Appunti del Corso di Intelligenza Artificiale

Reti Neurali

Prof. Federico Bergenti

27 marzo 2024

1 Reti Neurali e Neuronali

Per la neurofisiologia, un cervello è un organo formato da cellule tra loro connesse che prendono il nome di **neuroni**. Nel suo complesso, quindi, un cervello è una **rete neuronale**. Nel caso specifico del cervello umano, normalmente si ritiene che:

1. Ogni cervello contenga un numero di neuroni dell'ordine di 10^{11};
2. Ogni cervello contenga 20 tipi di neuroni o poco più; e
3. Ogni cervello contenga un numero di collegamenti tra neuroni dell'ordine di 10^{14}.

Quindi, la rete neuronale che forma un cervello umano è caratterizzata da un numero enorme di neuroni quasi tutti uguali tra loro collegati mediante relativamente poche connessioni. Le connessioni consentono di trasportare informazioni attraverso la rete e ogni neurone riceve e invia informazioni da e per pochi altri neuroni con un tempo di reazione dell'ordine di 1 ms e, quindi, abbastanza lentamente.

Figura 1 mostra una schematizzazione delle parti principali di un neurone e, in particolare, mostra:

1. Il **soma**, che è la struttura cellulare che racchiude il **nucleo** di un neurone;
2. I **dendridi**, che sono i collegamenti in grado di veicolare informazioni mediante il movimento di ioni; e
3. Le **sinapsi**, che sono i punti di interfaccia tra diversi neuroni e si trovano nelle parti terminali delle **arborizzazioni** dell'**assone** di ogni neurone.

Il trasporto di informazioni tra i neuroni avviene mediante l'interazione tra dendridi e sinapsi di diversi neuroni. Quindi, ogni neurone ha una struttura semplice in grado di elaborare le informazioni veicolate dal trasporto di ioni mediante dendridi e sinapsi.

Una **rete neurale** (o **rete neurale artificiale**, da **Artificial Neural Network**, **ANN**) è uno strumento hardware/software in grado di simulare una rete di neuroni. Essa funziona con gli stessi principi della rete neuronale del cervello umano anche se, normalmente, una rete neurale è strutturalmente molto più semplice di un cervello umano. In più, le reti neurali più comuni hanno un'architettura dei collegamenti tra i neuroni progettata per uno specifico scopo. Quindi, anche se è possibile dire che una rete neurale sia un simulatore di un semplice cervello umano, le reti neurali sono progettate per specifici scopo e solo marginalmente sono pensate per riprodurre i fenomeni neurofisiologici.

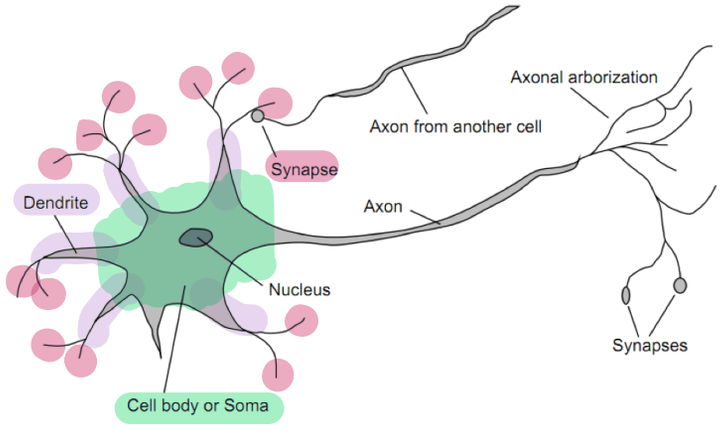

Figura 1: La schematizzazione di un neurone

Una rete neurale simula il comportamento di una rete di neuroni andando a simulare il comportamento dei singoli neuroni. Il simulatore di un neurone viene detto **unità di McCulloch-Pitts**. L'unità di McCulloch-Pitts non è altro che un modulo hardware/software pensato per simulare il comportamento di un neurone dal punto di vista del trasporto e dell'elaborazione delle informazioni. Infatti, un'unità di McCulloch-Pitts riceve informazioni rappresentate mediante numeri reali tramite una quantità prefissata di canali di comunicazione. Poi, l'unità elabora l'informazione ricevuta producendo un valore numerico legato ai valori ricevuti dai canali di comunicazione. Si noti che, spesso, quando si parla di reti neurali, le unità di McCulloch-Pitts sono chiamate semplicemente neuroni.

In sostanza, un'unità di McCulloch-Pitts non è altro che una funzione reale di $n \in \mathbb{N}_+$ variabili reali, dove n è noto e potenzialmente diverso per ogni neurone simulato. La funzione realizzata da un'unità di McCulloch-Pitts con $n \in \mathbb{N}_+$ ingressi è molto semplice e può essere descritta come segue:

$$y = g\left(\sum_{i=1}^{n} w_i x_i - w_0\right)$$

dove $y \in \mathbb{R}$ è il valore prodotto dall'unità a fronte degli n ingressi reali $(x_i)_{i=1}^n$ utilizzando n coefficienti reali $(w_i)_{i=1}^n$ detti **pesi**, un coefficiente reale aggiuntivo w_0 detto **peso di bias** e una funzione $g : \mathbb{R} \to \mathbb{R}$ detta **funzione di attivazione**. Nell'ambito di una rete neurale, il risultato del calcolo di un'unità di McCulloch-Pitts viene spesso utilizzato come ingresso di un'altra unità collegata alla rete. In questo modo, una rete neurale è in grado di compiere elaborazioni complesse essenzialmente legate alla struttura della rete e non alle capacità di calcolo delle unità di McCulloch-Pitts.

