**Introduzione**

Con questa guida il mio obiettivo è quello di presentare il tema delle reti neurali in modo semplice per coloro che hanno almeno delle basi universitarie dei concetti di Analisi 1 e 2 per il calcolo delle derivate parziali e di Algebra lineare per il calcolo matriciale. La guida è divisa in due fasi, la prima comprende la comprensione dell’argomento in termini matematici-logici e la seconda comprende la scrittura di un codice in Python che implementi quanto enunciato.

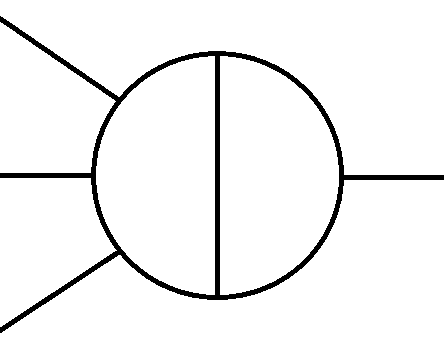
**Reti neurali**

Incominciamo dalle basi, ovvero che cos’è una rete neurale artificiale? Partendo dal modello di rete neurale biologica, una rete neurale artificiale altro non è che uno schema logico secondo il quale passano informazioni. In termini matematici, è una funzione che trasforma un vettore in input in un vettore in output.

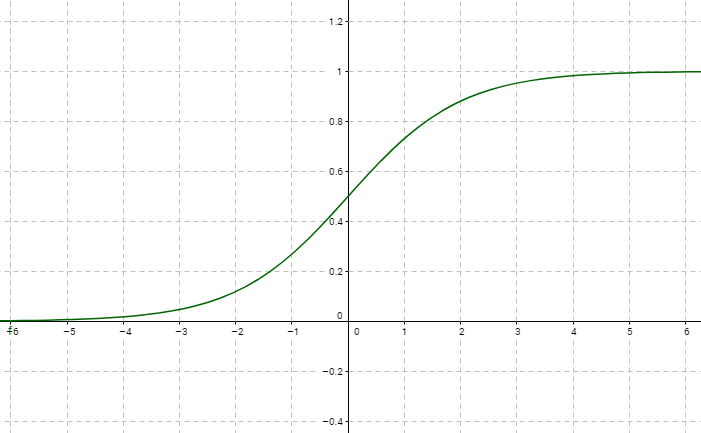
Consideriamo il vettore di input e il vettore di output in spazi multidimensionali, e la funzione la nostra rete neurale, tale che . Questo è quello che avviene a livello macroscopico della nostra rete. Focalizziamoci invece ora sull’elemento fondamentale che la costituisce: il percettrone, ovvero il neurone artificiale, corrispondente di quello biologico.

**Una rete di percettroni**

Per immaginare un percettrone possiamo prendere in esempio sempre il modello biologico del neurone. Ogni neurone è costituito essenzialmente da tre parti: dentriti, corpo cellulare e assone. I dentriti hanno il compito di ottenere le informazioni dai neuroni che lo precedono, per poi propagarla verso altri attraverso l’assone. Il percettrone si basa sostanzialmente sugli stessi aspetti di base.



Il funzionamento è il seguente: il nucleo del percettrone riceve gli input dagli assoni dei percettroni che lo precedono, li somma tra loro e al risultato ottenuto applica una funzione , chiamata “funzione attivatrice”, che normalizza i risultati ottenuti in un range definito. Nel nostro caso prenderemo in considerazione come funzione attivatrice, la *funzione sigmoidea*, in quanto è una funzione derivabile in tutto l’intervallo . Andamento della funzione sigmoidea in figura.



**Weights e biases**

Ogni percettrone riceve input da quelli che lo precedono. Gli input non arrivano con la massima intensità, ma sono ridotti di un certo fattore che va da 0 (valore minimo) a 1 (valore massimo). Questo fattore prende il nome di weight, ovvero “peso”. Per cui, ad ogni collegamento che vedremo all’interno di una rete neurale andremo a considerare un weight ad esso associato. Mentre i biases sono degli input supplementari che ogni percettrone possiede. Per definizione il bias ha sempre valore 1 e nel momento in cui viene aggiunto alla sommatoria degli input deve essere moltiplicato per il suo weight corrispondente (ad ogni collegamento corrisponde un weight). Nel nostro caso non andremo a considerare i bias, sia nei calcoli sia nella rete, ma si può applicare lo stesso metodo tenendo presente dei biases.

**Una struttura a layer**

Organizzando insieme più percettroni ottieniamo i diversi layer che compongono una rete neurale: input layer, che contiene il vettore di input; hidden layers, che contengono uno svariato numero di percettroni; output layer, che contiene il vettore di output.

