Relazione progetto Machine Learning e Sistemi Intelligenti per Internet

Giordano Andreola 546572
Matteo Cardilli 550208
Simone Albero 552166

Link repository GitHub: https://github.com/Friscobuffo/briscola

Introduzione

L'obiettivo del nostro lavoro è stato quello di creare un'intelligenza artificiale in grado di giocare, e vincere, a briscola. Tra le sfide che ci siamo imposti la più grande è stata sicuramente quella di non utilizzare alcuna libreria che fornisca, nemmeno in parte, le funzioni necessarie al compimento del lavoro (quantomeno per quanto riguarda il gioco e gli algoritmi di reinforcement learning). In particolare sono state utilizzate tecniche più "tradizionali" del machine learning, per rimanere più possibile in tema con il corso; non sono state quindi utilizzate reti neurali, nonstate queste generalmente siano più efficaci, a favore invece di algoritmi di reinforcement learning più "puri".

Nello specifico, al fine di valutare lo stesso problema con approcci e metodologie tra loro diversificate, si è scelto di approfondire più di un algoritmo, tra questi distinguiamo:

- Montecarlo
- Sarsa
- Qlearning

Inoltre si è progettato ed integrato, un ambiente rappresentativo per il gioco della briscola al quale è stato delegato il compito di simulare il gioco fornendo informazioni significative sullo stato delle partite e generando i reward utili in fase di apprendimento.

Per tutte le nozioni teoriche relative al reinforcement learning e agli algoritmi approfonditi, rimandiamo al libro “Sutton & Barto, Reinforcement Learning: An Introduction”.

Vincoli di progetto

Il linguaggio di programmazione scelto per la realizzazione del progetto è Python, in particolare la versione 3.10.12.

Si specifica inoltre che, al fine di ridurre per quanto possibile i vincoli esterni al progetto, come precedentemente illustrato, si è scelto di integrare nel codice

esclusivamente le librerie utili a semplificare le operazioni di più basso livello quali gestione di file e matrici. L'elenco delle suddette librerie è il seguente:

- random, versione: 1.2.2
- numpy, versione: 1.25.2
- pickle (libreria di sistema)
- os (libreria di sistema)
- time (libreria di sistema)

Specifiche hardware

Tutte le simulazioni e dunque le statistiche che verranno in futuro analizzate, sono state generate in riferimento ai seguenti dispositivi:

- Dispositivo 1:
  - Computer: MacBook Pro
  - Sistema operativo: Ventura 13.4
  - Chip: Apple M1 Pro
  - Memoria RAM: 16GB
- Dispositivo 2:
  - Computer: Honor MagicBook 15
  - Sistema operativo: Fedora 38 Workstation
  - CPU: AMD Ryzen 5 3500U
  - Memoria RAM: 8GB

Per brevità in seguito si farà riferimento al modello del computer per specificare il calcolatore utilizzato in fase di apprendimento.

L'ambiente

Rappresenta un'astrazione delle procedure tipiche del gioco della briscola, oggetto del nostro apprendimento. Si è cercato di renderlo il più possibile disaccoppiato dall'algoritmo di apprendimento, permettendoci di valutare lo stesso problema con metodi e approcci differenti, limitando al massimo i cambiamenti.

In particolare, l'ambiente implementa una interfaccia che fa riferimento a due procedure nello specifico. Questo tipo di approccio si presta bene ai problemi di reinforcement learning e rappresenta una versione semplificata dell'ambiente diffuso nella libreria "Gym" sviluppata da openAI. Nello specifico, la prima procedura, ha il compito di inizializzare l'ambiente, riconducendo il gioco allo stato di partenza. Formalmente non ha parametri e restituisce lo stato iniziale. La seconda procedura ha invece il compito di eseguire un passo avanti nel gioco, aggiornando le informazioni sullo stato della partita che ne scaturiscono e generando i reward. Formalmente prende una azione come parametro e restituisce stato, reward e informazioni supplementari quali il termine della partita.

Occorre tuttavia sottolineare che questa rappresenta per l'appunto un'astrazione, pertanto è stata utilizzata esclusivamente come punto di riferimento nella concretizzazione dei vari approcci che differiscono di caso in caso facendo capo alla struttura precedentemente illustrata.