Si noti che un'unità di McCulloch-Pitts viene descritta completamente dai pesi, comprensivi del peso di bias, e dalla funzione di attivazione. Normalmente, la funzione di attivazione è fissata per tutta la rete e quindi la descrizione di un'unità di McCulloch-Pitts si riduce all'enumerazione dei pesi. Quindi, spesso, si preferisce estendere gli n ingressi con un ingresso aggiuntivo fissato al valore -1 in modo da poter descrivere un'unità di McCulloch-Pitts solo mediante un vettore dei pesi $\mathbf{w} = (w_i)_{i=1}^{n+1}$, dove w_{n+1} è il peso di bias. Noto $\mathbf{w}$, l'elaborazione dell'unità è espressa da

$$y = g(\mathbf{x} \cdot \mathbf{w}) \tag{2}$$

dove $\mathbf{x} = (x_i)_{i=1}^{n+1}$ e $x_{n+1} = -1$.

Normalmente, la funzione di attivazione di un'unità di McCulloch-Pitts non è un'arbitraria funzione reale di variabile reale, ma viene scelta tra una delle seguenti funzioni di uso comune. Si noti che, oltre a quelle discusse, esistono tante altre funzioni di attivazione utilizzate per applicazioni reali, ma quelle elencate nel seguito sono tra quelle più utilizzate.

La prima possibilità, che è la più semplice e la meno usata, è di scegliere come funzione di attivazione la **funzione identità**

$$\mathrm{Id}(x) = x \tag{3}$$

Si noti che questa scelta rende l'unità di McCulloch-Pitts una funzione lineare. La seconda possibilità, spesso utilizzata per la sua semplicità realizzativa, è la funzione **ReLU** (da **Rectifier Linear Unit**)

$$\mathrm{R}(x) = \max\{0, x\} \tag{4}$$

per $x < 0$, la f.ne ritorna 0

La terza possibilità, spesso utilizzata per elaborare valori binari mediante reti neurali, è la **funzione di Heaviside** (con $\mathrm{H}(0) = 1$)

$$\mathrm{H}(x) = \begin{cases} 1 & \text{se } x \geq 0 \\ 0 & \text{se } x < 0 \end{cases} \tag{5}$$

La quarta possibilità, simile alla precedente, è la funzione **signum** (o segno)

$$\mathrm{sgn}(x) = \begin{cases} 1 & \text{se } x > 0 \\ 0 & \text{se } x = 0 \\ -1 & \text{se } x < 0 \end{cases} \tag{6}$$

La quinta possibilità, che è la più frequente perché è non lineare e derivabile ovunque, è la **funzione logistica** (o **funzione sigmoide**)

$$\mathrm{S}(x) = \frac{1}{1 + \mathrm{e}^{-x}} \tag{7}$$

Figura 2 mostra due delle cinque funzioni di attivazioni discusse in precedenza, la funzione ReLU e la funzione sigmoide.

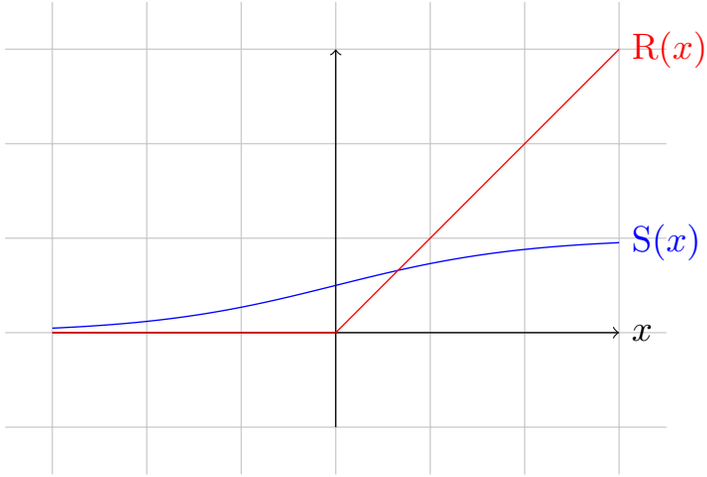

Figura 2: Le funzioni $R(x)$ (rossa) e $S(x)$ (blu)

Una rete neurale è quindi un grafo orientato pesato i cui nodi sono unità di McCulloch-Pitts e, per ogni unità, i pesi sugli archi sono i pesi in ingresso all'unità, comprensivi del peso di bias associato ad un ingresso aggiuntivo fissato a -1. Normalmente, non si ammettono autoanelli in una rete neurale in modo da non dover tenere conto dei ritardi introdotti dalle singole unità di McCulloch-Pitts. In generale, per una rete neurale non è possibile identificare quale sia l'ingresso e quale sia l'uscita, anche se è sempre possibile identificare quale sia l'ingresso e quale sia l'uscita di una singola unità di McCulloch-Pitts perché il grafo è orientato. Si noti che è sempre possibile ipotizzare che una rete neurale sia un grafo completamente connesso, eventualmente utilizzando dei pesi nulli per indicare l'assenza di archi tra unità di McCulloch-Pitts.

