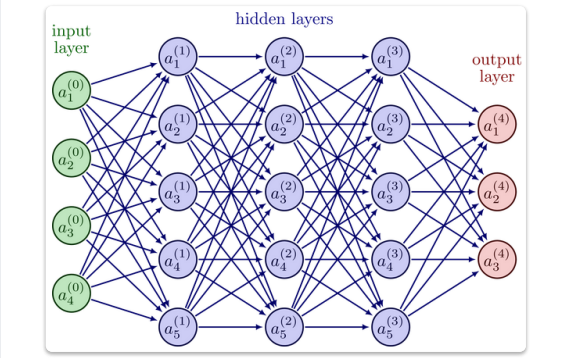
Lezione08

Table of contents

Types of Neural Networks

Feed-forward neural networks



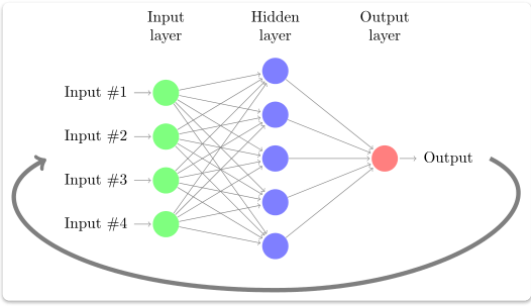
Rete Feed-forward con hidden layers esposti

Le reti più semplici da comprendere e realizzare sono le reti dette in avanti, Feed-forward, chiamate così perché nessun arco si connette con layer precedenti.
Questo tipo di reti ha una struttura a *livelli*:

- il 1^{o} livello, contiene gli input su cui la rete lavorerà, *input layer*;
- livelli intermedi si preporranno a ogni uscita, *hidden layers*;
- l'ultimo livello sarà composto dagli output, *output layer*.

La tecnica di addestramento di questa rete, come vedremo fra poco, è quella *supervisionata*. Le reti feed-forward sarebbero le reti combinatorie, che abbiamo visto quando abbiamo appreso per la prima volta, le funzioni booleane.

Recurrent neural networks



Rete ricorrente con hidden layers esposti

Le Recurrent neural networks sono sempre delle reti ricorrenti con la differenza che il *grafo diretto* che nel diagramma le descrive, non necessariamente è aciclico e non necessariamente ha livelli. La distinzione tra input/output non viene fatta nel senso d'inizio e fine, ma piuttosto per ogni iterazione compiuta, che come risultato finale avrà una stabilizzazione (ideale) su valore asintotico della funzione in approssimazione.

Questo tipo di rete viene tipicamente addestrata in modo *non supervisionato*.
In generale, le reti ricorrenti sono reti sequenziali asincrone, senza temporizzazione e quindi rinuncianti al ciclo di clock.

Types of Training

Training phases

Training phase

La prima fase dell'intero processo di addestramento, in senso stretto, è l'**addestramento**.

In questa fase, forniremo alla nostra rete un vettore di valori su cui imparare, il **training set**: daremo valori ritenuti significativi e descrittivi di quello che vogliamo che si impari e tramite l'*algoritmo di addestramento*, troveremo quell'insieme di pesi che permettono un comportamento adeguato, che segua le aspettative.

Tipicamente, l'intero training set viene dato in ingresso alla rete per più volte consecutive, siccome gli algoritmi funzioneranno per convergenza; l'insieme di pesi che si andrà via via a calcolare, sarà sempre più valido per ciascuna **epoca** di addestramento della rete.

Test (or validation) phase

Nell'addestramento, stiamo supponendo che i vettori dati in pasto alla rete siano sufficientemente descrittivi nel problema da risolvere, quando in realtà non ne siamo sicuri.

La fase di **validazione** consiste nel prendere un nuovo insieme di vettori, per il quale conosciamo il comportamento della rete, fornendolo in ingresso nel momento in cui la nostra rete ha già compiuto l'addestramento.

Analizzeremo come la rete si comporta: la rete ha imparato su un insieme di esempi, ma noi controlliamo che funzioni bene anche su un altro insieme; se lo farà, vorrà dire che il training set originale è descrittivo a sufficienza. È in questa fase che scaturisce l'indice qualitativo della rete.

