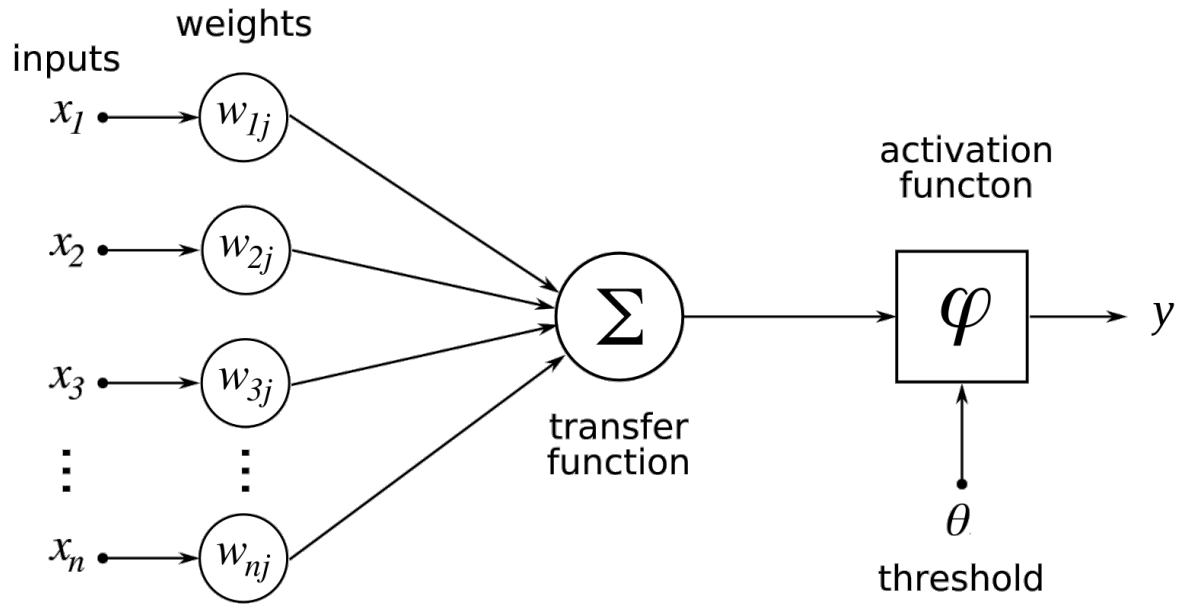
MyNN: la mia rete neurale

Specifiche v1.0

Esempio di implementazione in Java delle specifiche di progetto per la realizzazione di una rete neurale base, a singolo strato e multistrato con algoritmo di backtracking.

> *Le Reti neurali artificiali sono modelli matematici che rappresentano l'interconnessione tra elementi definiti neuroni artificiali, ossia costrutti matematici che in qualche misura imitano le proprietà dei neuroni viventi. Questi modelli matematici possono essere utilizzati sia per ottenere una comprensione delle reti neurali biologiche, ma ancor di più per risolvere problemi ingegneristici di intelligenza artificiale come quelli che si pongono in diversi ambiti tecnologici (in elettronica, informatica, simulazione, e altre discipline).* https://it.wikipedia.org/wiki/Rete_neurale

Neurone. Un neurone prende in input n valori reali $x_1, \ldots, x_n$ e calcola una combinazione lineare di questi (ovvero, una somma pesata) il cui risultato viene dato in input a una funzione di attivazione, che restituisce l'output del neurone. Ad esempio

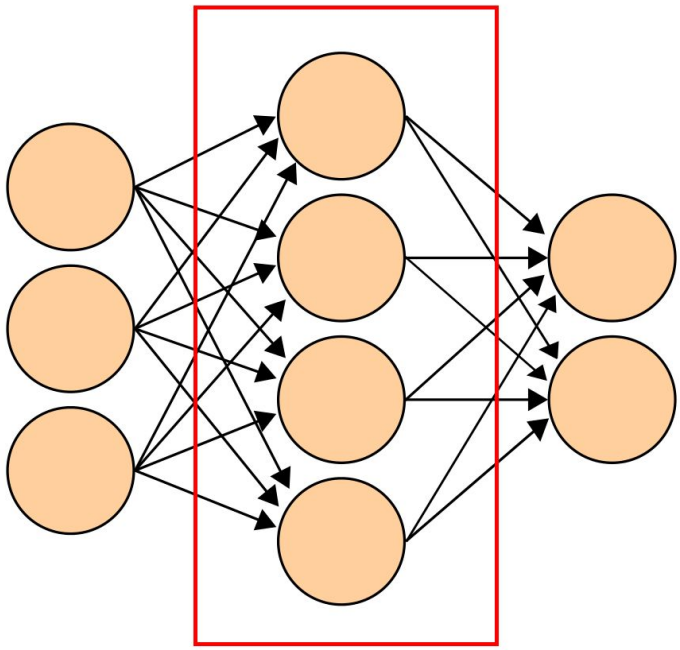


viene calcolata la funzione:

$$y = f\left(\sum_i w_i * x_i + \theta\right)$$

dove w_i sono i pesi associati a x_i, ovvero i pesi delle connessioni in input e f è la funzione di attivazione.