Data una rete neurale con $n \in \mathbb{N}_+$ neuroni, fissato un qualsiasi istante $t \in \mathbb{R}_{>0}$, il vettore $\mathbf{s}(t) \in \mathbb{R}^n$ formato delle uscite correnti dei neuroni viene detto **stato della rete** all'istante t. Lo stato della rete evolve dinamicamente partendo da uno stato iniziale $\mathbf{s}(0)$ che, tipicamente, si assume noto. In generale, però, non è detto che una rete neurale raggiunga uno stato finale.

Siccome, spesso, le unità di McCulloch-Pitts utilizzate in una rete neurale sono tutte caratterizzate dalla stessa funzione di attivazione, la descrizione della rete si riduce all'enumerazione di tutti i pesi associati agli archi del grafo completamente connesso che descrive la rete. Quindi, senza perdere di generalità, pur nell'ipotesi che tutte le unità di McCulloch-Pitts utilizzino la stessa funzione di attivazione, è possibile dire che una rete neurale viene descritta completamente una volta che siano noti i pesi associati agli archi che collegano le unità della rete.

Si noti che, una volta identificati i pesi adeguati, è anche possibile utilizzare una rete neurale per realizzare funzioni dello stato della rete. In questi casi, si ipotizza che esistano delle unità di McCulloch-Pitts che possano ricevere valori da elaborare dall'esterno. Coerentemente, si ipotizza che esistano delle unità di McCulloch-Pitts i cui risultati vengano resi disponibili verso l'esterno.

---

spiegazione file a parte

Esempio 1. Un'unità di McCulloch-Pitts che utilizza la funzione di Heaviside come funzione di attivazione può essere utilizzata per realizzare le funzioni dell'algebra di Boole sull'insieme $\{0, 1\}$. Come mostrato in Figura 1(a), se la funzione da realizzare ha $n \in \mathbb{N}_+$ ingressi, allora l'unità avrà n ingressi collegati con l'esterno, un ingresso fissato a -1 per il peso di bias, e un'uscita collegata con l'esterno.

Naturalmente, i pesi indicati in Figura 1(a) non sono gli unici possibili. In generale, se la funzione da realizzare deve associare un ingresso $\mathbf{x} = (x_i)_{i=1}^3$, con $x_3 = -1$, al valore 1, allora sarà sufficiente trovare un vettore dei pesi $\mathbf{w} = (w_i)_{i=1}^3$ tale che $\mathbf{w} \cdot \mathbf{x} \geq 0$. In modo simile, se l'uscita deve valere 0, allora dovrà essere $\mathbf{w} \cdot \mathbf{x} < 0$. Però, non è possibile trovare un vettore dei pesi adeguato a realizzare la funzione XOR, come mostrato in Figura 1(b). Infatti, non è possibile trovare un vettore dei pesi per cui valgano contemporaneamente le seguenti disuguaglianze:

$$\begin{aligned} \mathbf{w} \cdot (1, 0, -1) &\geq 0 & \mathbf{w} \cdot (0, 1, -1) &\geq 0 \\ \mathbf{w} \cdot (0, 0, -1) &< 0 & \mathbf{w} \cdot (1, 1, -1) &< 0 \end{aligned} \tag{8}$$

l'elemento del vettore x in 3° posizione è sempre -1

perché queste disuguaglianze equivalgono alle seguenti disuguaglianze

$$\begin{aligned} w_1 - w_3 &\geq 0 & w_2 - w_3 &\geq 0 \\ -w_3 &< 0 & w_1 + w_2 - w_3 &< 0 \end{aligned} \tag{9}$$

ma $w_1 \geq w_3 > 0$ e $w_2 \geq w_3 > 0$ sono incompatibili con $w_3 > w_1 + w_2$.

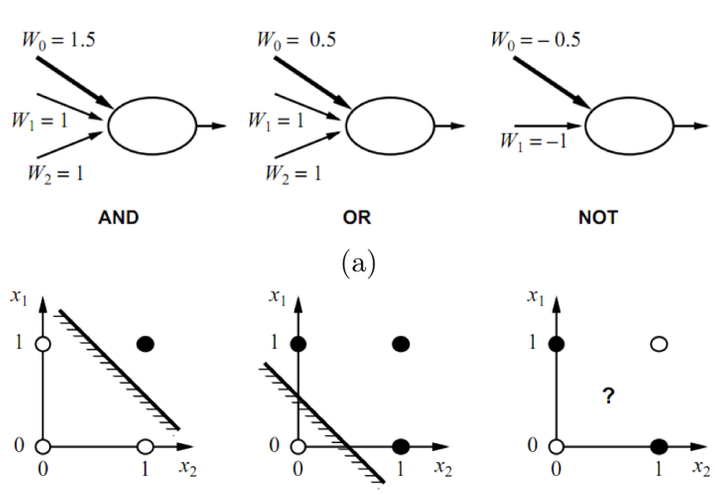

Figura 3: Le funzioni principali dell'algebra di Boole (a) realizzate mediante singole unità di McCulloch-Pitts e (b) realizzate, ad eccezione della funzione XOR, con la stessa tecnica

