UNIVERSITA' DI PADOVA

FACOLTA' DI INGEGNERIA

Corso di Laurea Triennale in Ingegneria dell'Informazione

Tesina di laurea triennale

INTRODUZIONE ALLA TEORIA DELLE RETI NEURALI

Relatore: Ch.mo Prof. SANDRO ZAMPIERI

Laureanda: MARTINA FISCHETTI

ANNO ACCADEMICO 2010-2011

Indice

Introduzione

L'apprendimento ha da sempre affascinato ricercatori di tutte le discipline: psicologia, filosofia, neurologia, automazione, informatica. Esso può essere definito come la capacità di riconoscere una struttura di interesse all'interno di un contesto: riconoscere un volto in una foto, leggere una targa fotografata, estrapolare un testo da un'immagine.

Per una persona risolvere problemi di questo tipo è estremamente naturale: nonostante abbia una velocità di computazione di pochi millisecondi il cervello umano risolve problemi visivi e linguistici molto più velocemente di un computer (che ha una capacità computazionale dell'ordine dei microsecondi). Già solo da questa osservazione risultano evidenti le enormi potenzialità di un computer in grado di imitare il funzionamento del cervello umano in situazioni di questo tipo. Su questa idea sono basati i metodi delle reti neurali artificiali (Artificial Neural Network- ANNs).

Le ANNs sono ispirate alla biologia e utilizzano metodologie di elaborazione dei dati analoghe a quelli dei neuroni: un modello ANNs impara dalle esperienze, le generalizza a nuovi input ed è in grado di prendere decisioni. Per la loro capacità di lavorare in parallelo, i modelli ANN sono inoltre molto utilizzati per la comprensione di immagini.

Un'applicazione dell'apprendimento automatico è la comprensione di un'immagine, cioè il riconoscimento di oggetti nella scena o il riconoscimento di testi (pattern recognition). Questo sarà l'esempio di applicazione da cui partiremo.

In questo contesto un vettore contenente le informazioni visive immediate (per esempio la tonalità di grigio di ogni pixel) viene mappato in uno spazio di nuove variabili chiamate *features* ed il processo di decisione ha a che fare con la partizione di questo spazio.

Una rete neurale artificiale è formata da un gran numero di neuroni o semplici processori (a cui ci si riferisce talvolta col nome di "unità", "nodi" o appunto "neuroni")

Un neurone artificiale imita le caratteristiche di un neurone biologico: ogni cellula del sistema nervoso umano è in grado di ricevere, elaborare e trasmettere segnali elettrici. Esso è costituito da quattro parti fondamentali: il corpo della cellula, le sinapsi, l'assone e i dendriti (Figura1). I dendriti ricevono informazioni elettriche da altri neuroni attraverso le sinapsi, e le trasmettono al corpo della cellula. Qui essi vengono sommati e se l'eccitazione

totale supera un limite di soglia la cellula reagisce passando il segnale ad un'altra cellula attraverso l'assone.

In maniera del tutto analoga un neurone artificiale riceve in input vari stimoli, ognuno dei quali è l'output di un altro neurone (Figura 2). Ogni input viene quindi moltiplicato per un peso corrispondente e sommato agli altri per determinare il livello di attivazione del neurone tramite un'altra funzione.