Strato di neuroni. Si definisce uno strato di neuroni S come un insieme di neuroni { N_1, …, $N_{|S|}$ } che prendono lo stesso input e utilizzano tutti la medesima funzione di attivazione:



Rete neurale. Infine si definisce una rete neurale NN come una serie di strati [S_1, …, $S_{|NN|}$], dove l'input iniziale della rete viene fornito a tutti i neuroni dello strato S_1, l'output dei neuroni di S_1 viene fornito in input ai neuroni dello strato S_2 e così via fino a $S_{|NN|}$.

In generale, si può affermare che un neurone è già una rete neurale basilare con un solo strato e un solo neurone. Per una ragione analoga, uno strato è anch'esso una rete neurale.

Assumendo che i pesi dei vari neuroni siano correttamente impostati, l'obiettivo di una rete neurale è quello di classificare l'input, codificato sotto forma di un vettore (x_1, x_2, …, x_n), mediante un vettore di output (y_1, y_2, …, y_m). Ad esempio, immaginiamo che il vettore in input sia la linearizzazione di un'immagine, ovvero la sequenza dei pixel della stessa. La rete può calcolare la probabilità che l'immagine raffiguri un gatto (y_1), un cane (y_2), uno studente di Java (y_3), ecc. Dato in input il vettore dell'immagine, la rete calcola gli output dei neuroni del primo strato, che viene fornito ai neuroni del secondo strato e così via fino ad ottenere l'output dei neuroni dell'ultimo strato. I pesi, se correttamente impostati, permetteranno di ottenere un valore elevato per y_1 laddove la figura in input contenga un gatto e così via.

Una rete neurale dispone dei seguenti metodi:

- **toString** mostra la rete nel formato illustrato sotto.
- **process** che riceve una lista di valori in input e restituisce l'output calcolato su di essi
- **trainIstanza** riceve una lista di valori in input e una lista dei valori attesi di output, ovvero i valori che la rete idealmente dovrebbe restituire a fronte di quell'input. Il metodo, modifica i pesi in accordo con la **formula di aggiornamento** e restituisce la somma degli errori ottenuti dal confronto tra l'output effettivo della rete e quello ideale passato in input (si veda sotto per una spiegazione del processo di addestramento).
- **train** riceve un insieme di addestramento e fa addestramento in accordo al **algoritmo d'addestramento** spiegato sotto.

Le funzioni di attivazione utilizzabili sono le seguenti (si fornisce anche la funzione derivata f' per l'addestramento con il metodo train, si veda sotto):

- Step

$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$$

- Identity

$$f(x) = x$$
$$f'(x) = 1$$

- Logistic

$$f(x) = \frac{1}{1 + e^{-x}}$$
$$f'(x) = f(x)(1 - f(x))$$

- Hyperbolic Tangent (TanH)

$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

$$f'(x) = 1 - f(x)^2$$

- Rectified Linear Unit (ReLU)

$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$$

$$f'(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$$

Formato del file di codifica della rete neurale

Possibilità di codificare la struttura della rete da file (utilizzando la reflection per istanziare i neuroni) in un formato compatibile con quello dell'esempio seguente:

```
nome=Rete1
layer={ nome=input activationFunction=TanH inputUnits=10
outputUnits=5 }
layer={ nome=output activationFunction=Identity inputUnits=5
outputUnits=1 }
ecc.
```

I layer (ovvero gli strati) sono forniti in ordine di elaborazione, dal primo all'ultimo. In fase di caricamento è necessario controllare che il numero di unità in output di uno strato corrisponda al numero di unità in input allo strato successivo. Il file può contenere anche i valori iniziali dei pesi w delle connessioni neurali. **L'ultimo peso della lista di pesi `weights` sarà il valore del threshold.** Ad esempio:

```
layer={ nome=input activationFunction=TanH inputUnits=5
outputUnits=2
weights=[[1,0.5,0.223,0,0.89,0.0],[0.2,0.3,0.4,0.5,0.6,0.0]]
}
```

imposta come pesi 1, 0.5, …, 0.89 per i 5 input del primo neurone dello strato e 0.2, 0.3, …, 0.6 per i 5 input del secondo neurone dello strato. Nel caso in cui i pesi non vengano forniti,

il loro valore iniziale sarà impostato casualmente tra 0 e 1.

