

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**

**Presentata da:**

**Mattia Ceccarelli**

**Correlatore: (eventuale)**

**Prof./Dott. Nome Cognome**

Anno Accademico 2017/2018

# Indice

<b>1</b>	<b>Introduzione</b>	<b>2</b>
1.1	Algoritmi Genetici . . . . .	2
1.1.1	Operatori . . . . .	2
1.1.2	Struttura di un Algoritmo Genetico . . . . .	3
1.1.3	Applicazioni . . . . .	3
1.2	Reti Neurali . . . . .	3
1.2.1	Il perceptron . . . . .	4
1.2.2	Struttura <i>fully connected</i> . . . . .	5
1.2.3	Evoluzione di una Rete Neurale . . . . .	6
<b>2</b>	<b>Metodologia</b>	<b>7</b>
2.1	La popolazione di reti neurali . . . . .	7
2.2	Crossover . . . . .	7
2.3	Mutazione . . . . .	8
2.4	Selezione . . . . .	8
2.5	Funzione di Fitness . . . . .	8
<b>3</b>	<b>Risultati</b>	<b>9</b>
<b>4</b>	<b>Conclusioni</b>	<b>10</b>
	<b>Bibliografia</b>	<b>11</b>

# 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 ottimizzare 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 compongono 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  $x_1, x_2, \dots, x_n$  e restituisce 1 o 0 a seconda che la somma pesata degli input superi o no un valore di soglia, con pesi  $w_1, w_2, \dots, w_n$ . Ad esempio nel perceptron mostrato in figura 1.1:

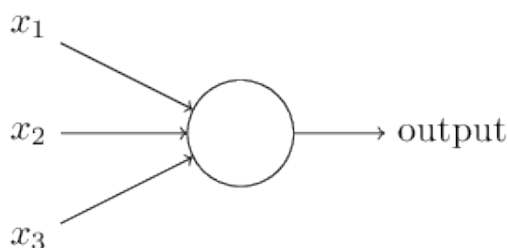


Figura 1.1: *Perceptron con 3 input ed un output*

l'output sarà determinato da:

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

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:

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

e l'output di un nodo della rete diventa :

$$y = \sigma\left(\sum_i x_i w_i + b\right) \quad (1.2)$$

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:

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

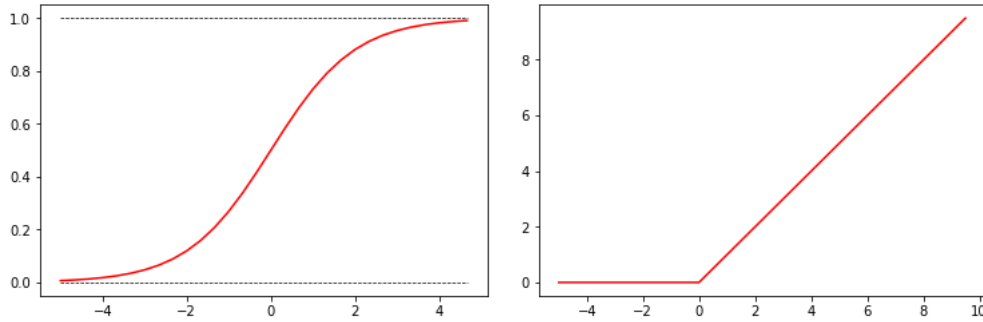


Figura 1.2: confronto tra due funzioni di attivazione: a sinistra sigmoideale e a destra *ReLU*

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:

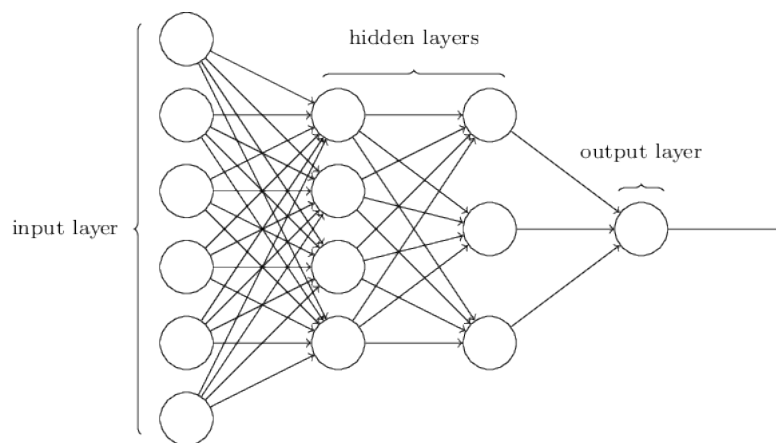


Figura 1.3: Esempio di rete neurale *fully connected* in cui viene mostrata la distinzione tra *input layer*, *hidden layer* e *output layer*

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 accenno a BackPropagation

### 1.2.3 Evoluzione di una Rete Neurale

# Capitolo 2

## Metodologia

In questo capitolo verranno descritti i metodi utilizzati per l'ottimizzazione di una rete neurale *fully connected* attraverso un algoritmo genetico: l'obiettivo è quello di evolvere la struttura degli *hidden layer* della rete al fine di classificare al meglio un dataset separato in due classi di dati.

### 2.1 La popolazione di reti neurali

L'algoritmo genera la prima popolazione di oggetti *Random Network* casualmente, i parametri interni sono stati scelti in modo che questa fosse limitata e non raggiungesse subito un risultato ottimale. In particolare il numero di layer è compreso tra 1 e 5 mentre il numero massimo di neuroni per layer è 25. La popolazione è stata mantenuta al massimo di 20 individui per lo stesso motivo. Il *cromosoma* di un individuo è quindi una lista di numeri di lunghezza variabile e l'algoritmo genetico è stato adattato a questo problema: la ricerca di un minimo in uno spazio in cui il numero di variabili non è definito.

### 2.2 Crossover

Il Crossover si differenzia da quello di un algoritmo genetico in cui la lunghezza di ogni cromosoma sia prestabilita. Infatti, è facile che i due genitori selezionati non abbiano lo stesso numero di hidden layer: in questo caso non è possibile "tagliare" il cromosoma nello stesso punto (come mostrato nel capitolo 1.1). Inoltre è necessario che la lunghezza dei figli possa essere diversa da quella dei genitori in modo da non limitare le possibilità di esplorazione.

Seguendo questi propositi, vengono scelti casualmente due punti di taglio, uno per genitore, da una media di tre interi casuali (questo per fare in modo che la lunghezza non cambi troppo tra una generazione e la successiva). Dopodiché il crossover procede



come in un classico algoritmo genetico, ossia restituendo due figli combinazione delle parti tagliate.

Un esempio di funzionamento :

genitore 1 = [10,3,4,11]; genitore 2 = [7,23,1,2,19]

prendendo due tagli alla terza e seconda posizione si ottengono:

figlio 1 = [10,3,4,1,2,19]; figlio 2 = [7,23,11]

i quali popoleranno la generazione successiva.

## **2.3 Mutazione**

## **2.4 Selezione**

## **2.5 Funzione di Fitness**

## Capitolo 3

### Risultati

Capitolo 4

Conclusioni

# Elenco delle figure

1.1	<i>Perceptron con 3 input ed un output</i>	4
1.2	<i>confronto tra due funzioni di attivazione: a sinistra sigmoideale e a destra ReLU</i>	5
1.3	<i>Esempio di rete neurale fully connected in cui viene mostrata la distinzione tra input layer, hidden layer e output layer</i>	5

# 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.