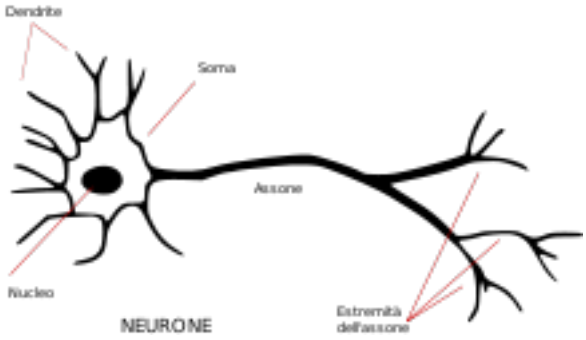


Figura1: Un neurone biologico

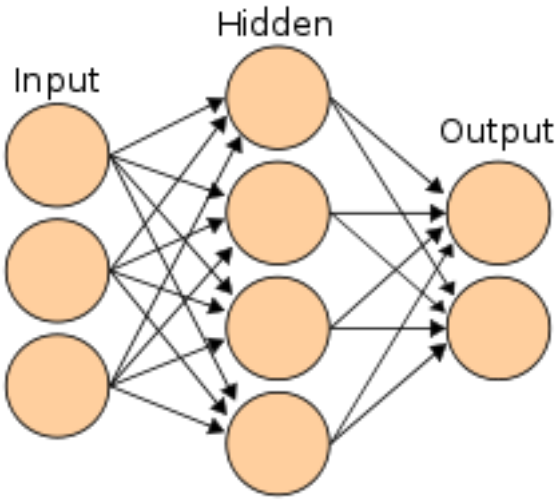
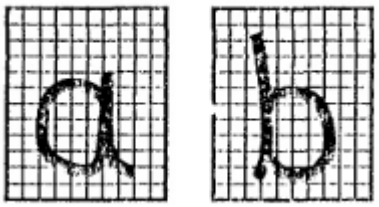


Figura2: Una rete neurale artificiale. La rete riceve segnali esterni su uno strato di nodi d'ingresso (unità di elaborazione), ciascuno dei quali è collegato a numerosi nodi interni, organizzati in più livelli. Ogni nodo elabora i segnali ricevuti e trasmette il risultato a nodi successivi.

1 Un'applicazione: riconoscimento di caratteri

Vogliamo iniziare la nostra trattazione da un esempio per rendere subito evidente come possano ridursi a problemi di classificazione (e ricerca di massimi e minimi) anche problemi che a prima vista possono sembrare di tutt'altro genere.

Supponiamo di avere delle immagini che rappresentino un insieme assegnato di lettere, per esempio "a" e "b", scritte a mano. Il nostro scopo sarà quello di trovare un algoritmo che sia in grado di riconoscere i due caratteri con sufficiente affidabilità. Si osservi che il problema è intrinsecamente probabilistico: nella scrittura a mano vi è una considerevole variazione su come i caratteri vengono tracciati.



Le immagini vengono acquisite da computer come un insieme di pixel. Questi possono essere considerati come un array di valori che rappresentano la tonalità di grigio di ogni singolo pixel. Avremo, per esempio:

$x_i = 0$ se l'i-esimo pixel è completamente bianco

$x_i = 1$ se l'i-esimo pixel è completamente nero.

Possiamo dunque definire il vettore $\mathbf{x} = (x_1, x_2, ..., x_d)^T$ che riunisce tutti i valori x_i(d indica il numero totale di valori e T la trasposizione).

Il problema di riconoscimento di caratteri si traduce dunque in un problema di classificazione: dato il vettore x dobbiamo sviluppare un metodo per discriminare se l'immagine corrispondente a tale vettore appartiene alla classe C_1(cioè se è una "a") o alla classe C_2(cioè è una "b"). Per far questo supponiamo di avere a disposizione un gran numero di immagini sia di "a" sia di "b", e che un operatore umano le abbia già classificate. Questo insieme viene chiamato *data set* o *sample* o *training set*. La nostra macchina dovrà alla fine essere in grado di riconoscere nuove immagini a partire dall'allenamento svolto su tali dati, e produrre un'uscita y che denoti l'insieme di appartenenza del nuovo carattere (nel nostro esempio potrebbe essere $y = 1$ se l'immagine appartiene a C_1, $y = 0$ se l'immagine appartiene a C_2).

Il primo problema che ci si trova ad affrontare è quello della grande mole di dati (un'immagine tipicamente ha 256x256 pixel, cioè d =65 536).