Addestramento della rete neurale (metodo train())

L'addestramento è quella fase in cui la rete impara ad associare dei valori corretti a un insieme di dati in input. Per addestrare la rete si utilizza un insieme di dati già annotati, ovvero un insieme di valori di input per i quali si conosce già la risposta corretta. Tale insieme è per l'appunto chiamato insieme di addestramento (training set).

Si consideri come training set di esempio la tabella dell'AND logico, ovvero:

INPUT	OUTPUT
[0.0, 0.0]	[0.0]
[0.0, 1.0]	[0.0]
[1.0, 0.0]	[0.0]
[1.0, 1.0]	[1.0]

La fase addestramento modifica iterativamente i pesi delle connessioni in modo tale che si minimizzi una certa funzione errore **E**. L'errore generalmente si ottiene dalla differenza tra l'output calcolato a fronte di ciascun input e l'output "atteso" fornito per l'input nell'insieme di addestramento.

Input	Output atteso		Output della rete prima dell'addestramento	Errore calcolato
[0.0, 0.0]	[0.0]	==>	1.0	**E = 0.0 1.0 = 1 (elevato)**
[0.0, 1.0]	[0.0]	==>	1.0	**E = 0.0 1.0 = 1 (elevato)**
[1.0, 0.0]	[0.0]	==>	1.0	**E = 0.0 1.0 = 1 (elevato)**
[1.0, 1.0]	[1.0]	==>	1.0	**E = 1.0 1.0 = 0 (ottima)**

Alla fine dell'addestramento la nostra rete potenzialmente restituirà l'output desiderato.

Input	Output Atteso		Output della rete dopo l'addestramento	Errore calcolato
[0.0, 0.0]	[0.0]	==>	0.0	**E = 0.0 0.0 = 0 (ottima)**
[0.0, 1.0]	[0.0]	==>	0.0	**E = 0.0 0.0 = 0 (ottima)**

[1.0, 0.0] [0.0] ==> 0.0 **E = 0.0 0.0 = 0 (ottima)**

[1.0, 1.0] [1.0] ==> 1.0 **E = 1.0 1.0 = 0 (ottima)**

Algoritmo d'addestramento (pseudocodice)

```
train(  insieme_di_addestramento )
DO
        somma_errori  ← 0
        FOR (x,y) IN insieme di addestramento
                errori← trainIstanza( x, y )
                somma_errori = somma_errori + errori
        END FOR
WHILE ( somma_errori > 0.01)
```

Formula di aggiornamento

Percettrone

Una rete con un solo strato e con la funzione di attivazione Step, riceve il nome di **Percettrone**. La formula di aggiornamento dei pesi è la seguente:

$$w_i = w_i + \eta(o_i - y_i) * x_i$$

Dove η si conosce come costante di apprendimento, deve essere strettamente positiva e regola la velocità dell'apprendimento (generalmente prende valori tra 0.01 e 0.001).

x_i = input chi deve memorizzare. o_i l'output associato al input x_i, y_i output de la rete (process) e w_i sono i pesi della rete.

I threshold devono essere calcolati come se fossero altri pesi con il input in 1.

$$\theta = \theta + \eta * (o_i - y_i)$$

Rete a singolo strato

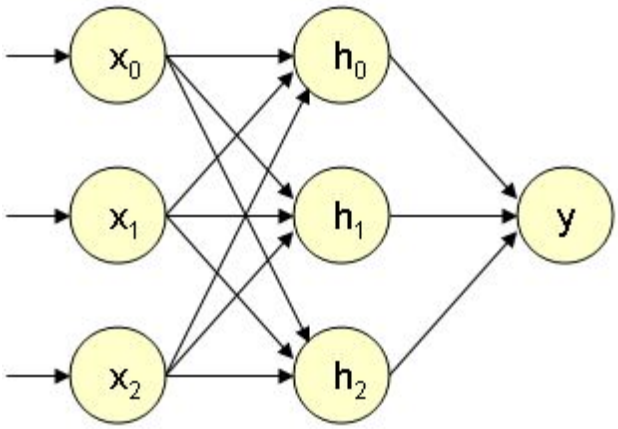
L’addestramento di un singolo strato per tutte le altre funzioni di attivazione utilizza la **derivata** f’(x). La derivata dà la direzione verso cui muoversi lungo la quale si ha il massimo incremento della funzione di attivazione. La formula di aggiornamento dei pesi è la seguente:

$$w_i = w_i + \eta * (o_i - y_i) * f'(\sum_j w_j x_j + \theta) * x_i$$

Il threshold si addestra come se fosse un’altro peso.

