PROGETTO INFORMATICO IN MACHINE LEARNING

Un giocatore di tic-tac-toe realizzato attraverso reti neurali

Sviluppato da: Enrico Cagnazzo

Sommario

Abstract

Lo scopo di questo progetto è creare una rete neurale di tipo MultiLayer perceptron (MLP) che impari autonomamente a giocare a tic-tac-toe conoscendo sole le regole basilari:

- Si inizia il gioco con una griglia 3x3 vuota in cui i giocatori X e O posizionano il proprio simbolo su una casella vuota alternandosi tra loro
- Il giocatore che riesce a posizionare tre propri sulla stessa riga, colonna o diagonale vince
- Il gioco finisce quando uno dei due giocatori vince o la griglia è piena

L'addestramento avviene confrontando la soluzione ottenuta dalla rete neurale con quella ricevuta dall'algoritmo MiniMax che, data la sua natura *brute force*, garantisce che la soluzione trovata sia quella ottima.

Infine verrà effettuato un confronto tra i risultati ottenuti da diverse MLP che si differenzieranno per una diversa configurazione degli hidden layers.

Introduzione all'algoritmo MiniMax

Anche se l'argomento principale del progetto sono le reti neurali, di tipo MLP nello specifico, in questa sezione è presente una piccola digressione riguardo l'algoritmo MiniMax.

Questo algoritmo effettua una ricerca esplorando un albero creato in maniera ricorsiva e scegliendo la soluzione migliore presente.

In quest'albero ogni nodo rappresenta una configurazione della griglia (in generale lo stato del gioco). Partendo dalla radice (la situazione attuale) vengono aggiunti tanti nodi figli quante possibili mosse ed in ogni figlio viene applicata una differente mossa garantendo in questo modo la completezza della ricerca.

Questo procedimento viene effettuato ricorsivamente fino a quando in tutte le griglie presenti nei nodi foglia sono terminali (uno dei due giocatori ha vinto o la griglia è piena).

Quando la costruzione dell'albero è terminata ad ogni nodo foglia viene attribuito un valore in base alla situazione finale (generalmente vittoria ≥ pareggio ≥ sconfitta).

A questo punto si risale l'albero considerando che il giocatore che effettuare la mossa vuole avere il guadagno massimo mentre l'avversario lo voglia ridurre il più possibile.

Di conseguenza ad ogni nodo intermedio viene attribuito il valore massimo dei figli se appartiene al livello della mossa del giocatore di turno altrimenti, nel caso si tratti del livello di una mossa dell'avversario, si sceglie il valore minimo.

Ulteriori informazioni sull'algoritmo MiniMax sono riportate in [1].

Stato dell'arte

Gli aspetti principali della progettazione di una rete MLP sono:

1) Come rappresentare l'informazione
2) Quale deve essere la struttura della rete MLP
3) Come adattare i pesi dei collegamenti neurali

Per il primo punto si può far riferimento a [2] in cui una rete neurale è stata creata, tramite la programmazione evolutiva, una rete neurale in grado di giocare a tic-tac-toe autonomamente. Ovvero la rete ha 9 neuroni in ingresso in cui si attribuisce 1 se è presente O, 0 se la cella è vuota e -1 se è presente una X; inoltre sono presenti 9 neuroni di output ad ognuno dei quali corrisponde una cella.

Per quanto riguarda la struttura della rete MLP (il numero di livelli e quello di neuroni per livello) bisogna tenere presente gli studi effettuati in [3] in cui si dimostra che un numero eccessivo di neuroni presenti nel livello intermedio non apporta benefici essendo questi ridondanti tra loro. Inoltre all'aumentare del numero di neuroni aumenta il numero di pesi e quindi i tempi di propagazione (sia in avanti che all'indietro).

Per quanto riguarda l'apprendimento della rete in [4] sono riportate diverse tecniche di diversa complessità e con efficienze diverse tra loro anche in base al problema è sottoposto la rete neurale. Le tecniche utilizzate in questo progetto sono il quickprop e il backPropagation che verranno esposte nella sezione successiva.

Modellazione del problema

Le principali strutture da implementare in questo progetto sono la rete neurale e l'albero relativo all'algoritmo MiniMax in cui memorizzare tutte le soluzioni possibili e per ogni griglia la migliore mossa.