Stato e azioni

Preliminarmente, è fondamentale notare che nessun membro del nostro gruppo può vantare un livello di competenza nel gioco della briscola. Tuttavia, dopo una serie di iterazioni e attente considerazioni, la definizione dello stato ha raggiunto una forma definitiva. La sfida principale in questo contesto è stata indubbiamente quella di trovare un punto di equilibrio tra la raccolta del massimo numero di informazioni rilevanti nello stato e la sua dimensionalità, con l'obiettivo di massimizzare l'efficacia dell'intelligenza artificiale durante il processo di addestramento. Va notato che la decisione iniziale di evitare l'impiego di una rete neurale di dimensioni più contenute in termini di spazio su disco e memoria RAM ha portato all'adozione di tecniche che richiedono una considerevole quantità di risorse, in quanto ogni coppia (azione, stato) deve essere memorizzata insieme al valore da essa generato. È evidente che l'approccio seguito è estremamente sensibile alle dimensioni dello stato, le quali comportano un aumento esponenziale del tempo e delle risorse necessarie.

Per tale ragione, all'inizio del progetto, le iniziali definizioni dello stato da noi concepite erano indubbiamente più esaustive dal punto di vista informativo rispetto alla versione corrente. Tuttavia, queste definizioni erano anche caratterizzate da una certa ingenuità, poiché hanno provocato numerosi malfunzionamenti dovuti al fatto che il dizionario (o la matrice) contenente l'insieme completo delle coppie stato-azione valore non poteva semplicemente essere allocato nella memoria RAM. A ciò si aggiunge che il numero di configurazioni potenziali per le partite di briscola è di proporzioni praticamente astronomiche. Si consideri, a titolo esemplificativo, che il solo numero di disposizioni possibili per mescolare un mazzo di 40 carte è pari a 40 fattoriale, ovvero oltre 8×10^{47}. Inoltre, da ciascuna di queste disposizioni può scaturire un numero virtualmente infinito di partite diverse, in base alle scelte effettuate dai due giocatori. Pertanto, è irrealistico tentare di rappresentare esaustivamente ogni singolo stato possibile, richiedendo invece una serie di compromessi.

Di seguito viene riportata una visione di alto livello dell'ultima definizione dello stato:

- informazioni sulle carte in mano del giocatore:
    - semeCarta0, fasciaPuntiCarta0 (4×3 possibilità)
    - semeCarta1, fasciaPuntiCarta1 (4×3 possibilità)
    - semeCarta2, fasciaPuntiCarta2 (4×3 possibilità)
- informazioni sui punti accumulati dall'avversario:
    - puntiAvversarioAlmeno45 (true/false)
- informazioni sulla carta in fondo al mazzo:

    - puntiCartaInFondoAlmeno10 (true/false)
    - briscola (4 possibilità)
- informazioni sul numero di briscole uscite:
    - fasciaBriscoleUscite (5 possibilità)
- informazioni sui carichi usciti:
    - almenoUncaricoDenaraUscito (true/false),
    - almenoUnCaricoSpadeUscito (true/false),
    - almenoUnCaricoBastoniUscito (true/false),
    - almenoUnCaricoCoppeUscito (true/false)
- eventuali informazioni sulla carta tirata per prima.
    - semeCartaTirata, fasciaPuntiCartaTirata (4×3 possibilità, presenti solo in caso il giocatore stia tirando la sua carta per secondo)

Nel dettaglio, al fine di mitigare la complessità della rappresentazione degli stati, abbiamo optato per una strategia diversa dalla memorizzazione specifica delle carte, ognuna delle quali potrebbe assumere quaranta diverse identità. Invece, si è scelto di rappresentare ciascuna carta mediante due attributi: il seme (con quattro possibili valori) e il valore espresso in punti, che per ridurre ulteriormente la dimensionalità, è stato aggregato in tre macrocategorie: zero punti, punti compresi tra due e quattro, e punti maggiori o uguali a dieci. Questo approccio ci consente di rappresentare ogni carta attraverso 4 (seme) moltiplicato per 3 (fasci di valori), per un totale di 12 possibili combinazioni. Per quanto riguarda, invece, la variabile "fasciaBriscoleUscite", essa segue la medesima logica delle fasce precedentemente illustrate e rappresenta il numero di briscole già giocate. La sua codifica è la seguente: assume il valore di 0 quando il numero di briscole precedentemente giocate è inferiore a 7, altrimenti il suo valore coincide con il numero effettivo di briscole giocate. Pertanto, questa variabile può assumere cinque valori distinti: 0, 7, 8, 9 e 10. Le rimanenti variabili dovrebbero risultare autoesplicative.