Una tecnica per mitigare questo tipo di problematica è quella di raggruppare la variabili in input per definire un numero più piccolo di variabili dette *features* (caratteristiche). Nel nostro esempio si potrebbe, per esempio, tener conto del rapporto altezza/larghezza, che indicheremo come $\tilde{x_1}$, visto che la "b" è tipicamente più alta e stretta rispetto alla "a". Come si può facilmente intuire $\tilde{x_1}$ da sola non è in grado di classificare perfettamente le due lettere: potremmo trovarci davanti una "b" bassa o una "a" alta e stretta. Pertanto si utilizzano in genere più *features*: come si può osservare confrontando l'istogramma in Figura 1.1 (corrispondente a tenere in considerazione una sola *feature*) e la classificazione di Figura 1.2 (due *features*), più variabili entrano in gioco più la classificazione è precisa. D'altra parte, però, l'aumento di variabili causa velocemente un problema di dimensionalità.

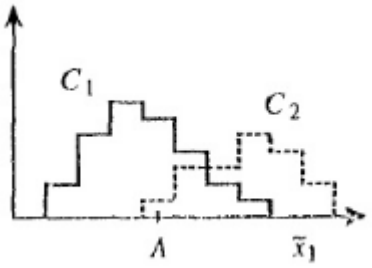


Figura1.1: Istogramma che rappresenta la frequenza associata alla variabile $\tilde{x_1}$ data dal rapporto altezza/larghezza dei caratteri e le classi C_1 per "a" e C_2 per "b". Si noti che la classe C_2 tende ad avere valori di $\tilde{x_1}$ più alti, ma c'è un'ampia regione di sovrapposizione.

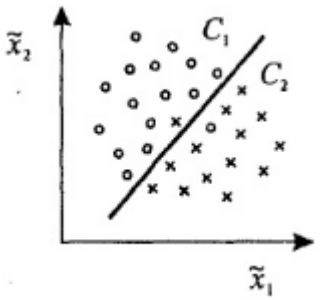
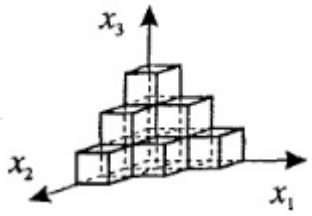


Figura1.2: Un problema di classificazione tra due classi utilizzando due *features.* Il limite decisionale mostrato in figura separa abbastanza bene le due classi anche se ci sono ancora alcuni dati classificati in maniera non corretta. Si noti che, se le due variabili $\tilde{x_1}$ e $\tilde{x_2}$ fossero state considerate singolarmente, la regione di errata classificazione sarebbe stata significativamente più ampia.

Per capire l'impatto che l'aumento della dimensione d può avere sul nostro algoritmo, consideriamo una tecnica di classificazione molto semplice (non consigliabile in pratica) per modellare una mappa non lineare dalle variabili x_i all'output y.

Iniziamo partizionando gli assi corrispondenti ad ogni variabile in modo che il valore di ogni variabile possa essere approssimativamente specificato dicendo in che intervallo esso si trova. Tale divisione corrisponde a dividere lo spazio dell'input in un gran numero di celle.



Ogni esempio del *training set* corrisponde ad un punto in una delle celle ed ha associato un valore dell'output y. Se quindi ci viene dato un nuovo punto nel piano possiamo determinare il corrispondente valore di y individuando in che cella esso si trovi. Più rendiamo fitta la nostra partizione più aumentiamo la precisione con cui l'input viene specificato. Tuttavia, se diciamo che le variabili in input vengano divise in M intervalli, il numero totale di celle sarà M^d. Si ha quindi una crescita esponenziale rispetto alla dimensione dello spazio di input. Poiché ogni cella può contenere al massimo un punto del *training set* , questo implica che il numero di esempi necessari per definire la mappa cresce anch'esso esponenzialmente. Questo fenomeno viene denominato *curse of dimensionality* (Bellman, 1961).