Da questo punto di vista, è quindi possibile costruire una rete che abbia un comportamento desiderato andando ad associare un peso adeguato ad ogni arco. Questo approccio, però, pur essendo idealmente percorribile, non è praticabile perché il numero di archi con peso non nullo è normalmente molto elevato. Quindi, per la costruzione di una rete, si ricorre spesso all'**apprendimento**, che prevede che i pesi della rete vengano ottenuti partendo dalle caratteristiche di alcuni esempi. Si dice, quindi, che una rete viene **addestrata** fornendole degli esempi che vengono raccolti in un **training set**.

Si noti che, in base alle caratteristiche degli esempi forniti per l'addestramento, si può parlare di due tipologie principali di addestramento:

1. Addestramento **supervisionato**: in cui viene fornito un insieme di esempi e, per ogni esempio, viene anche detto qual è l'uscita attesa per alcune unità di McCulloch-Pitts della rete che rivestono un particolare interesse.

2. Addestramento **non supervisionato**: in cui viene fornito un insieme di esempi e, sfruttandone le caratteristiche, si cerca di ottenere un insieme di pesi adeguato. *

L'addestramento consente di ottenere un insieme adeguato di pesi in grado di garantire che la rete si comporti bene nei casi d'esempio. Però, per valutare quanto bene la rete si comporti in situazioni non ricomprese negli esempi forniti, le prestazioni di una rete vengono sempre valutate utilizzando un secondo insieme di esempi detto **test set**. Training set e test set devono essere disgiunti ed è possibile dire che una rete **generalizza** se ha un comportamento adeguato anche per gli esempi del test set.

Quindi, come spesso si dice, una rete neurale non viene programmata ma viene addestrata ad avere dei comportamenti. Il **machine learning** (o apprendimento automatico) è quella disciplina che si occupa di studiare questo approccio alla sintesi di sistemi di calcolo, sia per reti neurali che per altri tipi di sistemi di calcolo. Quando le informazioni imparate mediante le tecniche del machine learning sono di tipo numerico, o riconducibili a tali, allora si parla di intelligenza artificiale **subsimbolica**.

* esempio che spiega cosa vuol dire che sfrutta le caratteristiche: si prende l'input (vettore di n.ri R dato che sono gli stati di alcuni nodi), si mettono in questo spazio R^n e ci si accorge che gli input sono tutti raggruppati vicini da una parte e tutti raggruppati vicini dall'altra parte. Quando sono vicini da una parte la rete li deve considerare tutti simili, quando sono vicini tutti dall'altra parte la rete li deve considerare tutti simili. La rete non deve considerare simili quelli da un gruppo o dall'altro. L'alg di addestramento, quello che fa è prendere i valori e cercare di capire quando sono abbastanza simili e trovare dei pesi che fanno in modo che se sono simili, l'uscita ha un certo valore; se sono simili dall'altra parte ha un altro valore, ma non glielo si dice dall'esterno qual è l'uscita, non gli si dice nemmeno i cluster che si studiano, è l'alg, per come è fatto, che prende gli esempi e valuta. Il NON SUP. non prevede l'oracolo che dice qual è l'uscita corretta.

L'alg. di back propagation non si applica a tutte le reti neurali ma alle reti neurali in avanti.

insieme di alberi disgiunti, in cui l'uscita è la radice di ogni albero. Più uscite ho e più la rete sarà in grado di produrmi valori in $\mathbb{R}^m$

2 Single-Layer Perceptron e Approssimazione di Funzioni

Una rete neurale **in avanti** (o **feed forward**) è un grafo orientato aciclico pesato organizzato a **strati** (o **livelli**). I nodi del grafo sono unità di McCulloch-Pitts e valgono le seguenti proprietà:

1. Gli strati sono numerati;
2. Ogni nodo appartiene ad un solo strato;
3. I nodi del primo strato non hanno archi entranti e il valore di uscita delle relative unità di McCulloch-Pitts viene fornito dall'esterno;
4. Ogni nodo, tranne quelli del primo strato, ha archi entranti che provengono solo dai nodi dello strato precedente;
5. I nodi dell'ultimo strato non hanno archi uscenti e il valore di uscita delle relative unità di McCulloch-Pitts viene fornito all'esterno;
6. I pesi sugli archi sono i pesi entranti nelle unità di McCulloch-Pitts; e
7. Uno dei valori di ingresso per ogni strato della rete viene fissato al valore di bias -1 e non riceve archi entranti.

Se non viene esplicitamente indicato il contrario, si assume che una rete neurale in avanti abbia il massimo numero di connessioni che le consentono di mantenere la struttura a strati. Si noti che gli strati diversi dal primo e dall'ultimo vengono normalmente chiamati **strati nascosti**. Infine, si noti che una rete neurale in avanti viene anche chiamata **perceptron** (o **percettrone**) e, quando la struttura della rete non è a strati, spesso si parla di rete neurale **ricorrente**, anziché semplicemente di rete neurale.