> **Un esempio classico di addestramento e validazione di una rete neurale, è quello predisposto per l'apprendimento della segnaletica stradale.**

Supervised training

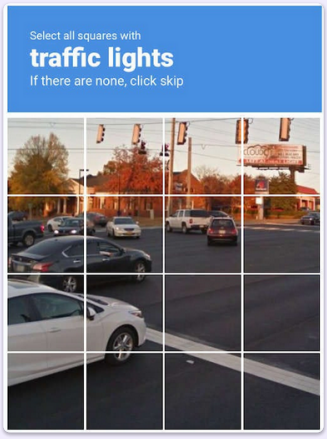
Quando facciamo apprendimento supervisionato, c'è un supervisore (o oracolo, o ente esterno) che conosce qual è il risultato corretto che la rete deve avere, per ogni input del training set.

- Il training set è composto da un'associazione di copie input/output $< x, y >$ corrette;
- L'errore della rete viene calcolato comparando il vero output y_n con l'output aspettato y;
- I pesi sono corretti per ridurre l'errore computato
  - per ogni copia input/output del training set;
  - per ogni epoca, raccogliendo un errore cumulativo.

Il modo di addestramento di cui stiamo parlando, viene nel consueto chiamato addestramento per rinforzo: fornito un input, vediamo qual è l'output; si calcola la distanza euclidea tra i due e se la distanza è 0 allora la rete in esame si sta comportando bene, altrimenti i pesi corretti e una nuova rete generata.

Continuando l'esempio dei cartelli stradali, l'oracolo ci dirrà quale di questi davvero è un cartello, mentre farà distinzione per oggetti che non c'entrano come panchine, tavoli, ecc. ecc.

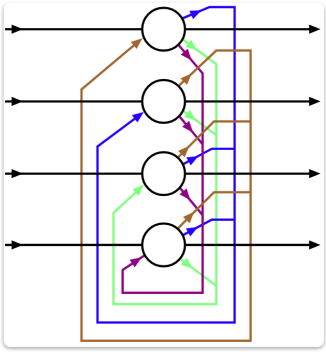
"reCAPTCHA": we are the oracles



Qualche volta ci sarà capitato, surfando una pagina web, d'incombere in queste finestre dove ci viene chiesto d'identificare i riquadri contenenti segnali stradali oppure semafori o macchine o bus.

Quello che stiamo impersonando non è altro che un *oracolo*: il dato scelto da noi, che molto probabilmente sarà quello esatto, verrà fornito alla rete e questa addestrata. Il training set della rete neurale viene fornito da noi; è proprio in questo modo che il sistema reCAPTCHA di Google ha profitto, perché il set viene poi venduto a terzi.

Unsupervised training



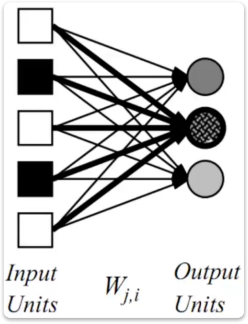
Una rete di Hopfield con 4 unità

Utilizzando caratteristiche intrinseche alle reti, forniamo campioni senza dire qual è l'output corretto. Nell'**algoritmo non supervisionato**, dire che si fa addestramento per rinforzo è sbagliato perché non stiamo più correggendo alcun errore, stiamo dicendo che l'input è corretto a prescindere.

Una volta segnati gli output asintotici, quello che ci resta da fare sarà associare i *passi per raggiungerli*, calcolando i pesi partendo dal training set, che spesso sono calcolati con semplici formule.

Fatto ciò, non è detto che gli esempi forniti siano davvero validi, dobbiamo passare comunque il nostro test set per vedere che il comportamento sia esatto.

Single-Layer Perceptrons (SLPs)



Un Single-Layer Perceptron (SLP), o percettrone a strato singolo in italiano, è formato da un unico strato di valori di input non neuroni ma semplicemente dati. Si chiama in questo modo perché gli unici effettivi neuroni sono quelli successivi al primo strato.