In generale tale problema è meno pesante utilizzando un approccio basato su reti neurali poiché queste riescono a sfruttare due proprietà dei dati in input:

- le variabili in input sono in qualche modo correlate e pertanto i punti

nello spazio non si collocano ovunque ma tendono a raggrupparsi su uno spazio di dimensione minore (*intrinsic dimensionality*)

- le variabili di uscita non variano drasticamente da una zona all'altra dello spazio ma lo fanno in maniera abbastanza lineare rispetto alle variabili di ingresso; è pertanto possibile stimare l'output di zone intermedie dove i dati non sono disponibili.

In molte applicazioni si utilizza comunque un preprocessing per ridurre la dimensione dei dati. La distinzione tra fase di preprocessing e di applicazione del neural network non è sempre netta, ma la prima è caratterizzata tipicamente da una trasformazione fissata delle variabili d'ingresso (ad esempio, nella nostra *feature extraction,* il preprocessing calcolava il rapporto altezza/larghezza), mentre la rete neurale utilizza tecniche più complesse per l'ottimizzazione di parametri adattativi.

2 Apprendimento e curve fitting

Come abbiamo già accennato, l'intero sistema di classificazione di caratteri scritti a mano del nostro esempio può essere visto come una funzione che mappa le variabili di input $x_1, x_2, ..., x_d$ nella variabile di output y che rappresenta la classe di appartenenza dei caratteri. In sistemi più complessi possono esserci più variabili di output y_k con $k = 1, ..., c$.

Supponiamo di voler determinare una forma appropriata per la funzione di mapping con l'aiuto di un *data set* di esempio. La mappa sarà una funzione matematica contenente alcuni parametri adattabili, i cui valori vengono ottimizzati a partire da questi dati. Scriviamo questa funzione come:

$$y = y(\mathbf{x}, \mathbf{w})$$

dove $\mathbf{x} \in \mathbb{R}^d$ è il vettore dei dati in input è $\mathbf{w}$ è il vettore di tali parametri adattabili, chiamati pesi (in inglese *weights*).

Prendendo per esempio $d = 1$ (cioè x scalare) e come y un polinomio di grado M avremo:

$$y(x) = w_0 + w_1 x + ... + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

Tale polinomio può quindi essere considerato come una mappa non lineare che dipende dai valori dei parametri $w_0, w_1, ...w_M$. Indicandoli infatti con il vettore $\mathbf{w} = (w_0, w_1, ..., w_M)$ vediamo che il polinomio è proprio una funzione della forma $y = y(\mathrm{x}, \mathbf{w})$. Il training set ci dà a disposizione una serie di N $\mathbf{x}^1, ..., \mathbf{x}^N$ [1], di cui conosciamo il valore corretto dell'uscita, che chiameremo $t^1, ..., t^N$ (t da *target value*). Nell'apprendimento sfrutteremo questi dati aggiuntivi: il nostro scopo sarà determinare $\mathbf{w}^*$ tale che $y(\mathbf{x}^n, \mathbf{w}^*)$sia il più vicino possibile a t^n per $n = 1...N$. Vogliamo cioè minimizzare la funzione d'errore

$$E = \frac{1}{2} \sum_{n=1}^{N} [y(\mathbf{x}^n, \mathbf{w}) - t^n]^2$$

al variare dei parametri "incogniti" $w_0, w_1, ..., w_M$.

La minimizzazione di una funzione d'errore di questo tipo richiede dei *target values* dell'output, ed è detta *supervised learning* in quanto per ogni input c'è un output desiderato già specificato. Nella sezione 4 ci occuperemo nel dettaglio di un algoritmo (l'algoritmo di Backpropagation) che viene ampiamente utilizzato nelle reti neurali per l'apprendimento supervisionato.

Esistono altre forme di apprendimento di cui noi non ci occuperemo: l'*unsupervised learning* e il *reinforcement learning*. Entrambi sono caratterizzati dal fatto che non necessitano di esempi noti. Per il primo, l'obiettivo è quello di modellare la distribuzione di probabilità dei dati in input o di individuarvi dei *cluster*; il secondo è caratterizzato dal fatto che la scelta del risultato è basata sulla massimizzazione degli incentivi che vengono dati da un agente esterno in corrispondenza di un output buono.

