Università degli Studi di Salerno

Dipartimento di Informatica
Corso di Laurea Triennale in Informatica

Fondamenti di Intelligenza Artificiale (FIA)

T.W.A.I.

(Tris Was Already Invented)

Docente	**Studenti**	**Matricola**
Prof. Fabio Palomba	Tozza Gennaro Carmine	0512120382
	Valva Lorenzo	0512119639

[`https://github.com/gennarocarmine/TWAI.git`]

Anno Accademico 2025-2026

Indice

Sommario

Il presente elaborato documenta lo sviluppo di T.W.A.I. (Tris Was Already Invented), un sistema di Intelligenza Artificiale applicato al gioco Forza 4. L'obiettivo del progetto è confrontare due paradigmi fondamentali dell'IA: l'approccio simbolico, implementato tramite l'algoritmo Minimax con ottimizzazioni Alpha-Beta Pruning e Move Ordering, e l'approccio connessionista, realizzato mediante una Rete Neurale MLP (Multi-Layer Perceptron). Particolare rilievo assume la fase di *Data Engineering*, in cui l'agente simbolico è stato impiegato come Oracolo per la generazione e l'etichettatura automatica di un dataset sintetico di oltre 100.000 configurazioni. I risultati sperimentali evidenziano il trade-off tra i due modelli: mentre l'approccio neurale offre un tempo di inferenza costante ($O(1)$) con un'accuratezza dell'86%, l'algoritmo di ricerca garantisce la correttezza formale e, grazie alle ottimizzazioni introdotte, raggiunge prestazioni *real-time* (0.13s) a profondità competitive, pur rimanendo vincolato alla complessità esponenziale dello spazio degli stati.

1 Introduzione

Il periodo festivo porta con sé tradizioni imprescindibili: i panettoni, le calze e le discussioni con i parenti che, pur lamentandosi dell'invadenza dell'Intelligenza Artificiale, passano il tempo a guardare video di gattini che ballano sui social. Ma porta anche le immancabili giocate a carte. È stato proprio durante l'ennesima partita persa a *Sette e Mezzo* che è sorta la domanda fatidica: "E se al nostro posto giocasse un'IA?".

L'idea embrionale del progetto, inizialmente battezzata con il nome in codice **(IA)'M FINE** (un gioco di parole sulla dichiarazione "sto bene" usata per non ricevere altre carte), mirava a creare un agente imbattibile in questo gioco tradizionale.

Tuttavia, durante la fase di analisi preliminare, è emersa una criticità strutturale. A differenza di giochi come il Blackjack, dove il banco espone parzialmente il proprio stato (una carta scoperta e una coperta), nel Sette e Mezzo l'interazione inizia con un livello di informazione imperfetta molto marcato, dato che la carta iniziale è privata e coperta. Questa caratteristica sposta il problema dal ragionamento strategico puro alla gestione della probabilità e del rischio (stocasticità), rendendo difficile un confronto diretto tra l'efficacia di algoritmi deterministici come il Minimax.

Cambio di Rotta

Cercando un'alternativa deterministica (a informazione perfetta), il pensiero è corso subito al gioco più classico: il Tris (Tic-Tac-Toe). Ma anche qui sorgeva un problema opposto: il Tris è un gioco banale, con uno spazio degli stati minuscolo e una strategia ottimale che porta inevitabilmente al pareggio.
Da qui nasce il nome definitivo del progetto: **T.W.A.I.** (*Tris Was Already Invented*).

Abbiamo quindi scelto il "fratello maggiore" del Tris: **Forza 4** (Connect-4). In questo contesto, l'intero stato del gioco è visibile a entrambi i contendenti, eliminando il fattore fortuna e permettendo di concentrare l'analisi esclusivamente sulle capacità computazionali e strategiche dell'Intelligenza Artificiale.

2 Definizione del problema

Connect Four (noto anche come Connect 4, Four Up, Plot Four, Find Four, Captain's Mistress, Four in a Row, Drop Four, e in Unione Sovietica, Gravitrips) è un gioco in cui i giocatori scelgono un colore e poi si alternano lasciando cadere gettoni colorati in una griglia a sei file, sette colonne verticalmente sospesa.
I pezzi cadono direttamente, occupando lo spazio più basso disponibile all'interno della colonna. L'obiettivo del gioco è quello di essere il primo a formare una linea orizzontale, verticale o diagonale di quattro dei propri gettoni. [5]

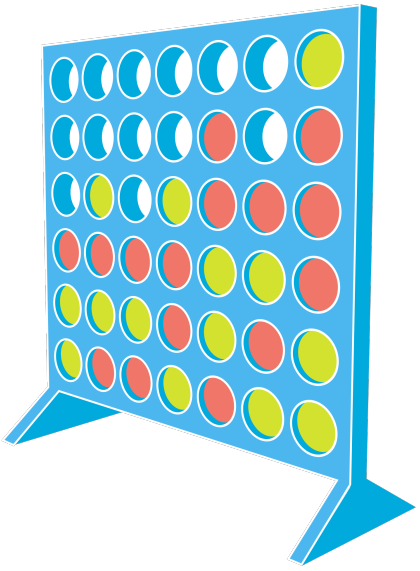

Figura 1: Immagine del gioco Forza 4.

Il gioco rientra nella categoria dei giochi a somma zero, a informazione perfetta e deterministici. Sebbene le regole siano semplici, la complessità computazionale non è trascurabile.

Nonostante il gioco sia stato "risolto" matematicamente (il primo giocatore ha una strategia vincente se gioca perfettamente), per un agente limitato da risorse computazionali e tempo di risposta (real-time interaction), la ricerca esaustiva è impraticabile. Il problema richiede quindi l'utilizzo di euristiche o approssimatori di funzione per valutare la bontà di una mossa senza dover esplorare l'intero albero di gioco fino alle foglie.