Ad esempio, in Figura 4 viene mostrata una rete neurale in avanti con tre ingressi forniti da tre delle unità di McCulloch-Pitts del primo strato, quattro unità di McCulloch-Pitts nel primo strato nascosto, quattro unità di McCulloch-Pitts nel secondo strato nascosto e un'uscita fornita dall'unica unità di McCulloch-Pitts dell'ultimo strato. Questa rete ha complessivamente nove unità di McCulloch-Pitts che compiono elaborazioni, quattro nel primo strato nascosto, quattro nel secondo strato nascosto e una nello strato di uscita.

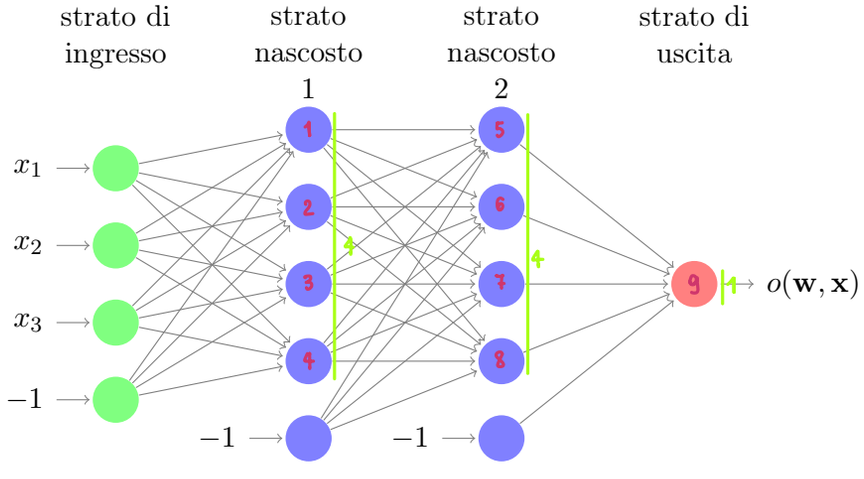

Figura 4: Esempio di una rete neurale in avanti

formata da 3 strati (lo strato di input non fa calcoli quindi non si conta)

multi layer perceptron 4-4-1

Un **Single-Layer Perceptron** (**SLP**) è una rete neurale in avanti con $n \in \mathbb{N}_+$ ingressi reali, nessuno strato nascosto, $m \in \mathbb{N}_+$ uscite reali e una funzione di attivazione $g : \mathbb{R} \to \mathbb{R}$. Si noti che si utilizza la notazione $(n, m)-$SLP per indicare esplicitamente n e m, mentre si parla semplicemente di SLP se n e m sono noti dal contesto. Come discusso nel seguito, un $(n, m)-$SLP può essere utilizzato come un approssimatore di una funzione $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ di cui si conoscano i valori per alcuni vettori. Alcuni dei vettori per cui la funzione è nota vengono quindi raccolti in un training set, e i rimanenti vengono raccolti in un test set, in modo da poter utilizzare l'addestramento supervisionato per trovare i pesi del SLP in grado di approssimare la funzione oggetto di studio. Si noti subito che il numero di ingressi n e il numero di uscite m fissano immediatamente la struttura del SLP e quindi è ragionevole ritenere che un SLP non possa approssimare una funzione arbitraria da $\mathbb{R}^n$ a $\mathbb{R}^m$. Figura 5 mostra un SLP con quattro ingressi e un'uscita.

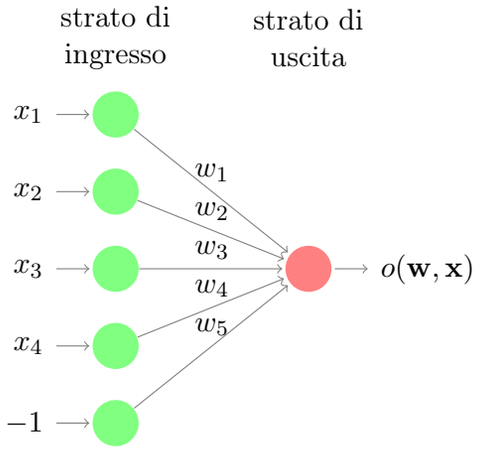

Figura 5: Esempio di un $(4, 1)-$SLP

Si consideri un $(n-1, 1)-$SLP e, per semplificare la notazione, senza comunque perdere di generalità, si ipotizzi che tutti i vettori di $\mathbb{R}^n$ utilizzati come ingressi del SLP, compresi quelli raccolti nel training set e nel test set, siano del tipo

$$\mathbf{x} = (x_1, x_2, \ldots, x_{n-1}, x_n = -1) \tag{10}$$

Naturalmente, se si vuole effettivamente disporre di n valori di ingresso per il SLP, sarà sufficiente considerare un $(n, 1)-$SLP.

