Corso di Intelligenza Artificiale

Addestrare una rete neurale a giocare a tris partendo da un dataset generato tramite l'algoritmo minimax

Questa tesina, realizzata per il corso di Intelligenza Artificiale presso il Politecnico di Torino, ha lo scopo di verificare se una rete neurale MLP (con un solo strato di nodi hidden) sia in grado di apprendere il gioco del filetto (tris).

Il progetto è stato realizzato sfruttando principalmente due linguaggi, Python e MATLAB, e si divide in 2 fasi:

- la prima fase, realizzata dallo script Python "*ttt_minimax.py*", consiste nella generazione del dataset con il quale verrà addestrata la rete neurale tramite l'algoritmo minimax
- la seconda fase, realizzata dagli script MATLAB "*TicTacToe_NN.m*", "*NN_Train_Test.m*" e "*NeuralNetwork.m*", consiste nell'addestramento di una rete MLP e nel suo testing.

Script Python: "*ttt_minimax.py*"

Lo script Python ha lo scopo di generare i dati con cui andare concretamente ad addestrare la rete.

Se considerassimo tutte le possibili configurazioni della griglia di gioco avremmo 3^9 (19.683) configurazioni per giocatore per un totale di 39.366 combinazioni differenti da utilizzare per addestrare la rete.

Siccome il nostro intento è addestrare la rete a rispondere in maniera efficace alla mossa del giocatore avversario, ho ridotto di molto il numero dei dati di input rimuovendo tutte quelle configurazioni che dal punto di vista del gioco non sono valide. Una configurazione è considerata valida solamente se il numero di simboli "***X***" è uguale al numero di simboli "***O***" o differisce al più di un'unità. Inoltre, non è considerata valida una situazione in cui vi sono più "***X***" che "***O***" ed è il turno del giocatore "***X***" (e viceversa).

Con questo accorgimento il numero di configurazioni valide cala drasticamente passando da 39.366 a 9.072.

Una volta generate tutte le novemila configurazioni lo script prepara i file utili all'addestramento della rete neurale. L'addestramento può avvenire in 2 modi diversi a seconda che il flag FOR_MATLAB venga settato a False o a True:

- nella prima modalità, il file viene diviso in due data set (quello di training e quello di testing) memorizzati su due file distinti: "*nn_training.txt*" e "*nn_testing.txt*". La divisione tra i 2 file viene effettuata a run-time, in modo pseudo-casuale, secondo il valore impostato nella variabile statica "*T_T_PERC*". Per poter avere 2 dataset circa equivalenti ma favorire leggermente di più la fase di training ho scelto di dividere le configurazioni tra i 2 file utilizzando il 60% delle entry per il file di training.

- nella seconda modalità, vengono separati gli input dai target (gli output desiderati) e salvati rispettivamente nei file "*nn_input.txt*" e "*nn_output.txt*". La suddivisione dei dati nei 3 dataset di training, validating e testing viene lasciata in carico allo script MATLAB.

Script MATLAB: "*TicTacToe-NN.m*"

Questo script legge i dati di input, imposta la suddivisione percentuale tra il training set, il validating set e il testing set e richiama lo script "*NN_Train_Test.m*". Questo script è stato realizzato per permettere all'utente di effettuare vari test in maniera automatica impostando differenti percentuali di suddivisione dei dati.

Dopo aver effettuato alcuni test ho lasciato all'interno dello script le configurazioni che ritenevo più significative. In particolare tra le configurazioni troviamo:

- [60, 0, 40]: la configurazione che più si avvicina alla proposta iniziale di suddividere i dati in due dataset di dimensione circa uguale. Come nello script Python ho scelto di favorire leggermente il training set per migliorare le prestazioni della rete.
- [50, 10, 40]: ripartizione scelta seguendo lo stesso ragionamento della configurazione precedente; anche se non originariamente previsto ho riservato alcuni dati per il validation set per verificare se questa ulteriore fase avrebbe dato o meno beneficio all'addestramento della rete.
- [70, 15, 15]: la configurazione suggerita dal tool per l'implementazione delle reti neurali di MATLAB.
- [90, 0, 10]: ho cercato di ritornare alla configurazione originale (la quale non prevedeva un validating set) avvantaggiando però molto la fase di training. L'ho ritenuto un esperimento interessante in quanto, avendo a disposizione tutte le possibili configurazioni di gioco, incorrere nel fenomeno dell'overfitting non dovrebbe essere svantaggioso per la rete.