2.1 Obiettivi

L'obiettivo primario del progetto è la realizzazione di un agente intelligente in grado di giocare a *Forza 4* in modo autonomo e competitivo. Il progetto mira a creare un software capace di sfidare un essere umano, superando le limitazioni di un approccio casuale attraverso l'implementazione di strategie di pianificazione e apprendimento.

Per raggiungere questo traguardo e confrontare diverse metodologie di risoluzione, il lavoro è stato suddiviso nei seguenti obiettivi specifici:

- **Generazione di un Dataset Sintetico:** Creare autonomamente i dati necessari all'addestramento, evitando dataset precostruiti. Per garantire la qualità dei dati, si adotterà un approccio di *Campionamento degli Stati* (State Sampling):
  - Generazione di configurazioni "mid-game" tramite *random walk* (mosse casuali) per coprire scenari complessi.
  - Etichettatura tramite "Oracolo": utilizzo dell'algoritmo Minimax per valutare ogni configurazione e assegnare l'etichetta di verità (Vittoria/Sconfitta/Pareggio).
- **Implementazione dell'Approccio Simbolico (Minimax):** Sviluppare un motore di gioco basato sull'algoritmo Minimax con ottimizzazione Alpha-Beta Pruning. Questo modulo serve sia come avversario "esperto" in grado di esplorare l'albero di gioco, sia come supervisore per la generazione del dataset.
- **Implementazione dell'Approccio Connessionista (MLP):** Sviluppare e addestrare una rete neurale (Multi-Layer Perceptron) sul dataset generato. L'obiettivo è ottenere un modello che apprenda la funzione di valutazione in modo statistico, garantendo tempi di risposta istantanei ($O(1)$) rispetto alla ricerca esponenziale del Minimax.
- **Sviluppo dell'Interfaccia Utente:** Creare un'applicazione interattiva che permetta all'utente finale di sfidare entrambe le intelligenze artificiali e visualizzare in tempo reale le decisioni prese dai modelli.

2.2 Specifica PEAS

La descrizione formale dell'ambiente operativo dell'agente è definita secondo il modello PEAS (*Performance, Environment, Actuators, Sensors*):

- **Performance (Misure di Prestazione):**
  - Efficienza: Numero di mosse minimo per raggiungere la vittoria.
  - Correttezza: Evitare mosse non valide (es. inserire in una colonna piena).
  - Velocità di risposta: Tempo impiegato per calcolare la mossa.
  - Tasso di vittoria: Percentuale di partite vinte contro vari tipi di avversari.
- **Environment (Ambiente):**
  - Griglia di gioco: Una matrice 6 X 7.
  - Avversario: Un essere umano o un'altra IA.
- **Actuators (Attuatori/Azioni):**
  - Inserimento gettone: l'attuatore sceglie una delle 7 colonne disponibili.
  - Segnalazione: comunicazione della mossa o dichiarazione di vittoria/resa.
- **Sensors (Sensori/Percezioni):**
  - Lettore di matrice: Funzione che scansiona lo stato attuale della scacchiera per sapere dove sono i propri gettoni, quelli dell'avversario e gli spazi vuoti.
  - Rilevatore di turno: Funzione che indica all'agente quando è il suo momento di agire.

2.2.1 Caratteristiche dell'ambiente

L'ambiente di gioco Forza 4 presenta le seguenti proprietà:

- **Completamente Osservabile:** L'agente vede l'intera griglia di gioco, non ci sono informazioni nascoste.
- **Deterministico:** Lo stato successivo dell'ambiente è completamente determinato dallo stato corrente e dall'azione eseuita dall'agente.
- **Discreto:** L'ambiente fornisce un numero limitato di percezioni e azioni distinte, chiaramente definite.

2.3 Analisi del problema

Lo spazio degli stati teorico è costituito da tutte le possibili configurazioni di una griglia 6×7 con tre possibili valori per cella (Vuoto, Giocatore 1, Giocatore 2), portando a un limite superiore di 3^{42} stati. Tuttavia, considerando i vincoli di gravità (le pedine devono poggiare su altre pedine o sul fondo), il numero reale di stati legali è stimato intorno a 4.5×10^{12}.

3 Tecnologie e Strumenti Utilizzati

Lo sviluppo di T.W.A.I. è stato realizzato interamente in linguaggio Python. La scelta di questo linguaggio è stata dettata dalla vastità del suo ecosistema per il calcolo scientifico e l'intelligenza artificiale. Di seguito vengono descritte le librerie principali che compongono l'architettura del sistema:

- **numpy:** Utilizzata per la rappresentazione in memoria della griglia di gioco. La scacchiera 6×7 è gestita come un array bidimensionale (`ndarray`) di interi. L'efficienza di NumPy nelle operazioni vettoriali e nella manipolazione di matrici è fondamentale per velocizzare le funzioni di valutazione e controllo della vittoria.
- **pandas:** Impiegata nel modulo di Data Engineering per la gestione strutturata del dataset.
- **scikit-learn:** La libreria di riferimento per l'implementazione della pipeline di apprendimento. Nello specifico, sono stati utilizzati:

  - `MLPClassifier`: Per la creazione e l'addestramento della rete neurale Multi-Layer Perceptron.
  - `train_test_split`: Per la partizione corretta del dataset in Training Set e Test Set.
  - `metrics`: Per il calcolo di accuratezza, matrici di confusione e report di classificazione.

- **matplotlib e seaborn:** Utilizzate nella fase di analisi per generare grafici e visualizzazioni (curve di loss, heatmap della matrice di confusione) utili a monitorare le performance dell'agente.

