Lez3_

October 29, 2023

1 Lezione 3 (Russo Russo)

1.1 Riepilogo

Neuroni biologici: già da molti anni gli scienziati sono stati attratti dall'idea di creare macchine intelligenti, facendosi ispirare dal cervello umano. In particolare ha grande importanza il *neutone*: elemento semplice che riceve una certa quantità di impulsi elettrici a conseguenza dei quali ne emette altri. Il nostro cervello è in grado di fare determinate cose grazie ad una rete di neuroni.

1.2 Modello del Perceptron

Modello molto importante, si hanno una serie di numeri reali in input, sommati tra loro pesando ciascuno con un peso w (parametro calcolato in fase di addestramento), la somma pesata viene poi data a una funzione (in questo caso segno) che restituisce il valore di output. Tramite questo modello si può fare classificazione binaria.

Oltre a vettore di input x, si prevede un parametro b che è termine di bias aggiornato durante la fase di training. Viene eseguito l'update **SGD** (aggiornamento dei pesi) e il calcolo del gradiente. Limite: Può comprendere solo modelli lineari perciò si è lasciato perdere (poi si sono utilizzati più perception vari layer, difficile da addestrare).

1.3 Perceptron output della funzione XOR

Sembra impossibile disegnare una retta per dividere istanze di tipi diversi. Perceptron è in grado di apprendere funzione sottostante solo per decisioni lineari. Per migliorare si possono mettere insisme più neuroni ottenendo una rete neurale artificiale.

Esitono 2 famiglie: - FeedForward: reti neurali senza cicli, info non torna mai indietro. - Ricorrenti: reti neurali in cui si hanno cicli. ### Reti feedforward Neuroni organizzati in livelli (3 tipologie): - livello di input: non svolge alcun ruolo a livello di computazione, rappresenta i dati di ingresso al modello (vettore bidimensionale $\rightarrow$ 2 unità). - hidden layers: trasformano input per produrre output diversi (possono essere più di uno, processando output a sequenza). - output layer: processa l'output degli hidden layers e torna un output.

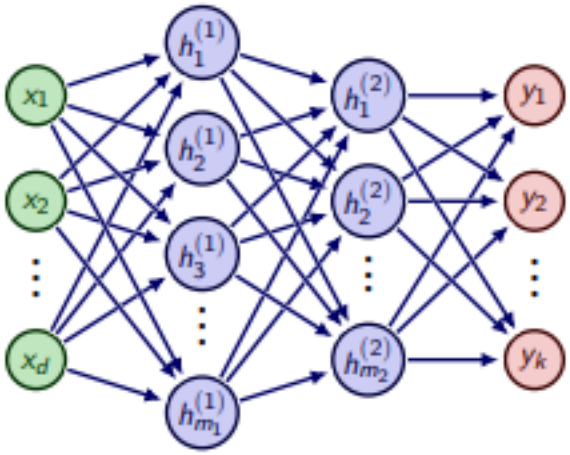
Output del modello di regressione. L'informazione viaggia solo da input verso uscita senza cicli. Per il momento assumiamo che livelli completamente connnessi tra loro (unità modelli di input tutte interconnesse con tutte quelle dei livelli successivi).

SciKit learn si usa la classe `MLPClassifier`.

Notazioni

- Ciascuna unità del modello di input è associata ad una variabile x_i
- Con $h^{(i)}_{m_1}$ indichiamo output dell'i-esimo livello nascosto (l'*apice* indica il livello nascosto a cui ci stiamo riferendo, il *pedice* indica il numero di componente considerato di quel livello). Le y rappresentano il vettore di output.

Il numero di livelli che compone la rete escludendo l'input si indica con L ed è chiamato profondità della rete. (in esempio rete con 3 livelli)



Questo è collegato al concetto di deep learning (reti profonde con molti livelli). Una rete con molti livelli implica molti livelli nascosti.

Il numero di unità nel livello è chiamata *ampiezza*.

Unità del livello di input mandano in uscita l'input non modificato, unità nel livello nascosto e di uscita applicano una funzione di attivazione al livello di uscita. Tutte le unità nello stesso livello usano la stessa funzione di attivazione (per ogni livello bisogna capire qual è). Tra livelli diversi le funzioni di attivazione possono esserre differenti.