Albero delle soluzioni

I nodi dell'albero delle soluzioni sono formati da una struttura dati che comprende la griglia e il valore attribuito al nodo dall'algoritmo MiniMax.

I figli di un determinato nodo sono le possibili evoluzioni della griglia a partire dalla configurazione presente nel nodo stesso.

La radice sarà la griglia vuota mentre le foglie saranno le configurazioni terminali (cioè le griglie piene o in cui uno dei due giocatori ha vinto).

Dato che ogni partita verrà iniziata con il simbolo "X" è sufficiente creare un solo albero. Se invece si scegliesse di invertire il simbolo iniziale e non il giocatore a cui appartiene bisognerebbe creare due alberi in cui la sequenza dei simboli è opposta.

Rete neurale

La rete neurale implementata è di tipo MLP. Come anticipato ha 9 neuroni di input e 9 di output.

Il numero di livelli hidden e quello dei neuroni presenti all'interno di ogni hidden layer sono degli iperparamentri del problema che verranno letti dal programma attraverso un file di configurazione.

La funzione di attivazione scelta per la rete neurale è la *"standard logistic function"*, la stessa utilizzata in [2].

Quando la rete neurale deve effettuare una mossa prima controlla se è possibile vincere con una sola mossa, in caso contrario acquisisce nel livello di input la griglia e restituisce come mossa successiva la cella libera corrispondente al neurone di output con il valore più alto (in questo modo non ci saranno mosse illegali).

Apprendimento della rete neurale

Tra i vari metodi di apprendimento proposti in [4] sono stati implementati sia il backPropagation che il quickprop (descritto originariamente in [5]).

Considerando la loss function

$$E = \frac{1}{2} \sum_{z \in output} (t_z - s_z)^2$$

la differenza tra questi due metodi è il modo in cui il differenziale di *E* rispetto ai pesi influenza la variazione dei pesi partendo dallo stesso differenziale.

Nel backPropagation i pesi sono modificati in modo tale da muoversi in direzione opposta rispetto al gradiente (per poter minimizzare la funzione) mentre per il quickprop si assume che la funzione degli errori (E) sia, localmente, una parabola rivolta verso l'alto o più genericamente una funzione tale che la sua derivata prima sia convessa; l'aggiornamento dei pesi nel quickprop è equivalente all'applicazione del metodo di approssimazione di Newton. In formule si può notare meglio la differenza:

BackPropagation:

$$\Delta w_{ij}(t) = \frac{\partial E}{\partial w_{ij}}(t)\eta$$

Quickprop:

$$\Delta w_{ij}(t) = \frac{\frac{\partial E}{\partial w_{ij}}(t)}{\frac{\partial E}{\partial w_{ij}}(t-1) - \frac{\partial E}{\partial w_{ij}}(t)} \Delta w_{ij}(t-1)\eta$$

dove η è il *learning rate*.

Per ricavare la formula del quickprop è stata effettuata un'approssimazione evitando di calcolare la derivata seconda della funzione degli errori[1]. Nel caso *t=0*, per semplicità, si assumono i parametri relativi alla variazione precedente pari ad 1.

Inoltre definendo

$$\delta_j := \begin{cases} (t_j - s_j)s_j(1 - s_j) & j \in output \\ \left(\sum_{k \in succ(j)} w_{jk}\delta_k\right) s_j(1 - s_j) & j \notin output \end{cases}$$

si può scrivere

$$\frac{\partial E}{\partial w_{ij}} = -\delta_j \cdot s_i$$

con w_{ij} peso qualsiasi che collega i neuroni *i* e *j* (anche se *i* è un bias).

Il metodo di apprendimento e il learning rate vengono definiti nel file di configurazione.

È stato preferito applicare il metodo *"learning by patterns"* rispetto al *"learning by epoch"* in modo tale da poter aggiornare più frequentemente i pesi della rete neurale e quindi avere una convergenza più rapida dei pesi.

[1] Per ulteriori approfondimenti riguardo i passaggi matematici relativi al quickprop si può consultare [4].