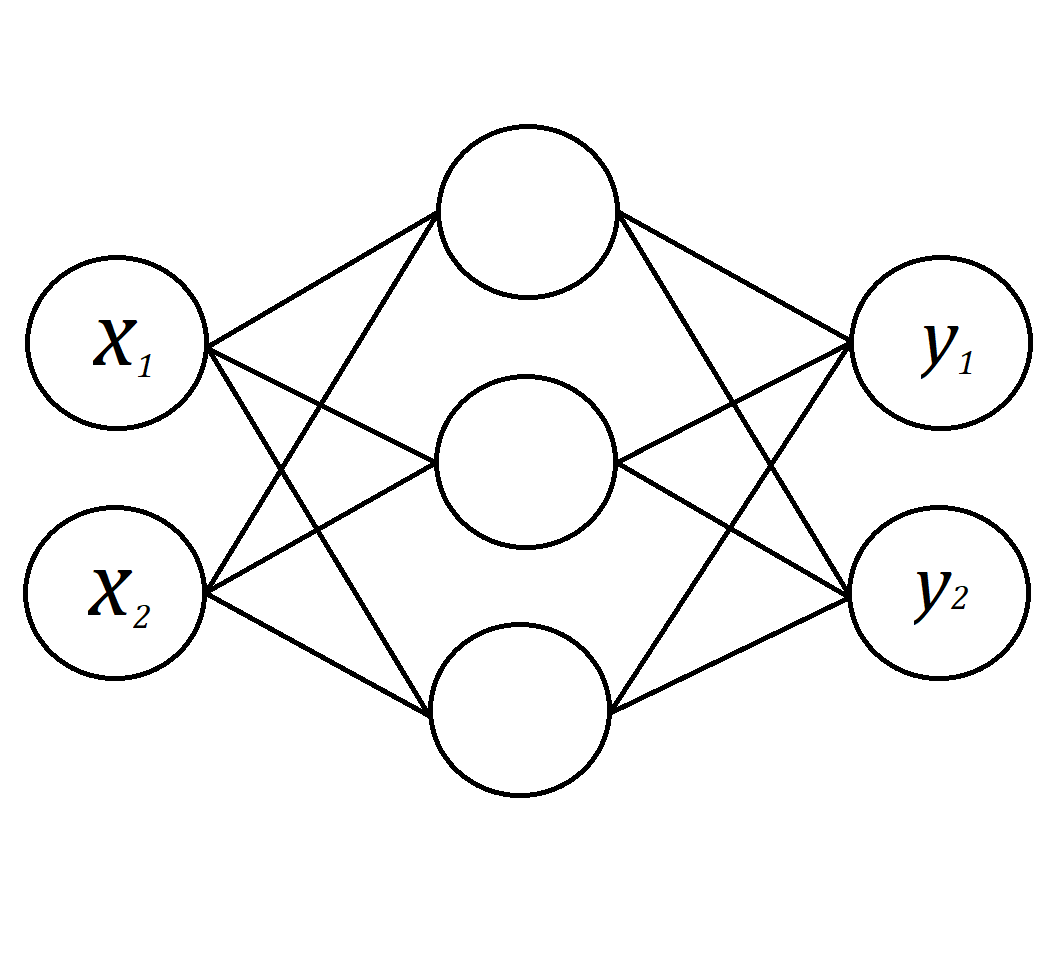


**Forward-Propagation**

Il Forward-Propagation è la tecnica che utilizza una rete neurale per far passare il vettore di input attraverso tutta la rete fino ad arrivare al vettore di output. Sull’input vengono eseguiti calcoli come prodotti, sommatorie e funzioni attivatrici in modo concatenato prima di arrivare al risultato ultimo.

**Matrici e algebra lineare**

Per il nostro esempio, prenderemo in considerazione una rete costituita da un input layer di due neuroni, un’hidden layer di tre neuroni e un output layer di altri due neuroni.



Scriviamo il vettore di input e quello di output come segue:

Appartenenti entrambi ad

Mentre scriviamo il vettore dell’output dell’hidden layer come:

A questo punto, in base a tutti i collegamenti tra i percettroni che andremo a tracciare nel grafico avremo un certo numero di weights, in questo caso 12. Inoltre possiamo dividere, per comodità, i weights tra l’input layer e l’hidden layer e tra l’hidden layer e l’output layer. Possiamo riassumere questi due gruppi in due matrici:

Dove il generico elemento è il weight che collega l’i-esimo percettrone del layer precedente al j-esimo percettrone del layer successivo. A questo punto per far avanzare il segnale di input fino all’hidden layer, dobbiamo far passare il segnale attraverso la matrice di weights e poiché i segnali che arrivano ad un neurone dell’hidden layer devono essere sommati, l’operazione che andremo ad applicare ai nostri vettori e matrici sarà un prodotto scalare. Come ultimo step andremo ad applicare la funzione sigmoidea a tutti i valori del vettore che otterremo.