$h_1 = \phi(\sum_{j=1}^{d} w_{1,j} \cdot x_j + b_i)$ dove:

- 1 indica pesi del primo livello nascosto.
- j indica che il peso proveniene dalla j-esima componente del livello (i-1)-simo.
- Il termine di bias b è specifico per questa unità (i indica che ho termine di bias per qualsiasi unità).

Il risultato di questa somma è input della funzione ϕ di attivazione che produce l'output di questo neurone artificale.

Scrivendo il tutto in forma vettoriale: $h = \phi(W \cdot x + b)$ con:

- W matrice dei pesi M righe e d colonne (ogni riga contiene pesi dell'i-sima unità del modello nascosto).
- b vettore dei bias,
- x vettore input.

Con più livelli nascosti:

- $h_i^{(l)}$ i-sima unità del livello l.

- $w^l_{i,j}$ denota il peso della connesione del j-simo unità del (l-1)-simo livello del i-simo unità j-simo livello.

1.3.1 Funzioni di attivazione

- **ReLU (Rectified Linear Unit)** $\phi(z) = max\{0, z\}$, rappresenta z per input positivi e 0 per negativi.
- **Parametric ReLU** $\phi(z) = \begin{cases} z & \text{se } x \geq 0 \\ p \cdot z & \text{se } x < 0 \end{cases}$, rappresenta z per input positivi e $p \cdot z$ per negativi (utile per algoritmi basati su gradiente perché derivata negativi diversa da 0); p numero molto piccolo.
- **Logistic sigmoid** mappa su un intervallo tra 0 e 1 $\sigma(z) = \frac{1}{(1+e^{(-z)})}$
- **Tangente iperbolica** mappa l'input tra -1 e 1 $tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

1.3.2 Output Units:

- Nel caso di *funzione di regressione* non si usa funzione di attivazione nel livello di uscita (o comunque non si usa la sigmoid, altrimenti l'output sarebbe compreso tra 0 e 1). $y = W^{(L)}h^{(L-1)} + b^{(L)}$
- Se voglio fare *classificazione binaria* mi basta un neurone per l'uscita che mi calcola la probabilità che appartenga ad una classe piuttosto che all'altra. $y = \sigma(W^{(L)}h^{(L-1)} + b^{(L)})$
- Per fare *classificazione* con n possibili valori quello che si fa è mettere un neurone per ogni classe, ogni neurone rappresenta la probabilità che il neurone appartenga ad una certa classe. Per questo tipo di problemi si utilizza il livello soft-max (nel livello di uscita calcolo la combinazione lineare del modello precedente ottenendo $z = W^{(L)}h^{(L-1)} + b^{(L)}$ combinazione lineare degli input dell'ultimo livello, attraverso la softmax poi calcolo l'uscita i-sima come $y_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_i}}$. Tutti i valori saranno compresi tra 0 e 1 e rappresentano ognuno la probabilità di appartenenza alle varie classi).

Partire dalla ReLU è sempre una buona scelta.

Nota

- Se usassimo *sigmoid*, cioè una funzione di attivazione che realizza l'output tra 0 ed 1, sarei limitato da tale range nell'uscita.

1.3.3 NN and Function Composition

Quello che fa il livello di uscita dipende dal task che sto cercando di risolvere con questa rete. Se ho a che fare con: - Regressione → posso usare una unità di uscita che applica regressione lineare. Posso pesare le uscite dell'ultimo livello nascosto per poi restituir equest oocome riulktat

- Classificazione binari → uso sigmoid per uscite dell'ultimo livello nascosto (mi dà la prbabbilità che input appartenga a una delle due classi).

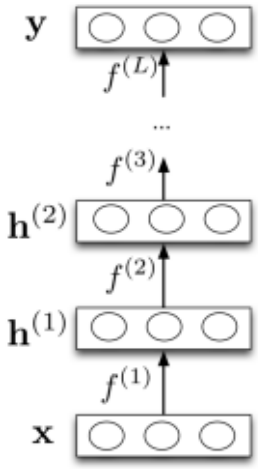
La probabilità è più o meno accurata a seconda dei valori di W e b (questi vanno calcolati nel training).