Per garantire un'interazione fluida e accessibile con gli agenti intelligenti, è stata sviluppata un'interfaccia web utilizzando **Streamlit**, un framework open-source orientato al Data Science.

La scelta di Streamlit ha permesso di trasformare gli script di backend (logica di gioco e modelli AI) in un'applicazione web interattiva in tempi ridotti, senza la necessità di gestire complessi stack front-end (HTML/CSS/JS).

4 Dataset e Data Engineering

Per soddisfare il requisito di progetto relativo alla gestione attiva dei dati, abbiamo evitato l'utilizzo di dataset pre-confezionati. Abbiamo invece sviluppato una procedura di generazione di dati sintetici, necessaria per applicare tecniche di apprendimento supervisionato in un contesto di gioco.

4.1 Metodologia di Generazione: State Sampling & Oracle Labeling

Per garantire che la rete neurale apprenda da una varietà di configurazioni non distorte dallo stile di gioco di un agente specifico, abbiamo evitato di registrare intere partite simulate. Abbiamo invece adottato un approccio basato sul campionamento degli stati, strutturato nei seguenti passi:

1. **Random Walk (Esplorazione):** Per ogni campione, viene inizializzata una scacchiera vuota. Due agenti *Random* effettuano un numero casuale di mosse $N \in [4, 24]$. Questo permette di raggiungere stati di

"metà partita" (S_{mid}) variegati e non deterministici, coprendo lo spazio degli stati in modo più uniforme rispetto a partite tra agenti esperti.

2. **Etichettatura tramite Oracolo:** Una volta raggiunto lo stato S_{mid}, il gioco viene congelato. L'algoritmo **Minimax** (con profondità di ricerca $d = 3$) analizza la configurazione e restituisce un punteggio euristico V.

3. **Classificazione e Bilanciamento:** Il punteggio V viene discretizzato nelle classi target y per l'addestramento supervisionato:

   - $y = 1$ (Vittoria P1) se $V > 50$.
   - $y = -1$ (Vittoria AI) se $V < -50$.
   - $y = 0$ (Pareggio/Incerto) altrimenti.

4.2 Struttura del File Dati (CSV)

Il dataset costruito (`connect4_dataset_hq.csv`) segue una struttura tabellare standard ed è composto da un totale di **43 colonne** per ogni riga (campione).

Le prime 42 colonne rappresentano lo stato completo della scacchiera. Poiché Forza 4 si gioca su una griglia bidimensionale di dimensioni $R = 6$ (righe) e $C = 7$ (colonne), il numero totale di celle è 42.

Per rendere i dati compatibili con l'input del Multi-Layer Perceptron, la matrice 2D viene sottoposta a un processo di **linearizzazione** (flattening), trasformandola in un vettore 1D.

- **Input (X):** Le colonne sono etichettate da `pos_0` a `pos_41`. L'indice i della feature `pos_i` corrisponde alla cella della scacchiera alle coordinate (r, c) secondo la formula $i = r \times 7 + c$. Ogni cella contiene un valore intero discreto:
  - 0: Cella Vuota.
  - 1: Pedina del Giocatore 1.
  - -1: Pedina del Giocatore 2.

- **Target (y):** L'ultima colonna, denominata `winner`, costituisce l'etichetta di classe per l'addestramento supervisionato. Questa colonna non contiene una mossa, ma la valutazione strategica fornita dall'Oracolo $(1, -1, 0)$.

Per la definizione dello schema dei dati, abbiamo preso a riferimento il *Connect-4 Game Dataset*[1]. Sebbene i dati siano stati generati internamente per garantire il controllo sulla distribuzione, abbiamo deciso di mantenere la medesima struttura logica del dataset di riferimento.

Questa scelta garantisce l'interoperabilità e permette il confronto con standard consolidati nel dominio del Machine Learning applicato ai giochi da tavolo.

5 Soluzione del problema

Il progetto TWAI implementa e confronta due approcci radicalmente diversi per la risoluzione del problema: un approccio simbolico basato sulla ricerca (Pipeline A) e un approccio connessionista basato sull'apprendimento (Pipeline B).

5.1 Pipeline A: Minimax con Alpha-Beta Pruning

Questa pipeline rappresenta l'approccio classico dell'Intelligenza Artificiale simbolica. A differenza delle reti neurali, l'agente non apprende dai dati, ma **esplora lo spazio degli stati**, simulando le conseguenze future di ogni azione possibile per determinare la strategia ottimale.

5.1.1 Cenni Teorici

L'algoritmo Minimax è un metodo ricorsivo decisionale utilizzato nella teoria dei giochi a somm zero con informazione perfetta. Dato uno stato S:

- **MAX (Agente):** Cerca di massimizzare il valore della funzione di utilità finale.
- **MIN (Avversario):** Tenta di minimizzare lo stesso valore (ovvero, massimizzare il proprio vantaggio).

Il valore di un nodo (stato di gioco) è calcolato retropagando i valori dai nodi foglia fino alla radice. Tuttavia, la complessità temporale di base O(b^d) rende l'algoritmo impraticabile per profondità elevate. Per ottimizzare la ricerca, abbiamo implementato la potatura **Alpha-Beta Pruning**.

Questa tecnica mantiene due parametri, α (miglior valore per MAX) e β (miglior valore per MIN). Se durante l'esplorazione si trova una mossa che peggiora la situazione rispetto a quanto già garantito dai rami precedenti, l'intero sotto-albero viene tagliato (pruned).

Ciò riduce la complessità temporale nel caso medio da $O(b^d)$ a $O(b^{d/2})$, permettendo all'agente di esplorare una profondità doppia rispetto al Minimax puro nello stesso intervallo di tempo.