Dato un vettore di ingresso $\mathbf{x} \in \mathbb{R}^n$, il valore calcolato dal SLP per un certo vettore dei pesi $\mathbf{w} \in \mathbb{R}^n$ vale

$$o(\mathbf{w}, \mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x}) = g\left(\sum_{i=1}^{n-1} w_i \, x_i - w_n\right) \tag{11}$$

perché l'ultima componente del vettore $\mathbf{x}$, quindi x_n, consente di trattare l'ultima componente di $\mathbf{w}$, quindi w_n, come peso di bias.

L'addestramento supervisionato ha lo scopo di trovare un vettore dei pesi del SLP in modo da minimizzare la distanza media tra il valore calcolato dal SLP e il corrispondente valore della funzione da approssimare. Si noti che il valore della funzione da approssimare è noto per ipotesi per tutti i vettori del training set, appunto perché si sta considerando un addestramento supervisionato. In più, si noti che un vettore dei pesi che minimizza la distanza media tra il valore calcolato dal SLP e il corrispondente valore della funzione da approssimare è tale da minimizzare le singole distanze tra il valore calcolato dal SLP

e il corrispondente valore della funzione da approssimare perché le distanze sono sempre positive o nulle. Quindi, è sufficiente cercare di minimizzare la distanza tra il valore calcolato dal SLP e il corrispondente valore della funzione da approssimare per ogni vettore del training set per trovare un vettore dei pesi adeguato.

Detto $\mathbf{x} \in \mathbb{R}^n$ uno dei vettori nel training set, l'errore compiuto dal SLP nell'approssimazione della funzione f per lo specifico $\mathbf{x}$ vale

fissato x e f(x)

$$\epsilon(\mathbf{w}, \mathbf{x}) = f(\mathbf{x}) - o(\mathbf{w}, \mathbf{x}) = f(\mathbf{x}) - g(\mathbf{w} \cdot \mathbf{x}) \tag{12}$$

Si noti che l'approssimazione della funzione f per il vettore $\mathbf{x}$ è tanto migliore quanto $|\epsilon(\mathbf{w}, \mathbf{x})|$ è piccolo. Quindi, la scelta del vettore dei pesi sarà tanto migliore quanto sarà in grado di rendere $|\epsilon(\mathbf{w}, \mathbf{x})|$ piccolo. In sintesi, fissato un vettore $\mathbf{x}$, il problema di individuare un vettore dei pesi $\mathbf{w}$ in grado di approssimare bene $f(\mathbf{x})$ può essere espresso mediante la ricerca di un $\mathbf{w}$ che minimizzi $|\epsilon(\mathbf{w}, \mathbf{x})|$.

Con l'obiettivo di utilizzare le comuni tecniche di ricerca dei minimi di funzioni reali in più variabili reali, normalmente si assume che la funzione di attivazione g sia di classe $C^\infty(\mathbb{R})$ e si cerca il minimo dell'**errore quadratico**

a noi interessa il valore assoluto di ε, un modo per minimizzare il v. assoluto di qualcosa è minimizzarne il quadrato, quindi anziché utilizzare il v. assoluto (che da problemi in 0 con le derivate) usiamo ε²: è un modo per studiare un f.ne diversa che ha la caratteristica "è minima dove è minimo il v. assoluto" in comune con il v. ass. e ε² diventa derivabile

$$e(\mathbf{w}, \mathbf{x}) = \frac{1}{2}\epsilon^2(\mathbf{w}, \mathbf{x}) \tag{13}$$

ci permettiamo di usarlo perché serve unicamente a mandarlo via quando calcoliamo le derivate. Non serve a niente e lo si mette così che non rimanga un 2x nella formula finale

cerchiamo il vettore w per cui $\frac{1}{2}\epsilon^2(x,w)$ risulta minima. Non ci sarà un unico minimo, non necessariamente si troverà il min. globale

anziché di $|\epsilon(\mathbf{w}, \mathbf{x})|$. Naturalmente, un vettore dei pesi $\mathbf{w}$ in grado di rendere minimo $e(\mathbf{w}, \mathbf{x})$ è anche in grado di rendere minimo $|\epsilon(\mathbf{w}, \mathbf{x})|$.

In sintesi, fissato un vettore $\mathbf{x} \in \mathbb{R}^n$, l'errore quadratico è una funzione di $\mathbf{w}$ di cui si può cercare il minimo mediante l'algoritmo di discesa del gradiente. Quindi, impostando il problema di approssimare f in questi termini, è necessario esprimere il gradiente di $e(\mathbf{w})$ in modo da utilizzarlo per aggiornare i pesi del SLP mediante la regola

coefficiente di apprendimento

il nuovo vettore delle variabili lo otteniamo sottraendo alpha per il gradiente dell'errore quadratico medio nello stesso vettore di partenza

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla e(\mathbf{w}) \tag{14}$$

dove $\alpha \in \mathbb{R}_+$ prende il nome di **coefficiente di apprendimento** (o **learning rate**).

abbastanza piccolo da garantire la convergenza da qualche parte; abbastanza grande da non richiedere milioni di cicli per arrivare al primo minimo locale

Per potere applicare l'algoritmo di discesa del gradiente è necessario esprimere il gradiente di $e(\mathbf{w})$ in funzione di $\mathbf{w}$ mediante il calcolo delle n derivate parziali di $e(\mathbf{w})$. Quindi, fissato $1 \leq i \leq n$, quello che interessa è