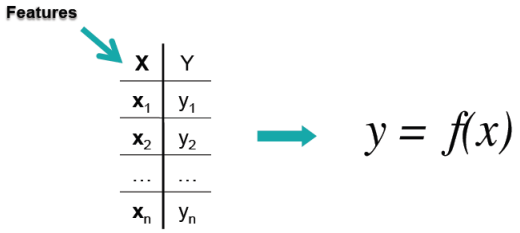


Figura2.1: Esempio di apprendimento supervisionato: si apprende una funzione dell'input a partire da esempi di output

[1] nota bene, nella notazione $\mathbf{x}^n$ n è l'indice non l'esponente. Esso non viene messo come pedice solo per non generare confusione con $x_1, x_2, ..., x_d$che sono le componenti del vettore trattato nel caso di $y_k(\mathbf{x}, \mathbf{w})$generica

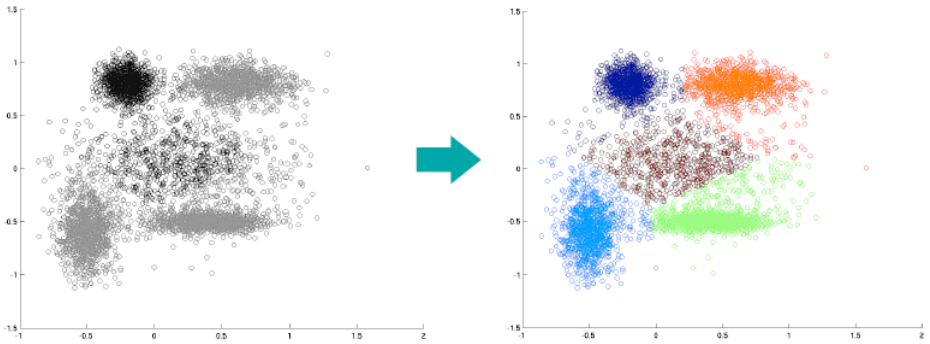


Figura2.2: Esempio di apprendimento non supervisionato: si apprende a riconoscere schemi o cluster nell'input senza alcuna indicazione dei valori di output

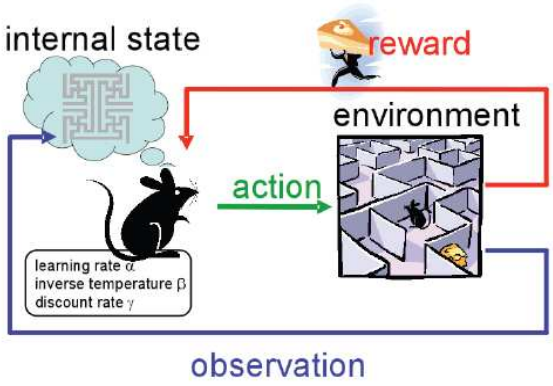


Figura2.3: Esempio di apprendimento per rinforzo: si elabora esplorando l'ambiente e ricevendo ricompense per azioni positive

Una volta allenata con il sistema di apprendimento più consono, la rete neurale sarà in grado di risolvere due tipi di problemi:

- problemi di *classificazione* (Figura 2.4): assegnare un nuovo input ad una classe scelta tra un numero limitato di possibilità (come per il riconoscimento di cifre o caratteri scritti a mano).