$$\theta = \theta + \eta * (o_i - y_i) * f'(\sum_j w_j x_j + \theta) * x_i$$

Rete a più strati



L’addestramento di una rete di due o più strati è simile. Vediamo il caso a due strati. L’ultimo strato si vede come se fosse una rete a un solo strato che prende come input l’output del primo strato. La formula di aggiornamento dei pesi e **il threshold** per l’ultimo strato è la seguente:

$$w_i^1 = w_i^1 + \eta * (o_i - y_i) * f_1'(\sum_j w_j^1 h_j^1 + \theta) * h_i^1$$

$$\theta = \theta + \eta * (o_i - y_i) * f_1'(\sum_j w_j^1 h_j^1 + \theta) * h_i^1$$

Per semplificare, si considera considera il threshold come un altro peso in tutte le formule.

Il rettangolo rosso riceve il nome di B_1.

$$w_i^1 = w_i^1 + \eta * \boxed{(o_i - y_i) * f_1'(\sum_j w_j^1 h_j^1)} * h_i^1$$

$$w_i^1 = w_i^1 + \eta * B_1 * h_i^1$$

L'addestramento del primo strato ha una formula un po' più complessa:

$$w_i^2 = w_i^2 + \eta * (o_i - y_i) * f_1'(\sum_j w_j^1 h_j^1) * w_i^1 * f_2'(\sum_j w_j^2 x_j) * x_i$$

Il rettangolo rosso qui sotto ha lo stesso valore chi il rettangolo B_1 di sopra:

$$w_i^2 = w_i^2 + \eta * \boxed{(o_i - y_i) * f_1'(\sum_j w_j^1 h_j^1)} * w_i^1 * f_2'(\sum_j w_j^2 x_j) * x_i$$

$$w_i^2 = w_i^2 + \eta * B_1 * w_i^1 * f_2'(\sum_j w_j^2 x_j) * x_i$$

E analogamente, si chiama B_2 a rettangolo rosso della formula del primo strato.

$$w_i^2 = w_i^2 + \eta * B_1 * w_i^1 * f_2'(\sum_j w_j^2 x_j) * x_i$$

$$w_i^2 = w_i^2 + \eta * B_2 * x_i$$

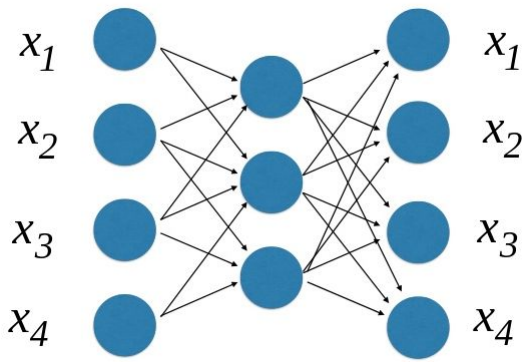
Generalizzando, pero il passo k di addestramento sarà:

$$w_i^k = w_i^k + \eta * B_k * h_i^k$$

$$B_k = B_{k-1} * w_i^{k-1} * f_k' \left(\sum_j w_j^k h_j^k \right)$$

Autoencoder (solo per gruppi di due studenti che vogliono prendere 35 punti)

l'autoencoder é una rede che ha come output i stesso valore di input.
Si usa come un preaddestramento.



In questo caso si deve un metodo

- **pretrain** riceve un insieme di valori, che sono la lista valori di input del insieme di addestramento.

```
pretrain ( insieme_di_valori )
FOR s IN strati(r)
```

r' ← nuova rete, il primo strato deve essere **s**, il secondo strato deve avere la stessa lunghezza che l'input.
r' . train((insieme_di_valori , insieme_di_valori))
si scrive i pesi di **s** sulla rete **r**
insieme_di_valori ← si calcola per ogni input dell insieme di valori l'output dello strato s
END FOR

Input di prova, funzione x^2

input	output
0.000	0.000
0.063	0.004
0.125	0.016
0.188	0.035
0.250	0.063
0.313	0.098
0.375	0.141
0.438	0.191
0.500	0.250
0.563	0.316
0.625	0.391
0.688	0.473
0.750	0.563
0.813	0.660
0.875	0.766
0.938	0.879
1.000	1.000