Let $f^{(\ell)}$ be the function implemented by the ℓ-layer, e.g.:

$$\mathsf{h}^{(1)} = f^{(1)}(\mathsf{x}) = \phi^{(1)}(\mathsf{W}^{(1)}\mathsf{x} + \mathsf{b}^1)$$
$$\mathsf{h}^{(2)} = f^{(2)}(\mathsf{h}^{(1)})$$
$$\ldots$$
$$\mathsf{y} = f^{(L)}(\mathsf{h}^{(L-1)})$$

The NN computes the composite function:

$$\mathsf{y} = f^{(L)} \circ f^{(L-1)} \circ \cdots f^{(1)}$$



$f^{(l)}$ è la funzione che viene implementata dal livello j dati gli input. L'uscita y sarà l'applicazione della funzione $f^{(l)}$ all'output di h^{l-1}. Se metto tutto insieme vedo che la rete neurale implementa una funzione composta da tutte le funzioni dei vari livelli. Questo mi dice che posso tracciare una linea di separazione tra livello di uscita e quelli precedenti.

La funzione implementata dall'ultimo livello di uscita viene applicata al risultato di tutte le funzioni dei livelli precedenti. Lo applico a features,che non sono calcolate a mano ma dagli altri livelli. Mi basta un modello lineare applicato a features più elaborate che vengono calcolate dai primi livelli delle reti neurali.

Tutte le funzioni di attivazione hanno la caratteristica di essere non lineari, complicando quindi il modello.

Non potrei usare le lineari anche nei livelli 'h'? Considero un modello avente funzione lineare:

- $h = W^{(1)}x + b^{(1)}$
- $y = W^{(2)}h + b^{(2)}$.

Da ciò si ottiene:

$y = W^{(2)}(W^{(1)}x + b^{(1)}) + b^{(2)} = W^{(2)}W^{(1)}x + W^{(2)}b^{(1)} + b^{(2)} = W'x + b'$

Questa funzione così ottenuta sembra la h. E' fondamentale avere funzioni non lineari altrimenti potrei avere diversi livelli con la stessa capacità di apprendimento di una rete con un unico livello.

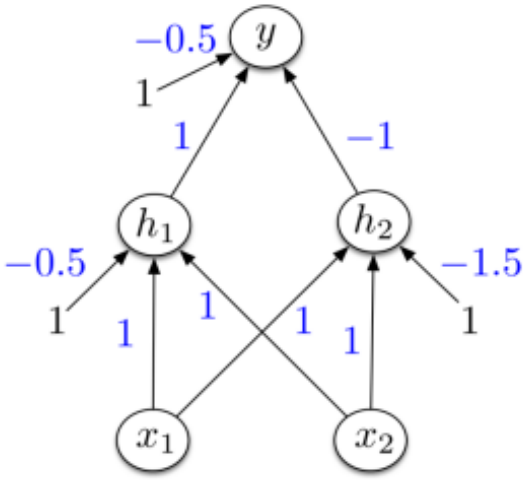
Esempio XOR XOR non poteva essere appreso dal Perceptron, vediamo le reti neurali. Ho: - in input 2 variabili binarie - 1 livello nascosto con funzione di Heaviside (gradino 1 se numero positivo, 0 se negativo) come attivazione - 1 livello di uscita. Supponiamo il livello sia già stato addestrato e quindi presenti parametri.

esempio slide provando con numeri $x_1 = 1$, $x_2 = 0$ si ha:

$h_1 = H(x_1 \cdot 1 + x_2 \cdot 1 - 0.5) = H(0.5) = 1$

$h_2 = H(1 \cdot 1 + 0 \cdot 1 - 1.5) = H(-0.5) = 0$

$y = H(1 \cdot 1 - 0 \cdot 1 - 0.5) = H(0.5) = 1$



$$H(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$

xor.ipynb

Nell'implementazione fornita comunque manca il training.

Test superato dalla rete neurale.

Quanto sono potenti le reti neurali? In generale come faccio, devo trovare la funzione di attivazione giusta per ogni problema? No, esiste il Universal approximation theorem che dice che una feedforward NN con un livello di output lineare, una qualsiasi funzione di attivazione "squashing" (range valori di uscita è un intervallo finito), e abbastanza unità hidden può approssimare qualsiasi funzione Borel misurabile da uno spazio a dimensioni finite a un altro con qualsiasi grado di errore.

Problema:

- teorema non ci dà un metodo per addestrare la rete, il training può fallire perchè non riusciamo a calcolare pesi che ci permettono di calcolare la funzione che ci serve (non fissa un algoritmo per trovare pesi)
- si può cadere nell'overfitting, impariamo la funzione sbagliata (impara troppo bene il training set)
- Non sappiamo la grandezza del livello nascosto, potrebbe anche essere vicino all'infinito.