- problemi di *regressione* (Figura 2.5): essere in grado di approssimare una funzione continua a partire da alcuni suoi campioni

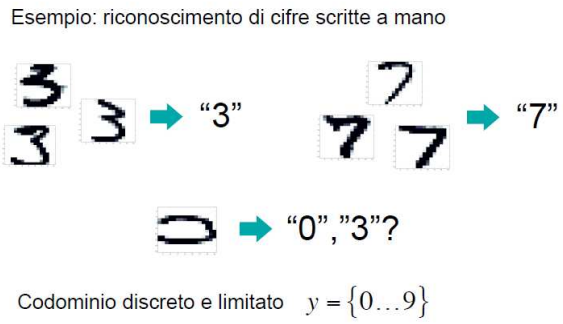


Figura2.4: Un esempio di problema di classificazione

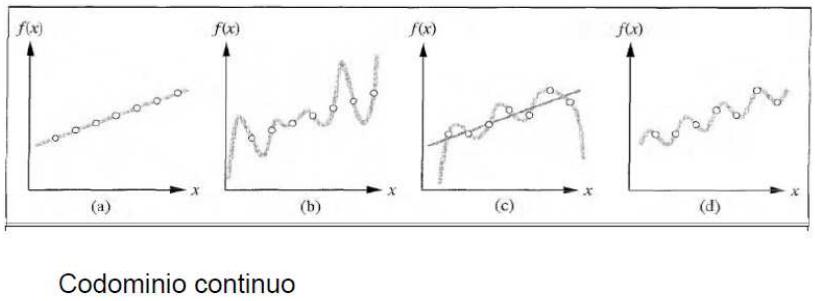


Figura2.5: Un esempio di problema di regressione

Entrambi i problemi possono essere visti come casi particolari di approssimazione di funzioni (come si può intuire dalle immagini sottostanti).

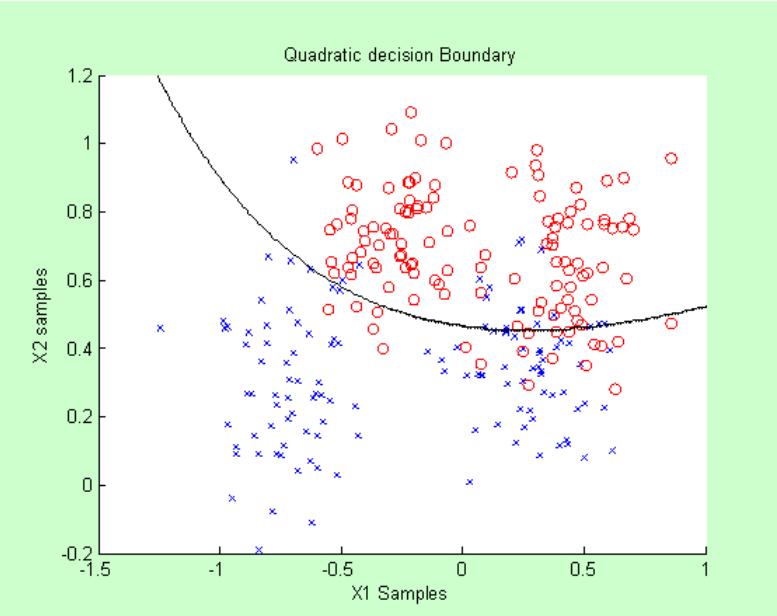


Figura2.6: Nella classificazione il problema è determinare la funzione di limite che meglio divide i dati

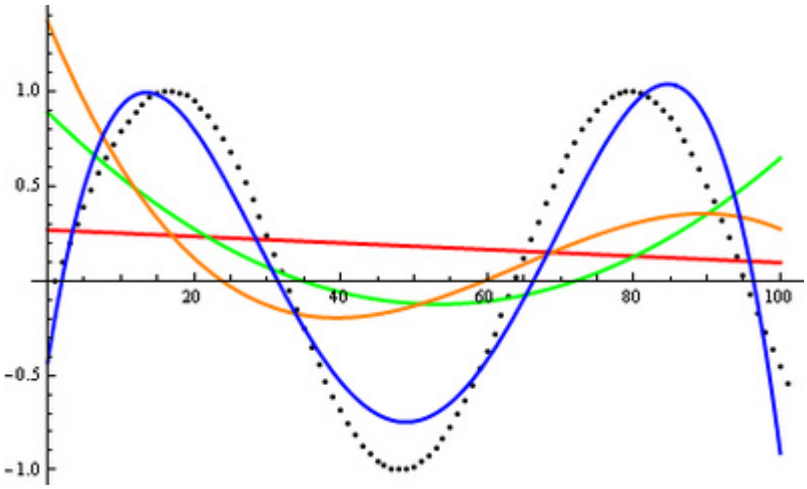
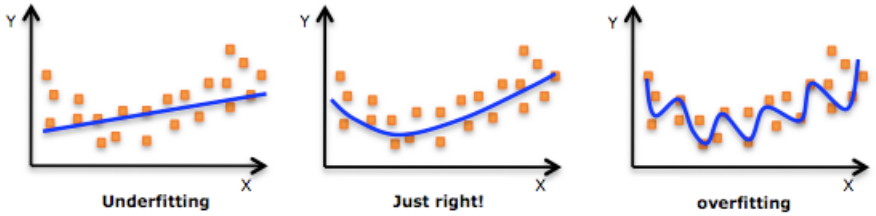