ID1

$$\frac{\partial e}{\partial w_i}(\mathbf{w}) = \frac{1}{2} \cdot 2 \cdot \epsilon(\mathbf{w}) \frac{\partial \epsilon}{\partial w_i}(\mathbf{w}) = \epsilon(\mathbf{w}) \frac{\partial \epsilon}{\partial w_i}(\mathbf{w}) \tag{15}$$

ma

derivata di epsilon

l'uscita prodotta dall'unità di McCulloch Pitts una volta fissato w e x

$$\frac{\partial \epsilon}{\partial w_i}(\mathbf{w}) = -\frac{\partial o}{\partial w_i}(\mathbf{w}) \tag{16}$$

perché $f(\mathbf{x})$ non dipende dal vettore dei pesi. Quindi, ricordando che l'uscita del SLP dipende unicamente da $\mathbf{w}$ perché $\mathbf{x}$ è fissato, si ottiene

$$\frac{\partial o}{\partial w_i}(\mathbf{w}) = \frac{\partial g}{\partial w_i}(\mathbf{w} \cdot \mathbf{x}) = g'(\mathbf{w} \cdot \mathbf{x}) \frac{\partial (\mathbf{w} \cdot \mathbf{x})}{\partial w_i} = g'(\mathbf{w} \cdot \mathbf{x}) x_i \tag{17}$$

La derivata di 0 in wi è la derivata di g applicato al prodotto scalare w·x

derivata della f.ne composta

derivando in wi, gli altri componenti non in posizione i, sono costanti e quindi si possono non considerare

perché

$$\frac{\partial (\mathbf{w} \cdot \mathbf{x})}{\partial w_i} = \sum_{j=1}^{n} x_j \frac{\partial w_j}{\partial w_i} = x_i \tag{18}$$

visto che è semplice constatare che

$$\frac{\partial w_j}{\partial w_i} = \begin{cases} 1 & \text{se } i = j \\ 0 & \text{altrimenti} \end{cases} \tag{19}$$

vedi spiegazione file esterno

Fissato un w_i iniziale, lo correggiamo per ottenere un w_i al ciclo successivo aggiungendo α (coeff. di apprendimento) per epsilon per la derivata di g in w_i (= il componente $i$esimo del gradiente)

Questo algoritmo ci dice che ci muoveremo andando nella discesa del gradiente (verso dove diminuisce), ottenendo nuovi pesi tutte le volte, e ogni volta, avendo un'approssimazione migliore della f.ne che stiamo cercando di approssimare.

Per questo alg. modifichiamo il criterio d'arresto: non più quando il gradiente è molto prossimo a 0, ma ci fermiamo quando l'errore medio compiuto su tutto il training set è abbastanza piccolo. Questo NON va bene, fissiamo l'arresto dopo tot epoche (=giro completo sul training set)

Quindi, in sintesi, fissato $\mathbf{x} \in \mathbb{R}^n$ nel training set è possibile aggiornare il vettore dei pesi del SLP in modo da spostare i pesi verso un minimo locale dell'errore compiuto nell'approssimazione di f usando la regola

$$w_i \leftarrow w_i + \alpha\, \epsilon(\mathbf{w})\, g'(\mathbf{w} \cdot \mathbf{x})\, x_i = w_i + \alpha\, \left(f(\mathbf{x}) - g(\mathbf{w} \cdot \mathbf{x})\right)\, g'(\mathbf{w} \cdot \mathbf{x})\, x_i \tag{20}$$

per $1 \leq i \leq n$ e assumendo che l'ultima componente di $\mathbf{x}$ valga -1.

Si noti che la formula per aggiornamento dei pesi di un SLP è spesso espressa nell'ipotesi che la funzione di attivazione g sia la funzione sigmoide

$$S(x) = \frac{1}{1 + \mathrm{e}^{-x}} \tag{21}$$

perché vale la seguente proprietà della derivata della funzione sigmoide

$$S'(x) = S(x)(1 - S(x)) \tag{22}$$

e quindi

$$S'(\mathbf{w} \cdot \mathbf{x}) = S(\mathbf{w} \cdot \mathbf{x})\,(1 - S(\mathbf{w} \cdot \mathbf{x})) = o(\mathbf{w}, \mathbf{x})(1 - o(\mathbf{w}, \mathbf{x})) \tag{23}$$

che permette di aggiornare i pesi senza dover necessariamente valutare esplicitamente la derivata della funzione di attivazione.

Aver esplicitato la formula per l'aggiornamento dei pesi di un SLP in modo da cercare di approssimare una funzione f di cui si conoscano alcuni valori permette di esplicitare l'algoritmo di addestramento supervisionato per un SLP. Questo algoritmo è descritto nell'Algoritmo 2 e prevede di partire da un vettore dei pesi iniziale, tipicamente casuale, aggiornando questo vettore dei pesi per ogni vettore del training set. In particolare, l'algoritmo descritto nell'Algoritmo 2 lavora su un $(n-1, m)-$SLP e ha i seguenti parametri:

- T: il training set formato da coppie $(\mathbf{x}, f(\mathbf{x}))$;
- α: il coefficiente di apprendimento;
- δ: la tolleranza accettata sull'errore medio; e
- ρ: il numero massimo di epoche.