> ⓘ **"percettrone": termine proposto per la prima volta nel 1958 dallo psicologo americano Frank Rosenblatt - Treccani, 2008**

(Osserviamo l'immagine) Supponendo che uno degli input sia fissato a -1, quello che possiamo assumere è che abbiamo 4 variabili reali, una costante reale e 3 output, quindi una funzione del tipo $\mathbb{R}^4 \to \mathbb{R}^3$. Per un single-layer perceptron, quello che cerchiamo è una *matrice dei pesi* che scelto l'i-esimo valore d'ingresso, calcoli il peso sulla j-esima uscita: viene attribuito un peso per uno degli archi. Graficamente, gli archi sono più o meno spessi per ricordarci quale sia la grandezza dei pesi.
Esaminando più accuratamente, guardando i colori bianco e nero degli input, possiamo pensare di avere, piuttosto che variabili reali, dei valori binari: la funzione da approssimare sarà $\mathcal{B}\{0,1\} \to \mathbb{R}^3$.

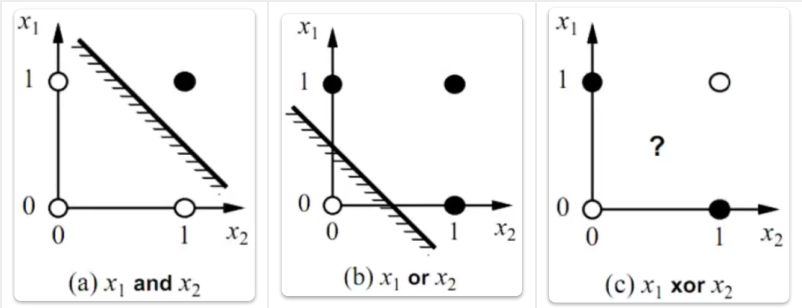
Questo tipo di rete in avanti è la più semplice che si possa realizzare.

AND, OR, XOR

Un esempio classico che implementa l'uso del SLP, è quello per la valutazione di *funzioni booleane*, in particolare `AND`,`OR` e `XOR`. Per ognuna di queste, proviamo a costruire una rete che valuti il significato corretto di ciascuna (per farlo, riprendiamo l'immagine sopra come riferimento).

$$\sum_j W_j x_j > 0 \quad \wedge \quad \mathbf{W} \cdot \mathbf{x} > 0$$

Gli input sono binari; delle 5 caselline di diversi colori, soltanto 3 input ci interessano e questi sono il peso di bias (-1) e i due valori appartenenti all'insieme $\mathcal{B}\{0,1\}$.
Di uscite ne rimarrà soltanto 1, anche lei binaria. Quindi, se vogliamo che il risulato sia booleano, la funzione di approssimazione che meglio s'addice è quella di Heaviside.



`AND`	`OR`	`XOR`

- `AND`
  - $\#0 = \{(0,0),(0,1),(1,0)\}$
  - $\#1 = \{(1,1)\}$

  Fingiamo di avere una retta (iperpiano) che separi gli input x_1, x_2: per come funziona la valutazione booleana, $\texttt{AND}((1,1)) = \texttt{TRUE}$ ovvero il pallino colorato di nero, mentre varrà `FALSE` per tutti gli altri casi.

- `OR`
  - $\#0 = \{(0,0)\}$
  - $\#1 = \{(0,1),(1,0),(1,1)\}$.

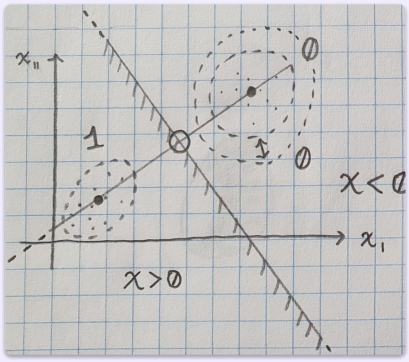
  La funzione booleana varrà $\texttt{OR}((0,1) \wedge (1,0) \wedge (1,1)) = \texttt{TRUE}$, mentre sarà `FALSE` per il resto dei casi.