5.1.2 Ottimizzazione: Ordinamento delle Mosse (Move Ordering)

L'efficacia della potatura Alpha-Beta dipende fortemente dall'ordine in cui le mosse vengono valutate. Nel caso peggiore, ovvero in cui le mosse migliori venissero esaminate per ultime, non avverrebbe alcun taglio e l'algoritmo degenererebbe in un Minimax puro, rendendo inutile l'ottimizzazione.

Nel **caso migliore**, se le mosse forti vengono valutate per prime, i valori di α e β convergono rapidamente, massimizzando i tagli.

Per avvicinarci a questo caso ideale, abbiamo implementato un'euristica di ordinamento statico specifica per il dominio di Forza 4. Poiché le colonne centrali della griglia offrono un numero maggiore di potenziali allineamenti (orizzontali, verticali e diagonali) rispetto a quelle laterali, è statisticamente più probabile che la mossa ottimale si trovi al centro.

Invece di iterare le colonne in ordine sequenziale (da 0 a 6), l'algoritmo esplora lo spazio delle azioni partendo dal centro verso l'esterno, seguendo l'ordine di priorità:

$$[3, 2, 4, 1, 5, 0, 6]$$

Questa semplice accortezza permette all'algoritmo di trovare rapidamente un valore di α elevato (una mossa forte), massimizzando il numero di rami tagliati (pruning) e riducendo significativamente il tempo di calcolo effettivo.

Perché Minimax e non Monte Carlo Tree Search (MCTS)?

La Monte Carlo Tree Search, resa celebre dal sistema AlphaGo di DeepMind che nel 2016 sconfisse il campione mondiale di Go Lee Sedol[4], rappresenta una delle tecniche più innovative degli ultimi due decenni nel campo dell'intelligenza artificiale per giochi. Nonostante la sua comprovata efficacia in domini complessi, abbiamo deliberatamente optato per il più classico algoritmo Minimax, motivati da considerazioni sia didattiche che pratiche strettamente legate alle caratteristiche specifiche del Forza 4.

La prima ragione fondamentale risiede nella natura del **fondamento teorico** dei due approcci. Il Minimax è un algoritmo deterministico con garanzie formali di ottimalità (assumendo una funzione di valutazione perfetta). La MCTS, basata su simulazioni stocastiche, richiede migliaia di rollout per convergere a una buona stima, introducendo variabilità nei risultati.

La seconda motivazione emerge dall'analisi dell'**efficienza computazionale relativa** nei giochi con diversi gradi di complessità combinatoria. Il Forza 4 presenta un branching factor relativamente contenuto: a ogni turno, un giocatore può scegliere tra al massimo 7 colonne (spesso meno, poiché alcune colonne potrebbero essere già piene), corrispondente a un fattore di ramificazione medio $b \approx 7$. In questo regime, il Minimax equipaggiato con le ottimizzazioni di Alpha-Beta Pruning e Move Ordering si dimostra estremamente efficiente.
La MCTS, invece, mostra il suo vero potenziale in giochi caratterizzati da branching factor proibitivamente elevati, come il Go, dove ogni mossa può essere posizionata su una delle ≈ 250 intersezioni libere della scacchiera. Per il Forza 4, tuttavia, questo vantaggio si dissolve, rendendo la MCTS una complicazione inutile che introduce overhead senza benefici tangibili.

La terza considerazione riguarda l'**interpretabilità e la trasparenza decisionale** dei due algoritmi. Il Minimax produce un albero di decisione esplicito, permettendo di analizzare esattamente perché una mossa è stata scelta. La MCTS opera tramite statistiche aggregate su simulazioni, rendendo difficile spiegare le decisioni.

In sintesi, la scelta del Minimax per il progetto T.W.A.I. riflette un principio di parsimonia ingegneristica: utilizzare la soluzione più semplice che

risolve adeguatamente il problema.

5.2 Pipeline B: Apprendimento Automatico (MLP)

La seconda soluzione applica i concetti di apprendimento supervisionato discussi nel corso, utilizzando una rete neurale per approssimare la funzione di valutazione ideale.

5.2.1 Architettura della Rete Neurale

Le reti neurali artificiali si ispirano al funzionamento del cervello umano, dove neuroni biologici interconnessi elaborano informazioni in parallelo. Nel contesto dei giochi da tavolo, una rete neurale può apprendere pattern strategici complessi osservando migliaia di configurazioni, sviluppando una sorta di "intuizione" statistica senza bisogno di regole esplicite programmate.

A differenza del Minimax, che deve esplorare l'albero di gioco ad ogni mossa, la rete neurale *pre-calcola* durante l'addestramento una funzione di valutazione approssimata. Una volta addestrata, la predizione avviene tramite una semplice moltiplicazione matriciale (forward pass), garantendo tempi di risposta costanti indipendentemente dalla complessità della posizione.

Scelta dell'Architettura: Multi-Layer Perceptron (MLP)

Abbiamo optato per un Multi-Layer Perceptron, la famiglia più classica di reti neurali feed-forward. Questa scelta è motivata da tre considerazioni fondamentali che ne garantiscono efficacia ed efficienza nel contesto specifico del Forza 4.

1. Il primo fondamento teorico risiede nel *Teorema di Approssimazione Universale*. Secondo il lavoro di Cybenko (1989) e Hornik et al. (1989), un MLP con anche un solo strato nascosto di dimensione sufficiente è in grado di approssimare qualsiasi funzione continua con precisione arbitraria, a patto di disporre di neuroni sufficienti e di una funzione di attivazione non lineare[2]. Questo garantisce teoricamente che l'MLP possa catturare la complessa funzione di valutazione del Forza 4.