Relativamente alle azioni, la loro rappresentazione adotta un approccio notevolmente più semplice ed intuitivo, poiché le associa direttamente alle carte selezionate per essere giocate dalla mano. In particolare, vengono descritte mediante tre valori distinti: 0, 1 e 2.

Per determinare il numero totale di stati possibili in base a questa definizione, è necessario considerare diverse situazioni aggiuntive. In particolare, dobbiamo tenere presente che il giocatore non ha sempre tre carte in mano; nel penultimo turno, ad esempio, dispone solo di due carte tra cui scegliere. Inoltre, ci sono momenti in cui il giocatore effettua il secondo tiro, e in questi casi è essenziale includere informazioni sulla carta che è stata giocata per prima. Al contrario, ci sono situazioni in cui il giocatore effettua il primo tiro. Inoltre, ogni stato deve essere moltiplicato per il numero di azioni possibili, poiché il gioco della briscola non può essere rappresentato come un modello perfetto. Ciò significa che, dato uno stato e un'azione, non è possibile prevedere in modo deterministico quale sarà lo stato

successivo. Pertanto, è necessario considerare tutte le possibili azioni che un giocatore potrebbe compiere in un determinato stato, aumentando così la complessità del numero totale di stati possibili. Alla luce delle considerazioni sopra esposte, è possibile dedurre quanto segue:

- configurazione del giocatore che tira per primo con tre carte in mano: $(4\times3\times4\times3\times4\times3)\times2\times2\times5\times(2\times2\times2\times2)\times3 = 1.658.880$
- configurazione del giocatore che tira per primo con due carte in mano: $(4\times3\times4\times3)\times2\times2\times5\times(2\times2\times2\times2)\times3 = 138.240$
- configurazione del giocatore che tira per secondo con tre carte in mano: $(4\times3\times4\times3\times4\times3)\times2\times2\times5\times(2\times2\times2\times2)\times(4\times3)\times3 = 19.906.560$
- configurazione del giocatore che tira per secondo con due carte in mano: $(4\times3\times4\times3)\times2\times2\times5\times(2\times2\times2\times2)\times(4\times3)\times3 = 1.658.880$

Il numero possibile di stati è quindi, sommando tutte le possibilità di ogni configurazione, 23.362.560.

Strutture dati e memorizzazione

Una volta chiarita la complessità della dimensionalità del problema, ci si è concentrati sull'analisi delle strutture dati destinate a immagazzinare le coppie (azione, stato) e i valori a esse correlati. In questa fase, si è esaminato il problema da secondo due prospettive differenti così da ottenere una valutazione più completa delle potenziali soluzioni.

Il primo approccio ricorre ad un dizionario, in cui le coppie (azione, stato) fungono da chiavi e i corrispondenti valori corrispondono ai valori associati a tali coppie. Questo metodo permette di popolare incrementalmente il dizionario durante il processo di addestramento, man mano che nuove coppie (azione, stato) vengono incontrate durante le simulazioni. Tuttavia è opportuno notare che questa struttura dati, sebbene molto efficiente in termini di dimensioni poiché non richiede la pre allocazione di tutte le possibili coppie (azione, stato) caratteristiche del problema, comporta un notevole costo temporale per l'accesso ai valori tramite le chiavi. Tale costo cresce in modo direttamente proporzionale alla dimensione del dizionario. Un altro punto di forza dei dizionari risiede nella loro flessibilità riguardo alla forma delle chiavi. Questa caratteristica si è dimostrata particolarmente vantaggiosa data la natura del gioco della briscola che, prevedendo diverse configurazioni dello stato, come precedentemente analizzato, si riflette sulla configurazione della chiave. Vale la pena anche menzionare il fatto che utilizzando i dizionari risulta molto più naturale verificare il numero di stati esplorati il quale corrisponderà banalmente alla dimensione del dizionario, in quanto stati non ancora esplorati, infatti, semplicemente non saranno inclusi nel dizionario stesso.

Il secondo approccio, invece, si basa sull'utilizzo di una struttura dati nella forma di una matrice multidimensionale o tensore. Questa matrice, se proiettata in una

visione "bidimensionale", associa a ciascuno stato un vettore contenente le azioni possibili da quel particolare stato. Va notato che questo approccio, a causa della natura della struttura dati impiegata, è notevolmente più rigido e vincolante rispetto al precedente, poiché richiede la preallocazione completa della matrice e una formalizzazione più rigorosa degli stati in anticipo. Tuttavia, dal punto di vista delle prestazioni temporali, è notevolmente più efficiente poiché gli accessi avvengono in tempo costante.