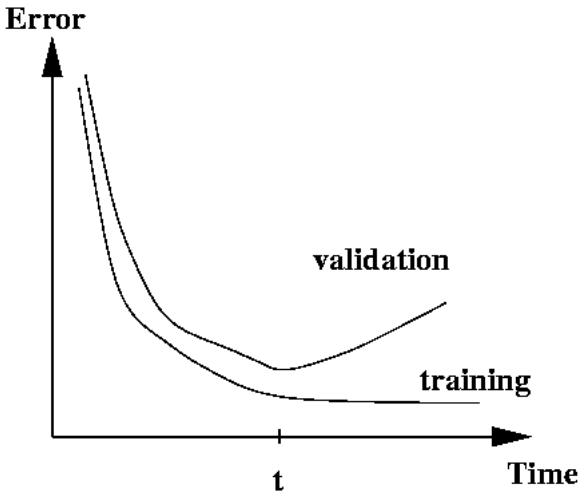
Figura2.7: Nella regressione il problema è determinare la funzione che meglio approssima la funzione nota solo per campioni (in figura le funzioni colorate cercano di approssimare la funzione tratteggiata)

A titolo di esempio illustriamo ora la tecnica del polynomial curve fitting per il problema di regressione.

Una proprietà importante del data set è che esso ha un aspetto sistematico nascosto che vogliamo sia preservato nella nostra funzione approssimante. Per assicurarci che il nostro risultato tenga conto di questo aspetto, possiamo generare i campioni come punti di una funzione *h(x),* affetti da un rumore casuale (nella figura 2.7, per esempio, i punti sono stati ottenuti campionando una sinusoide). In questo modo sapremo che avremo raggiunto il nostro obiettivo quando avremo trovato una funzione "simile" ad *h(x)*. Lo scopo principale del curve fitting nel nostro caso infatti è produrre un sistema in grado di fare buone predizioni per dati nuovi, cioè avere una buona generalizzazione. La miglior generalizzazione per nuovi dati viene ottenuta quando la mappa *y(x)* riesce a catturare la funzione sottostante *h(x)*, e non tanto quando riproduce accuratamente i singoli valori affetti da rumore. Se aumentiamo il grado del polinomio M, e quindi la flessibilità della nostra funzione approssimante, possiamo ottenere un fit più vicino ai dati. Se aumentiamo troppo M però ricadiamo in un problema di overfitting: la funzione approssimante segue perfettamente i dati rumorosi perdendo le caratteristiche di *h(x)*.



Il problema dell'overfitting viene risolto utilizzando una parte dal *training set* come *validation set*. La rete neurale viene allenata usando solo i dati di training, ma tale allenamento viene fermato periodicamente per testare sull'insieme indipendente di validazione le performance della rete finora definita. Questo test non modifica i pesi, ma serve solo a misurare la generalizzazione: *y(x)* approssima *h(x)* i risultati sul validation set miglioreranno con l'apprendimento. Quando invece la rete sta imparando caratteristiche vere solo per gli specifici dati di esempio (overfitting) e non per qualsiasi dato, allora i risultati sul validation test tipicamente peggiorano. Pertanto la tecnica adottata è quella di interrompere l'apprendimento non appena si ha un peggioramento nella performance sul validation set (ciò avviene al tempo t nella Figura che segue).