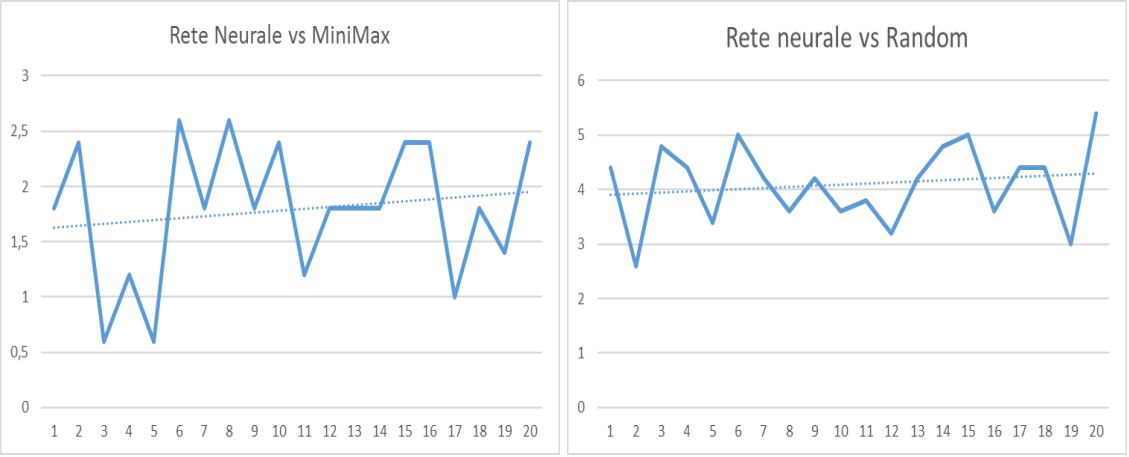
Test e analisi dei risultati

Il test è stato effettuato facendo sfidare la rete neurale con un giocatore casuale e con un giocatore che basa le sue mosse sull'albero MiniMax.

Tra i parametri sono presenti sia il numero di epoche che la frequenza con la quale effettuare i test.

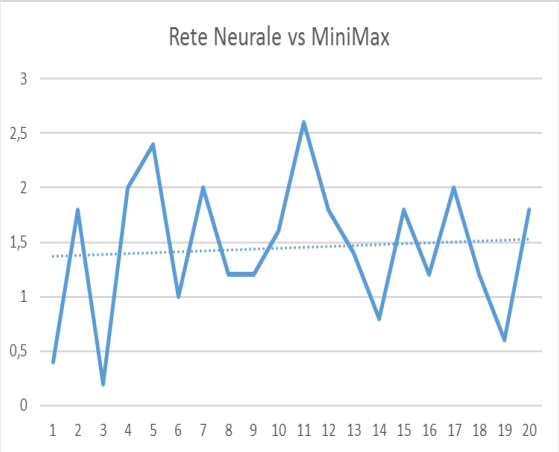
Nei test effettuati si giocano due partite in modo tale che entrambi i giocatori effettuino una partita in cui giocano la prima mossa. Alla vittoria vengono attribuiti 3 punti, al pareggio 1 e alla sconfitta nessuno.

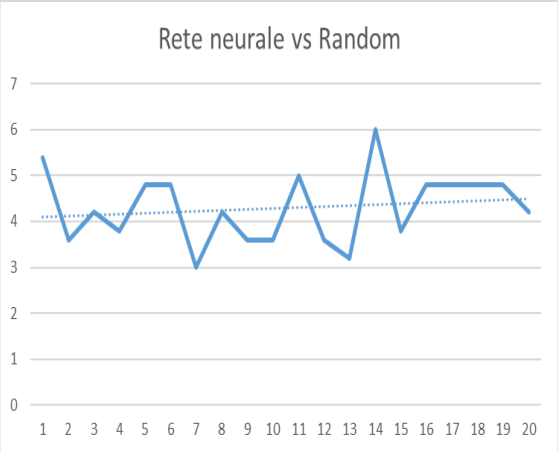
Di seguito sono riportati i grafici relativi alla media di test effettuati con 20 epoche e verifica dei risultati dopo ogni epoca, con la configurazione di neuroni 9-10-9 e con η= 0.00001 ottenuti applicando come metodo di apprendimento il quickprop:



Si può notare come la linea di tendenza sia crescente poiché la rete, imparando, migliora il suo punteggio. L'andamento oscillante dei risultati è dovuto alla natura stocastica del problema: il MiniMax sceglie una mossa random tra quelle per le quali prevede lo stesso risultato finale mentre il Random sceglie stocasticamente una cella libera in cui posizionare il proprio simbolo.