2. La seconda motivazione emerge dall'analisi dei *limiti strutturali dei percettroni singoli*. I percettroni a singolo strato (introdotti da Rosenblatt

nel 1958) sono limitati alla risoluzione di problemi linearmente separabili, come dimostrato da Minsky e Papert (1969)[3]. Nel Forza 4, la valutazione di una posizione dipende da interazioni non lineari tra le pedine: una mossa può essere vincente o perdente a seconda del contesto globale della scacchiera, rendendo indispensabile l'introduzione di strati nascosti capaci di estrarre rappresentazioni intermedie che catturino queste dipendenze contestuali.

3. Il terzo fattore determinante riguarda l'*efficienza computazionale*. A differenza di architetture più complesse come le reti convoluzionali (CNN) o ricorrenti (RNN), l'MLP ha una struttura semplice che si traduce in:

    - Tempi di inferenza estremamente rapidi (alcune decine di microsecondi).
    - Facilità di addestramento su dataset di dimensioni moderate (100k campioni).
    - Basso overhead computazionale, permettendo deployment su dispositivi con risorse limitate.

Topologia Implementata

L'architettura della rete è stata progettata seguendo un principio di astrazione progressiva, in cui ogni strato trasforma le informazioni grezze in rappresentazioni sempre più sofisticate della situazione strategica.

- **Input Layer (42 Neuroni):** Riceve il vettore linearizzato della scacchiera ($6 \times 7 = 42$ celle). Ogni neurone corrisponde a una cella e codifica lo stato come valore discreto: -1 (AI), 0 (vuoto), $+1$ (Player).
- **Primo Hidden Layer (64 Neuroni):** Questo strato apprende rappresentazioni intermedie dei pattern locali (es. "tre pedine allineate con uno spazio vuoto"). La dimensione (64) è stata scelta empiricamente per bilanciare capacità espressiva e rischio di overfitting.
- **Secondo Hidden Layer (32 Neuroni):** Combina i pattern locali del primo strato in valutazioni strategiche più astratte (es. "controllo del

centro", "minacce multiple"). La riduzione progressiva della dimensione ($42 \to 64 \to 32$) implementa un'architettura a "collo di bottiglia" che forza la rete ad estrarre solo le feature più discriminanti.

- **Activation Function (ReLU):** La *Rectified Linear Unit* ($f(x) = \max(0, x)$) è stata preferita alla sigmoide.

  La *funzione sigmoide* dominante negli anni '80-'90, comprime l'input nell'intervallo $(0, 1)$ rendendolo interpretabile come probabilità. Tuttavia, soffre del problema del **vanishing gradient**: per valori estremi ($|x| > 5$), la sua derivata tende a zero, causando l'attenuazione esponenziale del segnale di errore durante la backpropagation. Questo rallenta drasticamente l'apprendimento nei primi strati della rete.

  La *ReLU*, introdotta da Nair e Hinton (2010), risolve questo problema con elegante semplicità: trasmette il segnale inalterato quando $x > 0$ (gradiente costante pari a 1), altrimenti lo azzera. Questa architettura offre tre vantaggi critici: elimina il vanishing gradient mantenendo gradienti stabili, introduce sparsità naturale (circa il 50% dei neuroni inattivi), e richiede solo un confronto anziché costosi calcoli esponenziali, accelerando significativamente training e inferenza.

- **Output Layer (3 Neuroni con Softmax):** Produce una distribuzione di probabilità sulle tre classi possibili (-1: AI vince, 0: pareggio, $+1$: Player vince). La funzione softmax garantisce che le probabilità sommino a 1, permettendo un'interpretazione probabilistica dell'incertezza del modello.

5.2.2 Addestramento (Backpropagation)

Il modello è stato addestrato utilizzando l'algoritmo di Backpropagation (propagazione all'indietro dell'errore). L'obiettivo dell'addestramento è muoversi sulla "superficie dell'errore" per trovare il minimo globale, ovvero la configurazione di pesi che minimizza la differenza tra l'output della rete e il target desiderato. [6]

Il ciclo di apprendimento per ogni epoca segue tre passi fondamentali:

1. **Forward Pass:** Il vettore di input (la scacchiera) attraversa la rete strato per strato fino a produrre un output finale.

2. **Calcolo dell'Errore:** Viene calcolata la differenza tra l'output prodotto dalla rete e il target corretto fornito dall'Oracolo Minimax.

3. **Backward Pass:** L'errore viene propagato all'indietro dall'output verso l'input. I pesi delle connessioni vengono aggiornati proporzionalmente al gradiente negativo dell'errore (Discesa del Gradiente), permettendo alla rete di correggere progressivamente le proprie predizioni.

Questo processo iterativo consente al modello di apprendere la strategia di gioco codificata nei dati sintetici, riducendo l'errore di classificazione nel tempo.

6 Valutazione Sperimentale

L'obiettivo di questa fase non è solo misurare la "bravura" dell'IA, ma valutare se il sistema è utilizzabile in un contesto reale. Abbiamo analizzato due aspetti chiave: la capacità di generalizzazione (per la Rete Neurale) e la reattività temporale (per il Minimax). I test sono stati eseguiti su una macchina locale, simulando le condizioni tipiche di un utente medio.

6.1 Analisi Rete Neurale (MLP)

Per l'approccio basato su Machine Learning, ci siamo chiesti: "La rete sta davvero imparando o sta solo memorizzando?". Per rispondere, abbiamo monitorato due indicatori grafici.

6.1.1 Curva di Apprendimento (Loss)

Il grafico della *Loss Function* rappresenta l'errore commesso dalla rete durante l'addestramento.