Se ho un modello talmente perfetto da poter apprendere qualsiasi funzione → apprende qualsiasi cosa sia presente nel training set.

Un solo livello nascosto abbastanza grande può portare a problemi per training e per computazione. Usarlo in pratica non è sempre una passeggiata, ci vogliono accortezze. Nella ricerca successiva si sono ottenuti risultati migliori usando reti *più profonde* invece che *più larghe*. Questo ha portato all' utilizzo di reti sempre più profonde (concetto di deep learning,aggiungendo livelli nascosti a reti neurali).

Tradizionalmente si distingue tra reti:

- deep: più livelli nascosti
- shallow: pochi livelli nascosti

Esempio Rete che doveva, a partire da una immagine, leggere numero civico presente. Risultati: Aumentando livelli, in particolare passando da 3 ad 11, si nota un aumento della accuratezza: (3:92% $\rightarrow$ 11: 96%) miglioramento del 4% molto significativo.

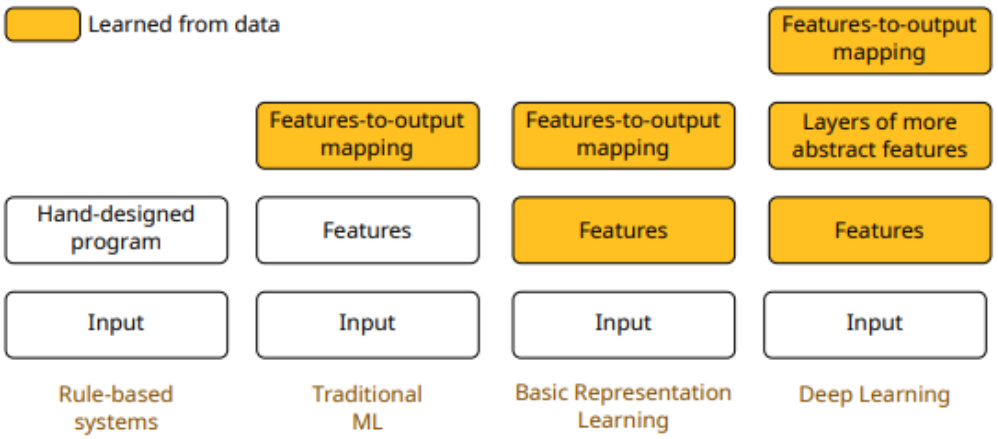
1.4 Deep learning

Collezione di metodi di ML basati su reti neurali. Rete neurale deep per fare representation learning. Il fatto di usare più livelli ha portato al cambiamento di quello che le reti neurali fanno. (nel ML si costruisce una rappresentazione di dati e si costruisce una mappa da rappresentazione a uscita). Il deep learning ha come obiettivo imparare rappresentazione e uscita. A partire dai dati riesce a costruire una rappresentazione per poi imparare mapping desiderato.

Avere più livelli aumenta il livello di composizione, a partire dai dati grezzi ogni livello costituisce una nuova rappresentaizone.

1.4.1 AI approaches

- Approccio più semplice sistema basato su regole. più if da cui deriva una risposta (sperabilmente giusta).
- ML tradizionale: prende input e features e algoritmo apprende mapping tra rappresentazione e output
- Basic rapresentation learning: prende l'input, apprende features e mapping tra queste e l'output.
- deep learning: prende l'input, ed apprende features, dalle quali genera nuovi livelli di abstract features usate per il mapping con l'output.



Esempio Prendiamo una rete che cerca di capire, data una immagine, se si tratta di animale, persona, oggetto. Per fare questo cerca di capire (ogni livello ha un compito) bordi (quando cambia colore) e angoli (bordi posizionati in un certo modo), forme, identità della forma. Il fatto che un livello sia in grado di capire che un oggetto è un bordo dipende dal training non dalla funzione di attivazione. Tutto questo viene fatto in base ai colori RGB dei pixel.

Evoluzione Negli ultimi anni si è avuto un boom del deep learning dovuto a: - numero maggiore di dati a disposizione - hardware migliore (per calcolo parallelo es. GPU) - librerie software

Dal 2015 applicato a reinforcement learning combinato a deep learning (utilizzato ad esempio nelle auto a guida autonoma). reinforcement learning si ha un'entità che impara perché riceve feedback.