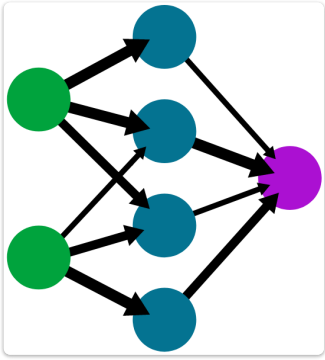
Lezione07

Table of contents

Artificial Neural Networks



Una semplice ANN feedforward

Una **rete neurale artificiale**, o Artificial Neural Network (ANN), è uno strumento che permette l'approssimazione di funzioni reali in più variabili reali. La funzione non la sappiamo esprimere, o meglio, non ci interessa farlo quanto piuttosto trovare i punti in cui la funzione è non nota, quindi approssimabile.

La parte semplice sono le singole componenti, i neuroni, la parte intricata sono i collegamenti che si vanno verso a formare tra i singoli, ovvero la rete (network).
Raggiunto un livello d'approssimazione sufficientemente alto, ci fermeremo nella computazione. Un programma facente uso di questa complessa struttura, mappa input/output per punti noti e tralascia ragionevolmente dettagli ininfluenti, che non ci interessano.

Con il **machine learning** (apprendimento), le reti vengono addestrate/costruite: tramite ciò, la rete apprende come si svolge un compito, quindi come approssimare una funzione con cui sta lavorando.

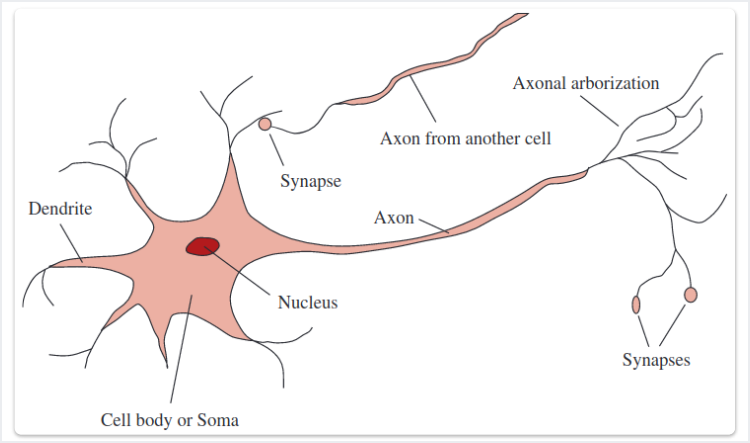
L'**addestramento** delle ANN ha lo scopo di:

- insegnare quali sono i comportamenti da aspettarsi per un certo input;
- generalizzare per assicurare comportamenti precisi a input sconosciuti.

Le ANN non riescono a giustificare i propri comportamenti:

- sono il fulcro delle AI sottosimboliche (e.g.: lo smartphone si sblocca alla percezione del viso giusto, qual'é il viso giusto?), a forza di deduzioni si raggiunge conclusione;
- sono difficili da usare per le eXplainable AI (XAI), ovvero quelle AI che al contrario giustificano il comportamento.

Neurons



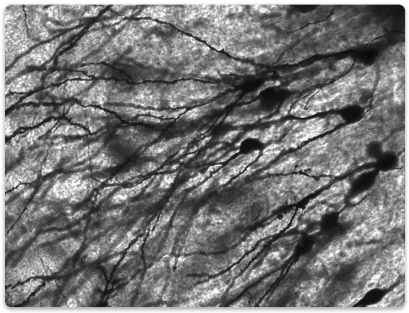
Stilizzazione delle parti fondamentali di un neurone

La possibilità di fare pensare la macchina proprio come una persona, scaturisce dal fatto che le singole componenti possono essere simulate; nella AI ci occupiamo di farlo esclusivamente dal punto di vista cellulare e non nel completo dettaglio che come sarà fatto presente, non ci è d'importanza.

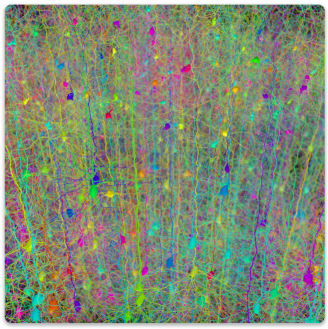
L'unità principale costituente il cervello, è il neurone.
Nel cervello umano, esistono una miriade di tipi diversi di neuroni (parliamo di una ventina); a noi non interessa simulare ogni tipo, perché quello che ci basta è il neurone esemplare, che si vede sempre nelle illustrazioni dei libri di biologia.
Come tutte le cellule, il nostro neurone ha:

- un nucleo (soma)
  il fulcro della cellula dove la maggior parte dei segnali passano
- un corpo che contrariamente alle tipicità, ha ramificazioni
  - a densità elevata, chiamate *dendriti*
    il cui scopo è quello di accettare/estendersi per formare collegamenti con altri neuroni tramite le estremità chiamate *sinapsi*
  - singole, lunghi filamenti chiamati *assoni*.