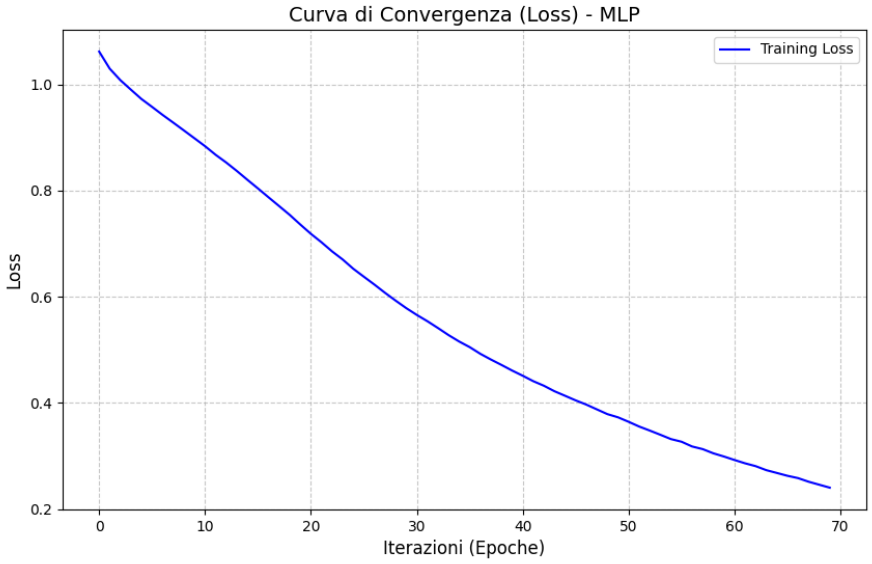

Figura 2: Curva di convergenza della Loss.

Come si nota dalla curva, l'errore crolla drasticamente nelle prime epoche: questo indica che la rete apprende molto velocemente le regole base (es. "mettere 4 gettoni in fila vince").

Successivamente, la curva si appiattisce (fase di *plateau*), indicando che il modello sta affinando la sua strategia per le situazioni più sottili. Il fatto che la curva non risalga suggerisce l'assenza di *overfitting* (la rete non sta "imparando a memoria").

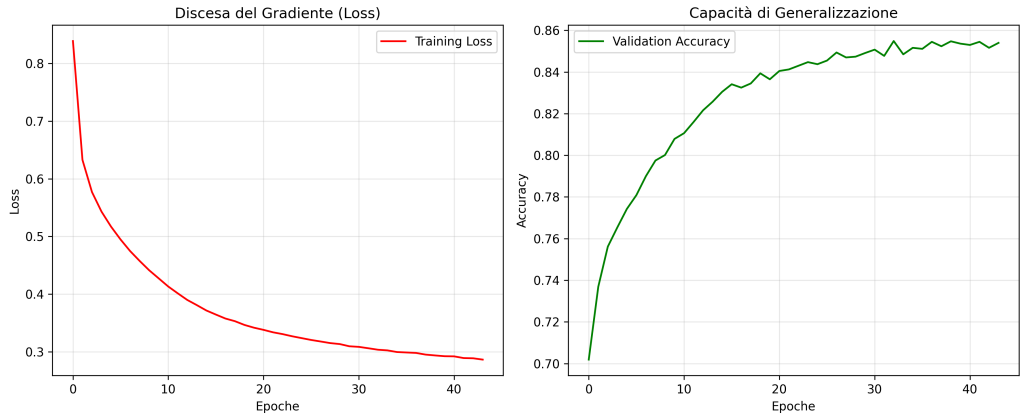

Figura 3: Curve di apprendimento: Training Loss (sinistra) e Validation Accuracy (destra).

Il grafico confronta la Loss Functioncon l'andamento della *Validation Accuracy*: il fatto che essa cresca parallelamente alla riduzione della Loss (senza divergere) dimostra che il modello possiede una buona capacità di generalizzazione e conferma l'assenza di *overfitting*.

6.1.2 Analisi degli Errori (Matrice di Confusione)

Dove sbaglia l'IA? La matrice di confusione ci aiuta a capirlo.

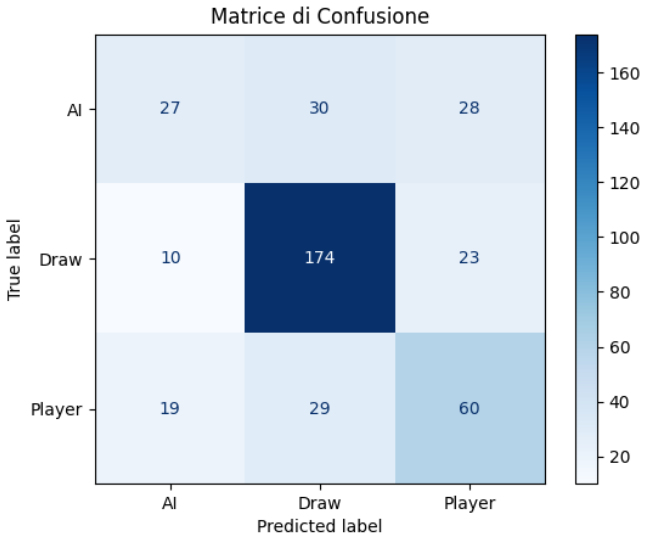

Figura 4: Matrice di Confusione (Modello preliminare, dataset ridotto).

La diagonale principale (i valori corretti) è molto marcata. La rete riconosce quasi perfettamente le vittorie e le sconfitte imminenti.

La maggior parte degli errori si concentra sulla classe "Pareggio/Incerto". Questo è comprensibile: anche per un umano è difficile prevedere un pareggio con 20 mosse di anticipo. La rete tende a essere "ottimista" o "pessimista" in situazioni di stallo, un comportamento tipico degli approcci probabilistici.

6.1.3 Impatto della Dimensione del Dataset

Un fattore determinante per le prestazioni del modello MLP è la quantità di dati forniti in ingresso. Le reti neurali sono algoritmi "data-hungry": la loro capacità di generalizzare cresce all'aumentare della varietà degli esempi visti.