- `XOR`
  Questa funzione, nel caso usassimo l'SLP, non sarebbe riproducibile o meglio, non sarebbe approssimabile con una precisione adeguata in accordo alle altre due funzioni. La retta che andiamo a prendere che separa i punti per produrre il risultato aspettato, qualunque inclinazione essa abbia, produrrà sempre almeno 1 dei 4 punti, errato (25% di errore). Ci servirebbe introdurre un layer aggiuntivo, per poter valutare correttamente la funzione.

Supervised Training for SLPs

Se forniamo i campioni, diciamo quanto viene la funzione per i singoli in ingresso, calcoliamo il valore dell'uscita in base all'insieme dei pesi che abbiamo, siamo in grado di calcolare l'errore. Quello che stiamo facendo è un addestramento supervisionato per le *SLP*: stiamo per esempio calcolando, la distanza euclidea del vettore fornito dall'oracolo e quello uscito effettivamente dalla rete.

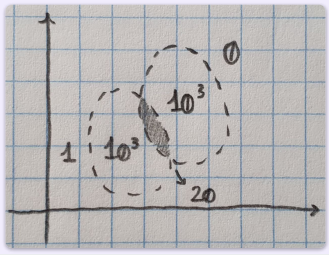
> ⁝≡ **Addestramento supervisionato in $\mathbb{R}^2$**



Immaginiamo un grafico cartesiano di ascisse x_1 e ordinate x_2 con 2 nuvole di punti distinte: la nuvola in basso a sinistra sarà quella con valore d'approssimazione pari a 1, la nuvola in alto a destra sarà pari a 0.

Nello stesso grafico, prendiamo una retta che separi le due nuvole; questa retta ha lo stesso scopo dell'esempio visto sopra: separare le due approssimazioni. Di questa retta tracciamo la sua perpendicolare passante per i 2 punti in centro alle nuvole.

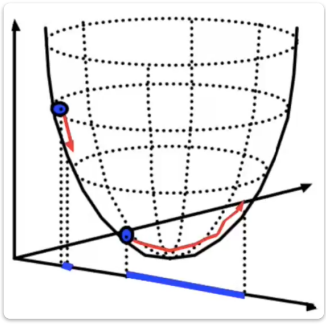
Nel momento in cui il valore approssimato si troverà a sinistra della retta originale, allora $x > 0$; se si troverà a destra allora $x < 0$. Lo scopo di questo esempio è quello di mostrare la precisione che riposa nella grandezza delle nuvole dei punti: via via che prendiamo raggio maggiore rispetto al precedente, la perpendicolare si sposterà in quanto il numero di punti che l'oracolo ha stabilito essere 1 o 0 aumenteranno. La precisione risale anche dalla densità dei punti ovvero quanto compatti sono gli uni con gli altri.



Su 2000 campioni ne sbagliamo 20 (1%)

Una rete che si comporti bene, è una rete che trovi il giusto raggio delle nuvole, per evitare che il baricentro risieda nell'errore formatosi dalla loro intersezione. Se comunque l'intersezione dei punti è di ordine decisamente inferiore rispetto alla grandezza delle due componenti intersecate, allora l'approssimazione sarà ideale.

L'errore calcolato con la distanza è una *funzione dei pesi* $\texttt{Err}(W)$.
Usando l'algoritmo di discesa del gradiente, sulla funzione dei pesi, ci è possibile determinare il percorso da seguire per ottenere un errore sempre minore per tutti gli input.



(Osserviamo l'immagine) L'algoritmo di discesa del gradiente trova un *minimo locale*. Il momento in cui abbiamo trovato questo minimo, è dettato dal comportamento del punto stazionario che presumiamo di avere

raggiunto: se il punto stazionario fluttua di un'ampiezza piccola rispetto 2 valori medi, questo verrà confermato. Il test set confermerà questa affermazione. I ratio di apprendimento lenti quindi:

- rendono la ricerca più accurata;
- aumentano il tempo necessario per convergere a un minimo locale.