Nel cervello umano, ci sono all'incirca 10^{11} neuroni, il che sembra un numero grande quando invece reso insignificante, rispetto al numero di collegamenti potenziali che ogni neurone può assumere, 10'000.



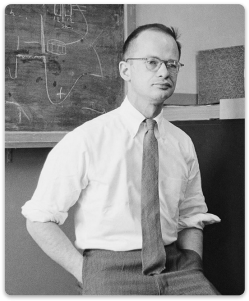
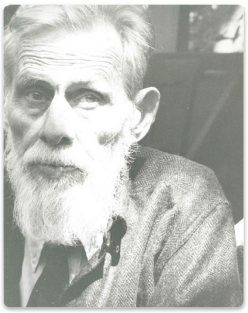
Neuroni marcati con la tecnica del Golgi



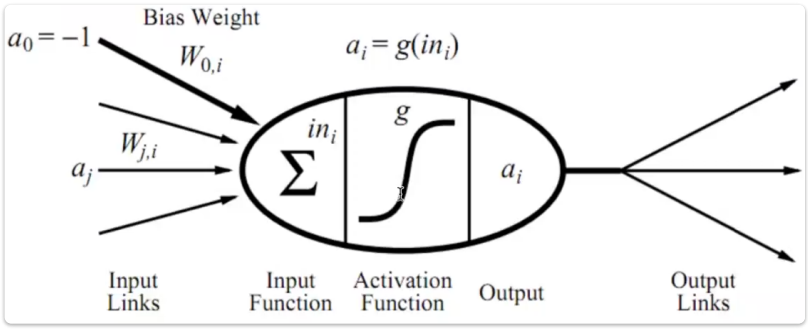
Una simulazione dei neuroni piramidali

Le informazioni, o meglio *cariche elettriche*, che avvengono all'interno dei neuroni, possono assumere verso opposto in base al senso con cui il segnale viene propagato. Il fatto che una carica entrante dalle sinapsi e passante per i dendriti sia positiva, implica che il segnale è solo in ingresso e che sarà l'unico caso che prenderemo in considerazione. Dal punto di vista molecolare, lo scambio di cariche non è altro che scambio di ioni^{+} e ioni^{-}, che non andremo a simulare in quanto non necessario, ma che ci serve sapere.

The McCulloch-Pitts Unit

	
Walter Pitts (1923-1969)	Warren Sturgis McCulloch (1898-1969)

Il cervello è un sistema complesso, non caotico.
Proprio per come funziona il cervello, nelle nostre simulazioni non vogliamo eventi in cascata: piccole variazioni dell'input, portano a enormi variazioni dell'output.
Per ovviare a questo problema, ci poniamo la domanda: come simulare un neurone?



Al quesito risposero gli allora neurofisiologi Pitts e McCulloch: proprio perché non ci interessa simulare ogni "singolo" neurone, nel senso di ogni tipo e ogni piccolezza, la nostra unità sarà identica per tutte le reti esistenti con la sola differenza della funzione che ne descrive il segnale.

Nell'unità di McCulloch-Pitts abbiamo le seguenti.

- Degli input, di cui interessa fare una somma pesata ($\sum$) perché ciascuna di essi importa e ha un peso.
  Partendo da questo proposito, ci accorgiamo che mappare un peso a 0 vorrebbe significare avere una somma pesata con risultato nullo. Per il motivo accennato sopra (proprietà non caotica), questo non lo vogliamo.
- Un bias weight ($W_{0,i}$), allo scopo di "normalizzare" l'input nel caso questo sia 0.
  Un input a 0 causa un output a 0: daremo un valore non nullo (-1) per evitarlo. Lo facciamo perché in fondo, vogliamo che qualcosa dal neurone, esca.
- La funzione di attivazione (g), a comportamento non lineare (tranne che in vicinanza all'origine).

$$a_i \leftarrow g(\mathbf{in}_i) = g\left(\sum_j W_{j,i} \cdot a_j\right)$$

Seguendo la nomenclatura dell'immagine sopra, l'equazione che descrive il neurone ha:

- i , l'indice descrivente la posizione del neurone nella nostra rete
  Se ne abbiamo 10^{11} avremo il neurone primo, secondo, terzo e così via
- j , l'indice dell'input per il neurone i
  Come in una matrice, fissata la posizione del neurone, variano sulla riga gli input j e sulle colonne l'indice del neurone j
- a_j , il valore di input del singolo neurone
- a_i , il valore di output del singolo neurone
  Che poi viene mandato tramite le sinapsi, agli input di qualche altro neurone
- $W_{i,j}$, il peso
  Associato a ciascun input del neurone

- $\texttt{in}_i$, la somma pesata degli ingressi
- g la funzione di attivazione
  In cui entra la somma pesata degli input, producente output

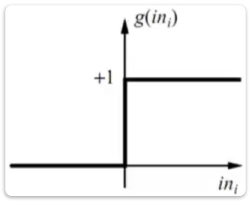
Quando vogliamo decidere che comportamento deve assumere il singolo neurone, l'unica possibilità che abbiamo è di trovare dei pesi che moltiplicati per l'input, generino l'output interessato. Se la nostra funzione assume il valore migliore che possiamo immaginarci, allora i pesi saranno quelli ottimali.

