Alma Mater Studiorum · Università di Bologna

Scuola di Scienze
Dipartimento di Fisica e Astronomia
Corso di Laurea in Fisica

TITOLO TESI

Relatore:
Prof./Dott. Enrico Giampieri

Correlatore: (eventuale)
Prof./Dott. Nome Cognome

Presentata da:
Mattia Ceccarelli

Anno Accademico 2017/2018

Indice

Capitolo 1

Introduzione

In questo capitolo si introdurranno i principali mezzi utilizzati nello svolgimento del progetto di tesi, ossia Algoritmi Genetici per la ricerca di minimi per una funzione ???????? e Reti Neurali fully connected, che svolgono il ruolo di funzione a molti parametri da ottimizare in un problema di classificazione.

1.1 Algoritmi Genetici

Gli algoritmi genetici sono software di ricerca ispirati dalla selezione naturale applicata ad una popolazione di individui, chiamati soluzioni, caratterizzati da un *cromosoma*, spesso rappresentato da una lista di numeri binari o da una stringa. Il parametro che differenzia soluzioni migliori o peggiori è il *fitness*, misurato attraverso la *funzione di fitness* la quale dipende dal problema. L' evoluzione della popolazione avviene attraverso la selezione dei migliori individui che passeranno il loro *cromosoma* alla generazione successiva.

1.1.1 Operatori

Mitchell [1999] I principali operatori che compongo un semplice algoritmo genetico sono:

Selezione Questo operatore seleziona i migliori individui, più è alto è il fitness e più è probabile che un individuo venga scelto per creare la nuova generazione

Crossover L'operatore di Crossover produce un taglio nel genoma degli individui "genitori" per formare due individui "figli": per esempio prendendo le due stringhe 111000 e 000111, producendo un taglio alla terza posizione otterremo le stringhe 111111 e 000000.

Mutazione L'operatore di mutazione si occupa di cambiare casualmente uno o più caratteri di individui scelti a caso nella popolazione.

Il funzionamento di un tipico algoritmo genetico,come descritto da Mitchell [1999] una volta definito il problema, procede in questo modo:

1.1.2 Struttura di un Algoritmo Genetico

1. Creazione casuale di n elementi, che rappresentano la prima popolazione.
2. Calcolo del fitness $f(x)$ di ogni soluzione x della popolazione.
3. Fino a che non sono stati generati n discendenti ripetere:
    a. Selezione di due genitori dalla popolazione dove un individuo può anche essere scelto più volte.
    b. Con probabilità p_c (probabilità di crossover) applicare l'operatore di crossover sui due genitori. Nel caso non avvenisse alcun crossover, copiare i genitori.
    c. Con probabilità p_m (probabilità di mutazione) applicare l'operatore di mutazione sui figli.
4. Sostituire la vecchia popolazione con la nuova generazione e ripetere dal secondo passaggio.

Ogni iterazione di questo processo è chiamata *generazione*.

1.1.3 Applicazioni

Il classico esempio di utilizzo di un algoritmo genetico è la ricerca dei massimi di una funzione. In tal caso, un individuo è rappresentato da una stringa di bit, la *funzione di fitness* è la funzione stessa e il *fitness* delle soluzioni è il valore della funzione calcolato nel punto di cui l'individuo è la rappresentazione binaria. Oltre ad essere l'esempio più semplice risulta anche quello più significativo: di fatto lo scopo di un algoritmo genetico è ottimizzare.
Da migliorare

1.2 Reti Neurali

Una rete neurale è una struttura interconnessa di semplici unità procedurali, chiamate nodi. La loro funzionalità si ispira ai neuroni del regno animale. La capacità di elaborazione della rete neurale è contenuta nella "forza" delle connessioni tra nodi, espressa dai *pesi* dei collegamenti, ottenuti da processi di *addestramento* o *apprendimento*. Gurney [1997]

1.2.1 Il perceptron

Nielsen [2015] Il perceptron è stato sviluppato negli anni '50 e '60 dal ricercatore Frank Rosenblatt ispirandosi ai lavori antecedenti di Warren McCulloch e Walter Pitts. È l'unità di base di una rete neurale e il suo funzionamento è il seguente: il perceptron riceve n valori in ingresso $x_1, x_2, ..., x_n$ e restituisce 1 o 0 a seconda che la somma pesata degli input superi o no un valore di soglia, con pesi $w_1, w_2, ..., w_n$. Ad esempio nel perceptron mostrato in figura 1.1:

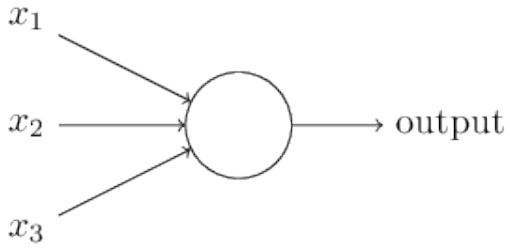

Figura 1.1: *Perceptron con 3 input ed un output*

l'output sarà determinato da:

$$\begin{cases} 0 \text{ se } \sum_i x_i w_i \leq \textit{valore di soglia} \\ 1 \text{ se } \sum_i x_i w_i > \textit{valore di soglia} \end{cases}$$

anche se è più comune trovare la scrittura:

$$\begin{cases} 0 \text{ se } \sum_i x_i w_i + b \leq 0 \\ 1 \text{ se } \sum_i x_i w_i + b > 0 \end{cases}$$

dove b è detto *bias* del perceptron. È attraverso *pesi* e *bias* che il perceptron può soppesare diverse prove e compiere decisioni.