Abbiamo condotto un'analisi di sensibilità addestrando la stessa architettura di rete su dataset di dimensioni crescenti, mantenendo inalterati gli iperparametri (learning rate, nodi hidden).

Tabella 1: Relazione tra dimensione del Dataset e Accuratezza

N. Campioni	Accuratezza (Test Set)	Delta Miglioramento
2000	67.12%	-
10.000	71.30%	+4.18%
20.000	75.33%	+4.03%
30.000	79.88%	+4.55%
50.000	82.32%	+2.44%
100.000	**85.95%**	**+3.63%**

Come evidenziato in Tabella 1, l'accuratezza scala in modo significativo con la quantità di dati. È interessante notare come il passaggio da 50.000 a 100.000 campioni abbia portato un ulteriore incremento del 3.63%, portando il modello a sfiorare l'86% di precisione totale.

Questo risultato conferma che per giochi posizionali complessi come Forza 4, la varietà delle situazioni di gioco (specialmente nel "mid-game") è fondamentale. Abbiamo pertanto selezionato il modello addestrato su **100.000 campioni** come versione definitiva per il deployment nell'applicazione finale.

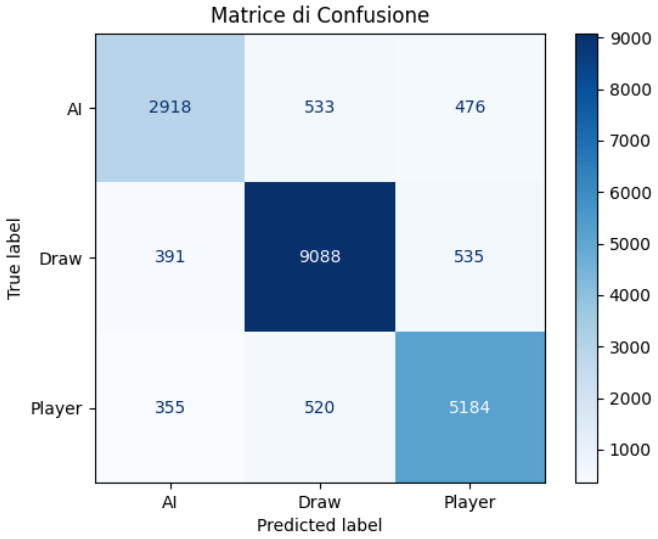

Figura 5: Matrice di Confusione (Modello finale su 100k campioni).

6.1.4 Dettaglio Metriche (Precision, Recall, F1)

Il dataset totale di 100.000 campioni è stato partizionato seguendo uno split 80/20 standard: il modello è stato addestrato su 80.000 campioni (Training Set) e successivamente valutato su 20.000 campioni inediti (Test Set).

Le metriche riportate di seguito (Precision, Recall, F1) si riferiscono esclusivamente alle performance ottenute su quest'ultimo set, per garantire una stima imparziale delle capacità di generalizzazione dell'agente.

Tabella 2: Report di Classificazione sul Test Set

Classe	**Precision**	**Recall**	**F1-Score**	**Support**
AI Wins (-1)	0.78	0.74	0.76	3946
Draw / Uncertain (0)	0.90	0.91	0.90	10110
Player Wins (1)	0.82	0.85	0.83	5944
Weighted Avg	**0.86**	**0.86**	**0.86**	**20000**

Dai dati in Tabella 2 emergono considerazioni interessanti sulla strategia appresa dalla rete:

- **Solidità sui Pareggi (Classe 0):** La rete ha un F1-Score molto alto (0.90) su questa classe. Poiché i pareggi o le situazioni di stallo sono la maggioranza nel dataset (circa il 50% del supporto), il modello è diventato molto abile nel riconoscere quando "non sta succedendo nulla di decisivo".
- **Prudenza dell'IA (Classe -1):** La *Recall* sulle vittorie dell'IA è la più bassa (0.74). Questo significa che il modello perde l'occasione di identificare alcune delle sue vittorie (Falsi Negativi), probabilmente a causa di una strategia conservativa appresa per minimizzare le sconfitte piuttosto che massimizzare le vittorie rischiose.
- **Riconoscimento Minaccia (Classe 1):** La rete è più brava a riconoscere quando sta vincendo l'avversario (Recall 0.85) rispetto a quando sta vincendo lei stessa. Questo è un comportamento desiderabile in un gioco difensivo: è meglio sovrastimare il pericolo che ignorarlo.

6.1.5 Validazione Operativa: Competenza in Partite Reali

Le metriche di classificazione (Precision, Recall, F1) misurano la capacità del modello di valutare correttamente configurazioni statiche, ma non garantiscono che l'MLP sappia *concatenare* decisioni sequenziali in una partita completa.

Per validare l'efficacia operativa, l'MLP è stato testato contro un avversario casuale in condizioni realistiche. La strategia di selezione della mossa dell'MLP opera come segue:

1. **Controllo Vittoria Immediata**: Se esiste una mossa che porta alla vittoria immediata dell'AI, viene giocata prioritariamente.
2. **Blocco Minacce**: Se l'avversario ha una mossa vincente, l'MLP la blocca preventivamente.
3. **Valutazione Posizionale**: Per ogni colonna valida, l'MLP simula mentalmente la mossa e classifica la configurazione risultante. Viene scelta la mossa che massimizza il valore predetto per l'AI (classe -1).

Questa strategia ibrida combina logica esplicita (win/block detection) con apprendimento statistico (valutazione MLP), garantendo competenza tattica di base anche in presenza di errori di classificazione.

6.2 Benchmark Minimax (Reattività)

Per l'algoritmo Minimax, la precisione è garantita dalla matematica. Il vero nemico è il tempo di esecuzione, che cresce esponenzialmente con la profondità di ricerca. In una prima fase, abbiamo misurato le prestazioni dell'algoritmo applicando la sola potatura Alpha-Beta con un ordine di visita delle colonne standard (sequenziale da 0 a 6).