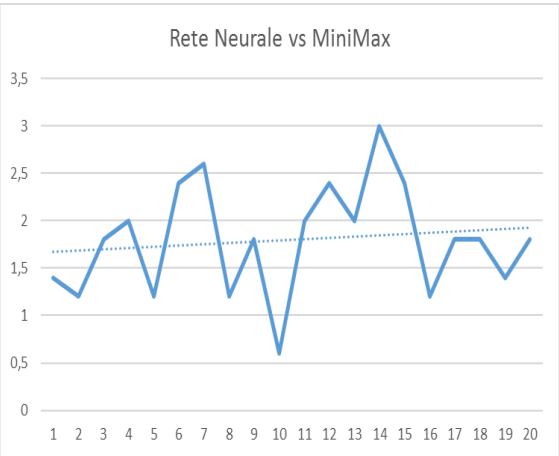
Di seguito sono riportati i risultati dei test effettuati impostando gli stessi iperparametri ma settando come metodo di apprendimento il backPropagation:

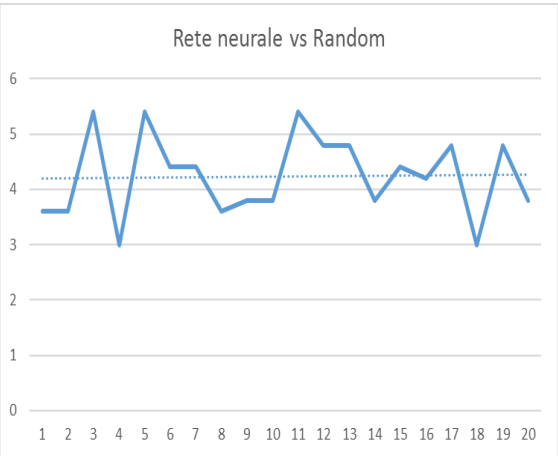




Si può notare come in questo caso la linea di tendenza sia più orizzontale rispetto a quella del quickprop perché tramite il backPropagation l'apprendimento è più lento nel tempo.

Un altro test importante è quello di confrontare i risultati dell'algoritmo al variare del numero di livelli di hidden. Nel caso seguente la configurazione della rete è 9-5-5-9, il metodo di apprendimento è il quickprop e il learning rate è di 0,00001.





Anche in questo caso si può notare un incremento delle prestazioni più lento rispetto alla configurazione 9-10-9 nonostante in entrambi i casi si usi il quickprop. Ciò avviene perché l'aggiornamento dei pesi è più lento perché il maggior numero di livelli rallenta la propagazione dell'apprendimento.

Conclusioni e sviluppi futuri

A causa della poca linearità della funzione degli errori e dell'alto numero di parametri la rete neurale non riesce ad apprendere totalmente le strategie del gioco ma riesce comunque a giocare meglio di un giocatore che effettua mosse casuali (anche omettendo la previsione della vincita in una mossa) e ottiene un terzo dei punti del giocatore MiniMax.

Inoltre, avendo utilizzato un metodo di *gradient descent* per l'apprendimento, la soluzione si bloccherà sempre in un punto di minimo locale che potrebbe essere parecchio differente dal punto di minimo globale.

Uno sviluppo futuro potrebbe essere la trasformazione della rete di tipo MLP in una di tipo CNN in cui applicare diversi filtri di convoluzione alla griglia per riuscire a capire autonomamente particolari configurazioni dei simboli ed acquisire strategie intelligenti.

Bibliografia

[1] P. Borovska, M. Lazarova, “Efficiency of parallel minimax algorithm for game tree search”, *Proceedings of the 2007 international conference on Computer systems and technologies, June 14-15, 2007, Bulgaria.*

[2] D. Fogel, "Using evolutionary programming to create neural networks that are capable of playing Tic Tac Toe", *IEEE International Conference on Neural Networks (ICNN), 1993.*

[3] R. Brad, I. Mihu, and M. Breazu. "Statistical analysis of multilayer perceptrons performances." *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on. Vol. 4. IEEE, 2001.*

[4] M. Riedmiller “Advanced Supervised Learning in Multi-layer Perceptrons – From Backpropagation to Adaptive Learning Algorithms.” *Int Journal of Computer Standards and Interfaces. 1994, 16: pp. 265-278. [Special Issue on Neural Networks].*

[5] S. E. Fahlman, "Faster-learning variations on back-propagation: An empirical study." *Proceedings of the 1988 Connectionist Summer School, 1988.*