Delle funzioni di attivazione, vediamo quelle più utilizzate in campo.

- *Linear function* $L(x) = x$
  La funzione più "inutile" di tutte. Con x s'intende la somma pesata.
  Questo tipo di funzione ci sta dicendo che da un input di certo tipo, produrremo output dello stesso tipo, proprio come in un prodotto di matrici, il cui produce un'altra matrice.

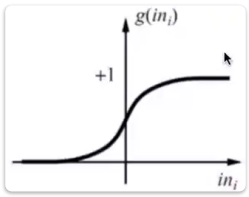
- *Rectified Linear Unit* $(\texttt{ReLU}) : R(x) = \texttt{max}\{0, x\}$
  Una funzione non lineare, usata (nel passato) parecchio nelle reti convoluzionali.
  (Proviamo a immaginarla) Dato $\texttt{max}\{0, x\}$, la funzione rimane a 0 fintanto che < 0. Salirà di 45^o partendo dall'origine, verso destra. Per valori sufficientemente grandi dell'input, la ReLU rimane lineare.

- *Heaviside Step Function* $H(x)$

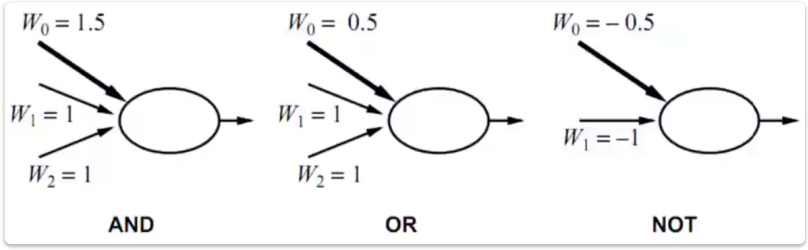
  Il fatto che si chiamino funzioni di attivazione deriva dal fatto che il neurone si attiva come se avesse una soglia: a input sufficientemente forti, il neurone si attiverà e produrrà una risposta forte e immediata (si dice che il neurone "spara").
  La funzione di Heaviside ha questo comportamento in modo estremo: l'uscita è 0 finché l'input è negativo e appena diventa positiva, spara.

- *Logistic function* $S(x) = 1/(1 + e^{-x})$

  Il fatto che il neurone spari è reale, ma non è che questo resti completamente inattivo; ha una risposta più leggera, descritta dalla funzione logistica.
  Ha la base e perché nello svolgimento dei calcoli, sarà più semplice con le semplificazioni che ne derivano.

Neurons as Logic Devices



Se la funzione è abbastanza semplice, i pesi possiamo trovarli autonomamente.
Vediamo una rete neurale degenere semplice a singolo neurone (rete ottimale) con input/output $\mathcal{B} = \{0, 1\}$, che approssimi alcune delle funzioni logiche.

AND

La funzione `AND` ha 2 input e 1 output binario, con 4 casi possibili, usando il peso bias $W_0 = -1.5$, possiamo trovare i pesi per i quali la nostra funzione si comporterà bene, utilizzando la funzione di Heaviside come attivazione g.

- caso $W_1 = 0 \land W_2 = 0$
  - moltiplicando per il peso 1, entrerà 0 a prescindere;
  - moltiplicando il peso di bias $W_0 = 1.5$ per $a_0 = -1$, otteniamo -1.5;
  - risultato: $0 + (-1.5) = -1.5$ negativo, $g(-1.5) = 0$
- caso $W_1 = 1 \land W_2 = 1$
  - moltiplicando per il peso 1, entrerà $1 \cdot 1 + 1 \cdot 1 = 2$;
  - moltiplicando il peso di bias $W_0 = 1.5$ per $a_0 = -1$, otteniamo -1.5;
  - risultato: $2 + (-1.5) = 0.5$ positivo, $g(0.5) = 1$
- caso $W_1 = 1 \land W_2 = 0$ oppure $W_1 = 0 \land W_2 = 1$
  - moltiplicando per il peso 1, entrerà $1 \cdot 1 + 0 \cdot 1 = 1$;
  - moltiplicando il peso di bias $W_0 = 1.5$ per $a_0 = -1$, otteniamo -1.5;
  - risultato: $1 + (-1.5) = -0.5$ negativo, $g(-0.5) = 0$

OR

La funzione `OR` lavora nel modo identico con cambiamento del peso di bias $W_0 = 0.5$.

NOT

La funzione `NOT` anche lei nello stesso modo, a differenza fatta per il peso bias $W_0 = -0.5$.

16/03/2023