L'algoritmo termina producendo un vettore dei pesi in grado di approssimare la funzione di cui si conoscono alcuni valori quando:

1. L'errore medio calcolato su tutto il training set è sufficientemente piccolo; oppure
2. È stato raggiunto un numero massimo di **epoche** (di apprendimento), dove un'epoca non è altro che un'iterazione completa dell'algoritmo su tutto il training set.

Si noti che la terminazione causata dal raggiungimento del numero massimo di epoche non garantisce che l'algoritmo trovi sempre un'approssimazione sufficientemente buona della funzione. Però, è necessario introdurre questa condizione di terminazione perché non è possibile garantire che esista un vettore dei pesi in grado di rendere l'errore medio sul training set sufficientemente piccolo. In altri termini, non è possibile garantire che un SLP sia in grado di approssimare bene quanto si vuole una qualsiasi funzione. Come caso estremo, ad esempio, si può considerare il tentativo di utilizzare un SLP per approssimare la funzione XOR dell'algebra di Boole, come discusso nell'Esercizio 1.

Si noti che queste condizioni di terminazione non sono quelle dell'algoritmo di discesa del gradiente comunemente utilizzato e descritto nell'Algoritmo 1. Infatti, queste condizioni

Algoritmo 2 Algoritmo di addestramento supervisionato per un $(n-1,1)-$SLP

function SLP_LEARN(T, α, δ, ρ)
  randomize $\mathbf{w} \in \mathbb{R}^n$ ▷ inizializza $\mathbf{w}$
  for $1 \leq r \leq \rho$ **do** ▷ ripeti per non più di ρ epoche
    $\bar{\epsilon} \leftarrow 0$ ▷ inizializza l'errore medio sul training set
    for $(\mathbf{x}, f(\mathbf{x})) \in T$ **do** ▷ per ogni vettore del training set
      $a \leftarrow \mathbf{w} \cdot \mathbf{x}$ ▷ inizializza $a \in \mathbb{R}$
      for $1 \leq i \leq n$ **do** ▷ per ogni componente di $\mathbf{x}$
        $w_i \leftarrow w_i + \alpha\,(f(\mathbf{x}) - g(a))\,g'(a)x_i$ ▷ aggiorna il peso w_i
      end for
      $\bar{\epsilon} \leftarrow \bar{\epsilon} + \frac{|f(\mathbf{x}) - g(\mathbf{w}\cdot\mathbf{x})|}{|T|}$ ▷ aggiorna l'errore medio sul training set
    end for
    if $\bar{\epsilon} < \delta$ **then** ▷ se l'errore medio è sufficientemente piccolo
      return w ▷ $\mathbf{w}$ è il vettore dei pesi cercato
    end if
  end for
  return w ▷ $\mathbf{w}$ è il vettore dei pesi cercato dopo ρ epoche
end function

di terminazione si adattano meglio all'addestramento supervisionato visto che si basano sull'errore compiuto nell'approssimazione e non sul gradiente di questo errore. Infine, si noti che, alle volte, queste condizioni di terminazione vengono sostituite da altre che tengono conto delle caratteristiche specifiche della funzione che si sta cercando di approssimare.

L'algoritmo descritto nell'Algoritmo 1 può essere facilmente esteso per utilizzare un $(n,m)-$SLP, tipo quello mostrato in Figura 6, per approssimare una funzione $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$. Per farlo è sufficiente disporre di un vettore dei pesi per ognuna delle m uscite richieste. In questo caso, gli m vettori dei pesi richiesti vengono raggruppati in un $2-$**tensore**, che non è altro che un elemento di $(\mathbb{R}^{n+1})^m$. Quindi, si potrà usare un $2-$tensore $W = (\mathbf{w}_i)_{i=1}^m$ le cui m componenti $\mathbf{w}_i \in \mathbb{R}^{n+1}$, con $1 \leq i \leq m$, sono i vettori dei pesi necessari ad approssimare la funzione $\mathbf{f}$. Si noti che, una volta introdotta la nomenclatura dei tensori, è comune indicare i vettori di $\mathbb{R}^n$, per un qualche $n \in \mathbb{N}_+$ con il termine $1-$tensori.

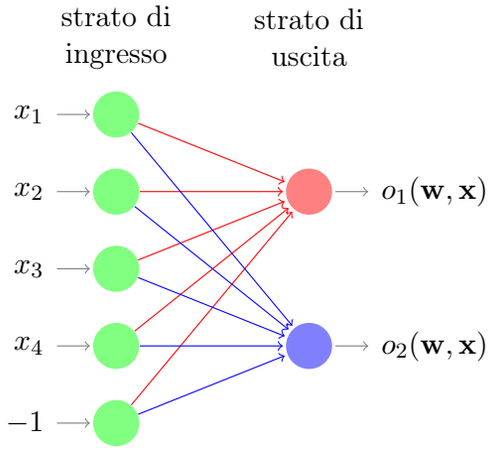

Figura 6: Esempio di un $(4,2)-$SLP