Da un vettore 1x2 e una matrice 2x3 otteniamo infatti un vettore 1x3, ovvero il vettore che contiene le sommatorie degli input, moltiplicati per i rispettivi weights, di ogni singolo neurone dell’hidden layer. A questo punto applichiamo la funzione sigmoidea al vettore ottenuto.

Abbiamo ottenuto l’output dei neuroni dell’hidden layer. Questo significa che l’output di un layer si ottiene sempre applicando una composizione di funzioni , dove è il vettore di input, la matrice dei weights, e la funzione attivatrice.

Ripetendo l’operazione per tutti gli hidden layer (nel caso ce ne fossero più di uno) e anche per l’output layer al termine del processo di Forward-Propagation otterremo il vettore di output generato da una funzione che è composizione di funzioni come prodotti scalari e sigmoidee.

**L’errore e l’allenamento della rete**

Nel momento di inizializzazione della rete i weights vengono generali in modo pseudo-randomico con valori decimali compresi tra 0 e 1. È abbastanza ovvio che la rete, per essere efficiente e generare vettori in output che rispecchiano un certo target, deve aggiustare sempre più i weights di cui è costituita, secondo il metodo della *discesa del gradiente*.

Ciò che andremo a fare analiticamente sarà:

* Calcolare di quanto si discosta l’output della rete dal valore di target, dato un certo input
* Calcolare l’errore totale rispetto ad ogni singolo weight, cioè quanto ogni weight partecipa all’errore totale della rete

Per fare ciò, ci affidiamo alle ***derivate parziali*** e al ***metodo di catena***.

**Backward-Propagation**

Come prima cosa, dato un certo input, calcoliamo di quanto si discosta il valore dell’output da quello desiderato (target).

Nel nostro caso, e quindi anche il target, che indichiamo con

Considerando l’errore totale come la somma dei singoli errori del vettore di output, possiamo scrivere:

Il fattore ½ verrà semplificato con il 2 nel momento in cui andremo a derivare l’errore, quindi è una componente di cui non dobbiamo preoccuparci.

Partendo a ritroso, deriviamo l’errore totale rispetto il weight che collega il primo percettrone dell’output layer con il primo percettrone dell’hidden layer. Facciamo questo per determinare di quanto abbia contribuito all’errore totale.

Perché l’errore del secondo elemento di non dipende da . Proseguiamo con i calcoli:

Applicando la ***regola di catena*** risulta evidente che la derivata parziale dell’errore rispetto il weight preso in considerazione è uguale alla derivata parziale dell’errore rispetto il suo output moltiplicata per la derivata parziale dell’output rispetto la somma di tutti i segnali in ingresso nell’ultimo percettrone moltiplicata per derivata parziale dell’input rispetto il weight considerato.

Calcoliamo le singole derivate parziali e successivamente mettiamole insieme.

Quindi:

Abbiamo così ottenuto la **regola delta**.

Non ci resta che aggiornare il nostro weight utilizzando il valore appena ottenuto:

Dove è il *learning rate* della rete. Più il learning rate è basso più la rete apprenderà in modo lento ma preciso. Contrariamente più il learning rate è alto, più la rete impara ad ogni ciclo ma con un’approssimazione maggiore.

Il processo va ripetuto per tutti i weights nella rete prima di processare nuovamente le informazioni in un nuovo ciclo.

Con questa formula è possibile aggiornare solo i weights che si trovano tra l’output layer e l’hidden layer, ma non quelli intermedi tra gli altri layer. Per fa ciò abbiamo bisogno di fare altre considerazioni, in quanto gli altri weights partecipano anche all’errore in che finora abbiamo ovviamente considerato.

Allo stesso modo, cerchiamo di calcolare la variazione del weight che collega il primo valore in input con il primo percettrone dell’hidden layer.

Calcoliamo le derivate, ad esclusione delle prime due in quanto già calcolate in precedenza:

Facciamo lo stesso con :

Mettendo insieme le derivate parziali calcolate sia ora e sia in precedenza:

Con il valore ottenuto possiamo aggiustare il weight :

Questo ci fa capire come la variazione dei i-esimo peso dipende unicamente dall’errore del percettrone nel layer successivo e dal valore di input del percettrone nel layer precedente che esso collega. A livello pratico l’errore di ogni percettrone può essere calcolato propagando all’indietro l’errore sull’output layer per ogni rispettivo weight, sommandoli tra di loro. Questo lo si può evincere dalla formula ricavata.