Metrica di valutazione

Per valutare le prestazioni delle intelligenze artificiali è stato deciso di farle giocare contro un avversario che esegue esclusivamente mosse casuali, per poi misurarne la percentuale di vittoria. A tal proposito tra l'ia e il "giocatore casuale" vengono simulate un numero sufficientemente grande di partite in modo da rendere l'impatto delle fluttuazioni casuali il più basso possibile, stimando quindi il più precisamente possibile la "vera" percentuale di vittoria contro il "giocatore casuale".
Una metrica così definita è sufficientemente oggettiva e rappresentativa per i nostri scopi, fatta la ragionevole assunzione che all'aumentare della bravura delle intelligenze artificiali aumenterà anche la loro percentuale di vittoria contro un avversario che esegue mosse casuali.

Filosofie di addestramento

Nel processo di addestramento, si è generalmente fatto ricorso all'approccio conosciuto come "soft-epsilon". In questo contesto, l'IA, partendo da uno stato specifico, seleziona con una probabilità uno meno epsilon una determinata azione che è localmente ottimale (in termini di sfruttamento), mentre con una probabilità pari epsilon, sceglie un'azione in modo casuale. Questa metodologia è stata adottata al fine di agevolare l'esplorazione dell'intero spazio degli stati, insieme alla presa di decisioni orientate all'ottimizzazione locale.
In successive implementazioni si è scelto di ridurre gradualmente il valore di epsilon all'aumentare degli episodi di addestramento, mirando a rendere la policy inizialmente più incline a fare mosse esplorative, per poi essere progressivamente sempre più orientata verso mosse greedy, in modo da continuare a raffinare la conoscenza dell'intelligenza artificiale delle sole mosse ritenute migliori.

Nonostante il criterio di addestramento generale precedentemente delineato, è importante sottolineare che le vere e proprie decisioni di progettazione emergono quando si affronta la gestione dei due giocatori coinvolti nel gioco della briscola. Infatti, l'andamento dell'addestramento è strettamente condizionato non solo dalle azioni selezionate dall'IA, ma altresì dalle azioni intraprese dal suo avversario. In tal senso, è essenziale definire il criterio utilizzato per la scelta delle azioni compiute dall'avversario. Anche in questo specifico ambito, si è optato per l'approfondimento di due approcci distinti.

Il primo approccio adottato consiste nell'effettuare l'addestramento simultaneo di due IA. Questa strategia si basa sull'assunzione che, con l'accumularsi degli episodi, entrambe le IA migliorano nel processo di selezione delle azioni, consentendo così ad una di apprendere dall'altra. Va notato che questo approccio è certamente valido e, come verrà ulteriormente esplorato, produce i risultati più promettenti. Tuttavia, dal punto di vista computazionale, comporta un dispendio maggiore di risorse, in quanto richiede ad ogni step l'aggiornamento di due IA separate.

Il secondo approccio, al contrario, implica l'addestramento di una singola IA. Questo si realizza mediante la definizione di una soglia prestabilita, la quale, una volta superata, genera una copia fittizia dell'IA in una specifica fase dell'apprendimento. In questo contesto, l'avversario non è altro che una duplicazione dell'IA stessa, la quale adotta costantemente una politica "greedy," permettendo così all'IA "principale" di progredire gradualmente durante la sequenza di episodi. È rilevante sottolineare che questo tipo di approccio risulta computazionalmente meno dispendioso, poiché comporta l'addestramento di una sola IA.

Reward

Per quanto concerne il sistema di assegnazione dei reward, esso si basa sul principio logico di attribuire un reward di +1 in caso di vittoria, -1 in caso di sconfitta e 0 in caso di pareggio. Questo approccio definisce chiaramente gli obiettivi dell'IA, orientandola in modo inequivocabile verso la ricerca della vittoria finale e, di conseguenza, lasciando meno spazio per altre considerazioni nelle sue decisioni.