Tabella 3: Tempi di esecuzione rilevati per Minimax con Alpha-Beta Pruning (Standard)

Profondità (d)	**Tempo Medio (s)**	**Nodi Stimati**
1	0.0002	~ 7
2	0.0006	~ 49
3	0.0034	~ 340
4	0.0174	$\sim 2,400$
5	0.0757	$\sim 16,800$
6	**0.2858**	$\sim 117,000$

Dai dati in Tabella 3 emerge chiaramente la natura esponenziale del problema: passando da profondità 5 a 6, il tempo quadruplica (da 0.07s a 0.28s). Sebbene 0.28s sia un tempo accettabile, lascia poco margine per ulteriori approfondimenti.

Per mitigare questo effetto, abbiamo introdotto l'ottimizzazione di Ordinamento delle Mosse, forzando l'algoritmo a valutare prima le colonne centrali (più promettenti). Questo aumenta la probabilità di effettuare "tagli" (pruning) anticipati nell'albero di ricerca. I risultati migliorati sono riportati di seguito:

Tabella 4: Tempi di esecuzione rilevati per Minimax con Alpha-Beta Pruning + Ordinamento delle Mosse

Profondità (d)	**Tempo Medio (s)**	**Nodi Stimati**
1	0.0005	~ 7
2	0.0009	~ 49
3	0.0051	~ 340
4	0.0111	$\sim 2,400$
5	0.0310	$\sim 16,800$
6	**0.1328**	$\sim 117,000$
7	0.2956	$\sim 820,000$

Analisi del miglioramento

Il confronto tra le due configurazioni evidenzia drasticamente l'efficacia dell'ottimizzazione. A parità di profondità $d = 6$, il tempo di calcolo è stato abbattuto di oltre il **50%**, scendendo da **0.2858s** a **0.1328s**.

Un dato particolarmente interessante emerge osservando il passo successivo: spingendo l'algoritmo ottimizzato a profondità $d = 7$, il tempo di esecuzione risale a **0.2956s**. Questo valore è quasi sovrapponibile a quello impiegato dall'algoritmo non ottimizzato per completare la ricerca a $d = 6$ (0.2858s). Pertanto, spingersi a $d = 7$ rimane sconsigliato, poiché significherebbe vanificare il margine di guadagno ottenuto, riportando il sistema ai limiti della latenza percepibile.

Abbiamo identificato in $d = 6$ la configurazione ideale per il sistema finale. Con un tempo di risposta ora ben al di sotto della soglia di percezione (0.13s), l'IA offre un'esperienza di gioco fluida e immediata ("Real-time"), pur mantenendo una profondità di analisi strategica elevata.

6.3 Analisi del Trade-off:

Il confronto finale evidenzia due filosofie opposte:

- **Minimax** è la scelta obbligata quando la correttezza tattica è critica (es. finali di partita, tornei). Il costo è una latenza crescente all'aumentare della profondità richiesta.
- **MLP** eccelle in scenari dove la velocità di risposta è prioritaria (interfacce real-time, dispositivi mobili). Una volta addestrata, ha un tempo di inferenza costante e bassissimo ($O(1)$), indipendentemente dalla complessità della scacchiera. Il compromesso è l'accettazione di errori occasionali.

7 Conclusioni

Il progetto T.W.A.I. ha raggiunto l'obiettivo di confrontare due paradigmi fondamentali dell'Intelligenza Artificiale applicati a un dominio a informazione perfetta.
I risultati confermano che, mentre l'approccio simbolico (Minimax) rimane insuperabile in termini di correttezza formale, l'approccio connessionista (MLP) offre un vantaggio decisivo in termini di scalabilità temporale, al costo di una marginale perdita di precisione.

L'esperienza ha evidenziato che la qualità del dataset è più determinante dell'architettura della rete. Il passaggio da un dataset di 2k campioni (67% accuracy) a 100k (86% accuracy) ha portato un miglioramento di 19 punti percentuali, dimostrando che "più dati battono algoritmi migliori" (Halevy et al., 2009). L'uso del Minimax come oracolo per etichettare stati intermedi (State Sampling) si è rivelato cruciale: evitando di registrare intere partite, abbiamo garantito una copertura uniforme dello spazio degli stati, includendo configurazioni rare ma tatticamente rilevanti.

T.W.A.I. dimostra che, sebbene "Tris Was Already Invented", c'è ancora spazio per l'innovazione nell'ambito dei giochi da tavolo. La coesistenza di approcci simbolici e connessionisti non è una competizione, ma una complementarità: il Minimax fornisce l'oracolo per addestrare l'MLP, che a sua volta permette deployment in scenari dove il Minimax sarebbe troppo lento.

In un'epoca dominata da modelli sempre più complessi, il progetto riafferma un principio fondamentale: **la semplicità funziona**.

Riferimenti bibliografici

[1] *Connect-4 Game Dataset*. URL: https://www.kaggle.com/datasets/tbrewer/connect-4.

[2] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals and Systems* 2.4 (1989), pp. 303–314.

[3] Prof. Dr. Martin Riedmiller. *Multi Layer Perceptrons*. URL: http://machine-learning-lab.com/_media/documents/teaching/ss12/ml/05_mlps.printer.pdf.

[4] David Silver, Aja Huang, Chris J Maddison et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489.

[5] Wikipedia. *Connect Four*. URL: https://en.wikipedia.org/wiki/Connect_Four.

[6] Barry J. Wythoff. "Backpropagation neural networks: A tutorial". In: *Chemometrics and Intelligent Laboratory Systems* 18(2), pp.115-155. (1993).