La formula di aggiornamento dei pesi non è nient'altro che la formula di discesa del gradiente che se ricordiamo, è sempre fatta in questo modo

$$\frac{\delta E}{\delta W_j} = \texttt{Err} \cdot \frac{\delta \texttt{Err}}{\delta W_j} = \texttt{Err} \cdot \frac{\delta}{\delta W_j}\left(y - g(\sum_{j=0}^{n} W_j x_j)\right) = -\texttt{Err} \cdot g'(\texttt{In}) \cdot x_j$$

che riadattata usando α come *ratio di apprendimento*, diventa

$$W_j \leftarrow W_j + \alpha \cdot \texttt{Err} \cdot g'(\texttt{In}) \cdot x_j$$

- x è l'insieme delle nostre variabili per calcolare un vettore di ouput;
- $\texttt{Err} \cdot g'(\texttt{In})$ il gradiente della funzione di errore;
- α coefficiente di apprendimento che dice quanto velocemente dobbiamo muoverci nella discesa del gradiente;
- W_j è il vettore calcolato in output j-esimo.

che esplicitata diventa

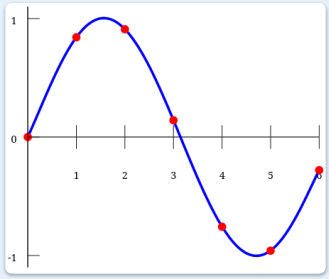
$$\bar{W}'_j = \bar{W}'_j - \alpha \nabla E(\mathbf{W})$$

L'addestramento supervisionato cerca un vettore di pesi per l'output di ogni neurone, allo scopo di minimizzare l'errore che esplicitato si calcola come

$$E = \frac{1}{2}\texttt{Err}^2 \equiv \frac{1}{2}(y - h_{\mathbf{W}}(\mathbf{x}))^2$$

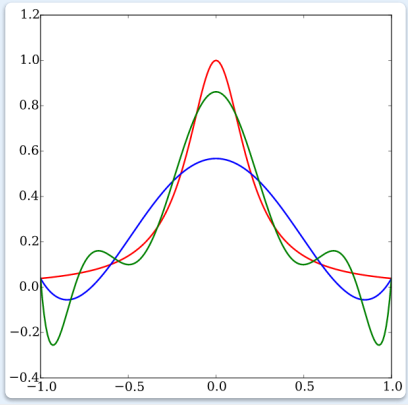
Perché non usiamo il polinomio interpolante per l'approssimazione della funzione?

Un *polinomio interpolante* è uno che ci permette l'approssimazione perfetta della funzione di cui abbiamo i punti, ma non conosciamo l'andamento.



Se abbiamo 6 punti, prendiamo un polinomio di grado 5: andando a sostituire x e y otterremo 6 equazioni lineari con 6 incognite. La risoluzione del polinomio porta a uno che non sbaglia mai, passante per tutti i punti in modo esatto.

Il problema è che se cercato in questo modo, e il numero di punti è elevato, abbiamo il *fenomeno di Runge's*, che attesta il comportamento oscillante agli estremi degli intervalli per i quali è stato calcolato il polinomio interpolante.



La funzione di Runge si comporta molto male se preso un ordine 9 di punti su cui viene fatta l'interpolazione (curva verde)

In punti lontani dal training set, la rete si comporterà male non tanto perché abbiamo trovato i coefficienti, ma piuttosto perché abbiamo scelto un polinomio: la scelta della funzione polinomiale fa sì che lontano dai punti interpolanti, la rete si comporti male.

Generalization and Overfitting

Trovando l'errore minimo, abbiamo insegnato alla rete a essere molto precisa sul training set ma non abbiamo detto come comportarsi sul test set: il fenomeno viene chiamato di **overfitting**. La rete è molto brava ad approssimare la funzione nei punti forniti, un errore nullo magari, ma per garantire che proprio in questi punti non sbagli, avrà comportamenti bizzarri nei restanti.

I comportamenti saranno scarsi per altri casi che non siano stati inclusi nel training set; bisogna trovare un equilibrio trovando un training set che sia abbastanza descrittivo, spiengendo sulla correttezza dei pesi in modo abbastanza ragionevole senza pretendere un errore nullo ma corrispettivo al test set e quindi abbastanza piccolo.

Per farlo si va a tentativi, costruendo SLP e osservando il loro comportamento: se si comportano male, vorrà dire che un singolo layer non ci basta e ci serve qualcosa di meglio.

21/03/2023