Tuttavia se la rete contenesse perceptron, anche un piccolo cambiamento nei parametri interni potrebbe causare un cambiamento netto nel comportamento della rete [Nielsen [2015]], per questo è preferibile utilizzare una *funzione di attivazione* che rende continuo l'output di un nodo. Un esempio di funzione di attivazione è la sigmoide definita come:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{1.1}$$

e l'output di un nodo della rete diventa :

$$y = \sigma(\sum_i x_i w_i + b) \tag{1.2}$$

risultato che è continuo e compreso tra zero ed uno.

Un altro tipo di funzione di attivazione è la *Rectified Linear Units* o *ReLU* e si presenta come:

$$y(x) = max\{0, x\} \tag{1.3}$$

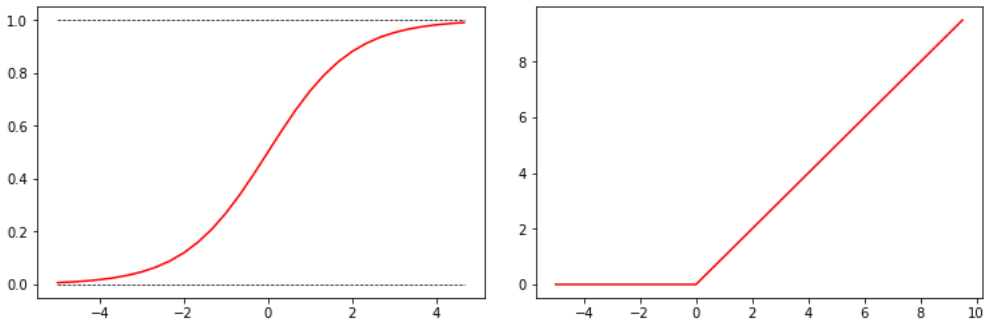

Figura 1.2: *confronto tra due funzioni di attivazione: a sinistra sigmoidale e a destra ReLU*

La scelta della migliore funzione di attivazione non è univoca e dipende dal problema che viene affrontato.

1.2.2 Struttura *fully connected*

La struttura di una rete neurale *fully connected* composta da molti *layer* di neuroni è come quella mostrata in figura 1.3:

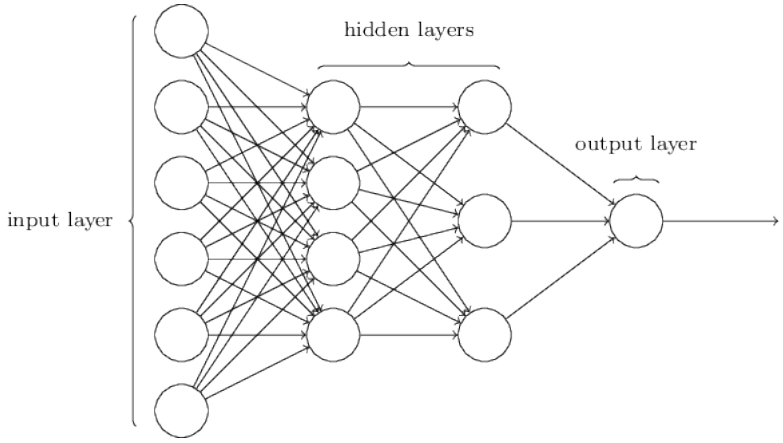

Figura 1.3: *Esempio di rete neurale fully connected in cui viene mostrata la distinzione tra input layer, hidden layer e output layer*

In una rete come questa ad ogni collegamento è associato un peso e ad ogni nodo è associato un *bias*: gli output dei *neuroni* del *layer* di input diventano a loro volta valori in ingresso del *layer* successivo in un procedimento a catena fino all'ultimo *layer*, che restituisce la risposta della rete. L'addestramento della rete consiste nel valutarne gli output in un determinato set di dati, chiamato *training dataset*, confrontarli con i valori attesi, forniti dallo stesso *dataset*, e modificare pesi e *bias* in modo che la risposta si avvicini a ciò che ci si aspetta.

Da completare con algoritmo di BackPropagation???

1.2.3 Evoluzione di una Rete Neurale

Capitolo 2

Metodologia

Capitolo 3

Risultati

Capitolo 4

Conclusioni

Elenco delle figure

Bibliografia

Kevin Gurney. *An Introduction to Neural Networks*. UCL Press, 1997. ISBN 0-203-45151-1.

J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

Melanie Mitchell. *An Introduction to genetic algorithm*. The MIT Press, 1999. ISBN 0262133164.

Michael A. Nielsen. *Neural Network and Deep Learning*. Determination Press, 2015.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.