Script MATLAB: "*NN_Train_Test.m*"

Questo script richiama la funzione contenuta in "*NeuralNetwork.m*" per addestrare e testare la rete.
Al termine di ogni chiamata alla procedura vengono stampati il numero di configurazioni correttamente classificate e la percentuale di corretti classificati.

Sebbene, come già anticipato nel paragrafo precedente, in questa particolare applicazione, non ritengo che l'overfitting possa essere un problema particolarmente significativo in quanto la rete non si troverà mai a dover rispondere a combinazioni diverse da quelle presenti nel data set con cui è stata addestrata, non ho effettuato test con un numero superiore ai 1500 nodi hidden in quanto il mio computer non è stato in grado di portare a termine con successo l'elaborazione.

Dopo aver effettuato vari test ho lasciato all'interno dello script alcune delle configurazioni con i migliori risultati ottenuti (i risultati di questa elaborazione sono visibili nel paragrafo "Risultati").

Script MATLAB: "*NeuralNetwork.m*"

Questo script, richiamabile attraverso l'omonima funzione, si occupa di inizializzare, addestrare e testare la rete neurale. La rete MLP viene inizializzata attraverso il comando "*feedforwardnet*" al quale viene passato il numero di nodi che si desidera utilizzare nello strato hidden.
Non specificando altri parametri vengono utilizzati quelli di default, che sono:

- due strati di elaborazione (1 strato hidden + lo strato di output)
- 1 solo neurone di output (il quale conterrà un valore compreso tra 0 e 8, ovvero l'identificativo della cella in cui il giocatore dovrebbe posizionare il suo simbolo)
- Algoritmo di addestramento Levenberg-Marquardt con backpropagation
- Funzione di trasferimento ai nodi hidden: sigmoide (tansig)
- Funzione di trasferimento ai nodi di output: lineare (purelin)

Una volta inizializzati i pesi e i bias e settate le percentuali di suddivisione dei dati nei 3 set (training, validating e testing), la rete viene addestrata e testata. Opzionalmente è anche possibile stampare a schermo grafici contenenti varie informazioni come le performance e la matrice di confusione.

La funzione ritorna al chiamante i valori di output ottenuti dalla rete (arrotondati all'intero più vicino), l'errore (ottenuto come differenza in valore assoluto tra i target forniti per l'addestramento e gli output ottenuti), le performance della rete ed il numero di elementi correttamente classificati.

Risultati

Come si può vedere dai seguenti dati generalmente le performance della rete aumentano all'aumentare del numero di nodi di hidden e all'aumentare della percentuale di dati riservati al processo di training. Nonostante un numero elevato di nodi interni e la ripartizione pesantemente squilibrata dei valori tra il training ed il testing set possa portare la rete al fenomeno dell'overfitting (con una conseguente scarsità di generalizzazione), in questo caso ritengo che il gioco possa valere la candela in quanto, come già anticipato, ci ritroviamo di fronte ad un problema molto ben delineato per il quale il verificarsi di configurazioni diverse da quelle previste è impossibile.

Train : Valid : Test	10 nodi	50 nodi	100 nodi	120 nodi	500 nodi	1000 nodi	1100 nodi
60:0:40	20,39%	30,89%	35,58%	37,20%	65,80%	66,38%	66,66%
50:10:40	21,81%	29,43%	30,93%	32,26%	27,83%	37,15%	55,34%
70:15:15	20,70%	31,88%	33,85%	35,07%	35,37%	73,88%	63,24%
90:0:10	20,46%	32,91%	37,90%	40,64%	85,13%	91,90%	92,15%