Tuttavia, in questo contesto, si è anche considerata un'alternativa metodologica pensata soprattutto per gli algoritmi Sarsa e Q-learning, i quali si basano sulla valutazione di due stati: lo stato attuale e quello successivo. In questa prospettiva, è stata presa la decisione di aumentare notevolmente il valore assoluto dei reward precedentemente descritti e di assegnare ulteriori reward, seppur di molto inferiori, proporzionalmente ai punti associati alle giocate di ogni step all'interno di ciascun episodio. Per l'appunto, è importante che questi reward aggiunti siano decisamente minori dei reward assegnati alla fine dell'episodio, in modo che l'intelligenza artificiale non perda di vista l'obiettivo finale, ovvero la vittoria della partita, in quanto attratta da reward immediati che porterebbero a soluzioni sub-ottimali. A tal proposito, il gioco della briscola si presta decisamente bene in quanto è possibile calcolare i punti relativi ad ogni giocata, già formalmente espressi nelle regole del gioco. È importante evidenziare che questo approccio potrebbe discostarsi da una prospettiva più convenzionale; tuttavia, si è scelto di esplorarlo principalmente a scopo didattico.

Parallelismo

Nonostante Python sia notoriamente ostile in termini di parallelismo, in quanto non è ancora presente un parallelismo a livello di thread, ma solo a livello di processi, un

tentativo di parallelizzare il codice è stato effettuato, seppur con scarsi risultati in termini di efficienza.
L'unica parte del codice che avrebbe senso parallelizzare è la parte in cui le due ia si scontrano, è stato quindi pensato di far giocare alle ia più partite contemporaneamente e solo alla fine procedere con l'aggiornamento del valore degli stati, in questo modo almeno in linea di principio l'apprendimento dovrebbe risultare più rapido, quantomeno soprattutto nelle fasi iniziali dell'addestramento delle ia in quanto potrebbero esplorare più velocemente nuovi possibili stati.
La realizzazione tuttavia, come anticipato prima, è risultata piuttosto fallimentare, peggiorando le prestazioni rispetto alla versione puramente seriale di circa 50 volte, in termini di partite giocate per secondo. Ciò è sicuramente dovuto al notevole overhead originato dalla creazione di un nuovo processo per ogni diversa partita, la quale dura troppo poco perché valga la pena dedicargli un processo tutto suo.

Conclusioni

In conclusione, gli addestramenti effettuati in base alle considerazioni precedentemente esposte sono stati portati a termine con successo nei casi degli algoritmi Montecarlo e Sarsa, mostrando buoni risultati. Questi addestramenti ci hanno fornito misure prestazionali utili per valutare le diverse scelte progettuali sperimentate. Di seguito sono riportati i dati più significativi nel contesto dei diversi addestramenti effettuati.

- Addestramento 1
    - Algoritmo utilizzato: MonteCarlo
    - Struttura dati: dizionario
    - Filosofia di addestramento: doppia IA
    - Sistema di reward: tradizionale
    - Policy: soft-epsilon
    - Dispositivo: Honor MagicBook 15
    - Tempo totale addestramento: 17h 14m
    - Totale stati esplorati: 6.002.680
    - Totale partite addestramento: 150.000.000 circa
    - Dimensione su disco: 262 MB
    - Simulazione contro giocatore che fa mosse casuali
        - Percentuale vittoria: [92.06%]
        - Percentuale pareggio: [0.72%]
        - Percentuale sconfitta: [7.22%]

- Addestramento 2
    - Algoritmo utilizzato: MonteCarlo
    - Struttura dati: dizionario
    - Filosofia di addestramento: doppia IA
    - Sistema di reward: tradizionale

    - Policy: soft-epsilon con decremento
    - Dispositivo: Honor MagicBook 15
    - Tempo totale addestramento: 7h 12m
    - Totale stati esplorati: 5.200.000
    - Totale partite addestramento: 60.000.000 circa
    - Dimensione su disco: 210 MB
    - Simulazione contro giocatore che fa mosse casuali
        - Percentuale vittoria: [90.12%]
        - Percentuale pareggio: [0.79%]
        - Percentuale sconfitta: [9.09%]

- Addestramento 3
    - Algoritmo utilizzato: Sarsa
    - Struttura dati: matrice
    - Filosofia di addestramento: singola IA
    - Sistema di reward: alternativo
    - Dispositivo: MacBook Pro M1
    - Policy: soft-epsilon con decremento
    - Tempo totale addestramento: 7h 30m
    - Totale partite addestramento: 70.000.000 circa
    - Dimensione su disco: 1.97 GB
    - Simulazione contro giocatore che fa mosse casuali
        - Percentuale vittoria: [72.10%]
        - Percentuale pareggio: [1.31%]
        - Percentuale sconfitta: [26.59%]

Tuttavia, è fondamentale notare che queste valutazioni sono ancora stime che richiedono ulteriori approfondimenti e studi, specialmente considerando che derivano da un insieme di scelte progettuali effettuate in contemporanea.