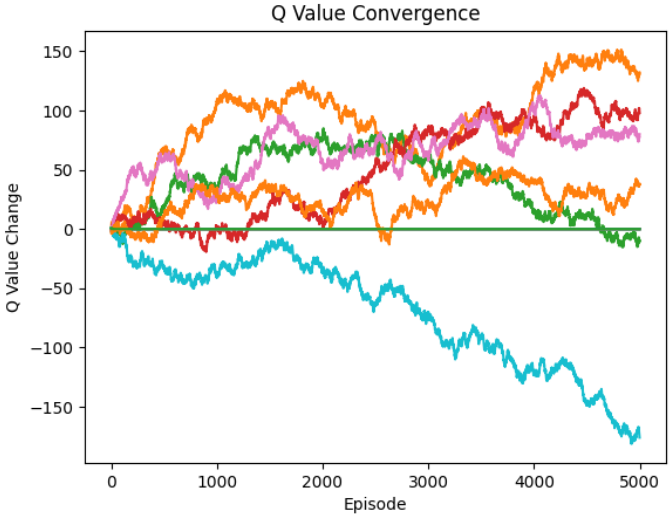


4.6.2 Convergenza dei valori

Facendo il grafico dei valori per verificarne la convergenza notiamo subito che la media aritmetica dei valori punto per punto cominciano a convergere quasi subito sullo zero, questo sta a rappresentare che il modello è arrivato alla saturazione delle mosse disponibili, solo due, data la poca diversità nelle mosse stesse, e che probabilmente la soluzione trovata per quel valore nello stato è già quella ottimale. Questo può essere presupposto perché le azioni essendo solo due, ed avendo entrambe valore di reward molto basso per scelta implementativa, dato che si è scelto di favorire l'azione che; dopo N epoche, avrebbe garantito il KO avversario.

Di seguito il grafico della convergenza dei valori.





4.6.3 Valutazione temporale del modello

Il modello, una volta implementato, è stato testato su una macchina di media/basso fascia e su una di medio/alta fascia. I risultati sono stati ottimali, dato che la macchina di bassa fascia era una macchina di vecchia generazione, avente come caratteristiche:

- Come OS: Windows 10
- Come processore: Intel(R) Core(TM) i3
- Come RAM installata: 8GB

Il modello è stato valutato eseguendolo nell'IDE PyCharm, di JetBrains, usando come browser web Chrome 109, e come simulatore delle battaglie il sito Pokémon showdown, e come collegamento tra le parti un'estensione scritta per chrome.

Il modello è stato utilizzato da persone aventi già conoscenze dell'ambiente delle battaglie Pokémon, ma aventi poca familiarità con il mondo competitivo di Pokémon e con il simulatore utilizzato, è stato quindi convenuto che il tempo di risposta e di decisione sono stati raccolti non tenendo conto del tempo impiegato dalle persone scelte per imparare a interagire con il simulatore.

I tempi di utilizzo del modello sono stati raccolti in una tabella e rappresentano i due esempi più interessanti, dato che mostrano il divario temporale tra chi già conosce il sistema e il simulatore e chi, invece, non ha molta familiarità con entrambi.

Persona	**Tempo impiegato (dall'utente)**
Persona con conoscenze 1	40 sec.
Persona senza conoscenze	1.5 min.

In particolare, possiamo osservare, che la persona con conoscenze ha impiegato ben 1,1 min in meno alla persona senza conoscenza per avere familiarità con l'integrazione del sistema, dato che probabilmente spaesato dalla prima interazione anche con il simulatore stesso.
Terminata la fase di valutazione dell'interazione tra utente e sistema, passiamo ora a rappresentare il tempo impiegato dall'algoritmo e dal sistema per suggerire la mossa.

Persona	**Tempo impiegato (dal sistema)**
Persona con conoscenze 1	50 sec.
Persona senza conoscenze	50 min.

In entrambi i casi il sistema è stato in grado di suggerire la mossa per lo stato corrente in meno di un minuto, esplorando con successo un numero arbitrario di epoche (5000) standard per entrambe le persone.

In una situazione normale, possiamo stabilire che per una decisione ottimale, la scelta della mossa corretta da eseguire può richiedere anche più di due minuti, dato che la pressione della battaglia e la scelta complicata della mossa insieme all'innumerevole numero di variabili da considerare può portare a protrarre il processo decisionale umano anche a molto più del singolo minuto.

5 Risultati

Date le valutazioni effettuate, compresa la complessità del problema, assimilato e definito l'approccio, implementato il modello e valutato, i risultati del modello stesso, nonostante un tasso di precisione del 75%, sono stati ottimi ma migliorabili.

Il problema sicuramente non è banale da risolver né da astrarre, la situazione dell'ambiente essendo molto variabile e complessa porta il modello a voler preferire un'astrazione generale piuttosto che il concentrarsi sulle varie sfumature che uno stato può avere; quindi, per avere dei risultati precisi è stato necessario affrontare moltissime epoche di addestramento, per comunque avere un margine di miglioramento ampio.

Temporalmente il modello ha mostrato massime capacità di ottimalità, la convergenza delle ricompense, dopo vari addestramenti, ha mostrato il limite del voler rappresentare solo due azioni possibili ma anche l'ottimalità della scelta, dato che il modello ha mostrato subito di essere riuscito ad apprendere stabilmente quali sono i valori ottimale per stati già affrontati in passato.

Ovviamente il modello stesso e la rappresentazione adottata hanno molti margini di miglioramento.

6 Considerazioni finali

Il progetto *"Cydon-IA"* rappresenta un'interessante esperimento di applicazione dell'intelligenza artificiale alle battaglie Pokémon 1 vs. 1.
Le seguenti considerazioni riflettono l'analisi del processo di sviluppo, l'implementazione del modello e i risultati ottenuti:

- **Complessità del problema**: le battaglie Pokémon sono complesse e soggette a molte variabili. La scelta di rappresentare lo stato corrente tramite una tupla semplifica l'approccio, ma potrebbe non catturare tutte le sfumature dell'ambiente.
- **Apprendimento per rinforzo**: l'approccio basato sul Q-Learning è appropriato per affrontare il problema. La valutazione del vantaggio del turno e la funzione di reward sono fondamentali per guidare l'agente verso strategie efficaci.
- **Limiti e ottimizzazioni**: il modello ha mostrato buone prestazioni, ma può essere ulteriormente ottimizzato. La convergenza dei valori Q e la scalabilità su nuove situazioni richiedono attenzione continua.
- **Tempo di risposta e decisione**: il modello suggerisce mosse in tempi brevi, ma la decisione umana può variare notevolmente. La pressione della battaglia e la complessità delle scelte influenzano il tempo di decisione.

In conclusione, *"Cydon-IA"* rappresenta un primo approccio all'applicazione dell'intelligenza artificiale al mondo dei Pokémon tramite il Q-learning.