Concludiamo il paragrafo sottolineando che l'apprendimento supervisionato può essere realizzato anche con metodi diversi da quello delle reti neurali di cui ci occuperemo noi.

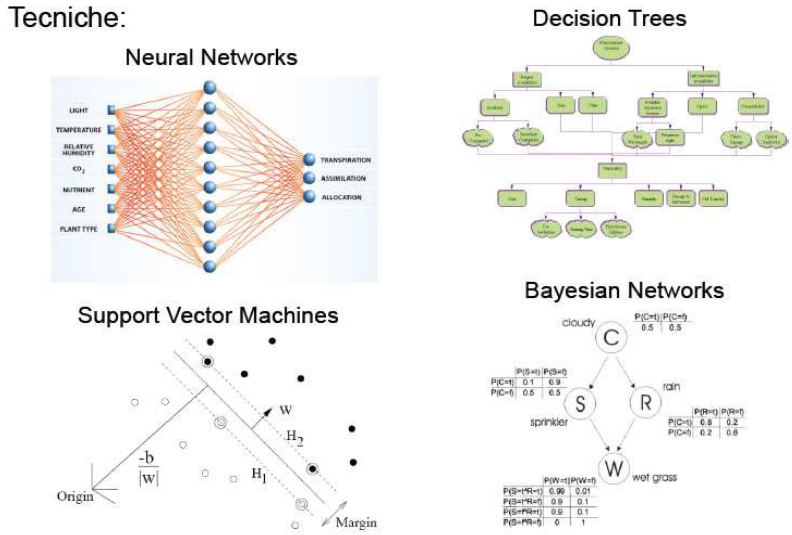


Figura2.8: Altre tecniche di apprendimento supervisionato

3 Reti neurali

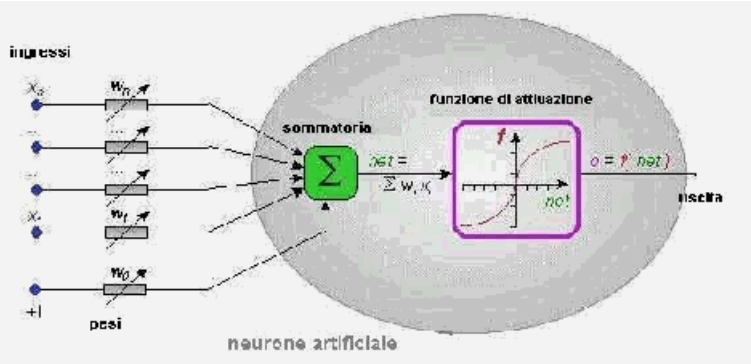
Come è noto le reti neurali sono strutture non lineari di dati organizzate come strumenti di modellazione. Esse possono essere utilizzate per simulare relazioni complesse tra ingressi e uscite che altre funzioni analitiche non riuscirebbero a rappresentare.

Come già accennato nell'introduzione una rete neurale artificiale riceve segnali esterni su uno strato di nodi d'ingresso (unità di elaborazione), ciascuno dei quali è collegato con numerosi nodi interni, organizzati in più livelli. Ogni nodo elabora i segnali ricevuti e trasmette il risultato a nodi successivi.

In questa sezione ci occuperemo di come gli ingressi vengano rielaborati in ogni singolo livello della rete, e di quale sia l'utilità di avere più livelli.

3.1 Funzioni discriminanti

Una funzione discriminante è una funzione che riceve in input i dati d'ingresso e il cui output rappresenta la classificazione di tali dati. L'esempio più semplice di funzione discriminante è la combinazione lineare delle variabili d'ingresso. Tale semplice funzione discriminante può essere generalizzata trasformandone il risultato tramite una funzione non lineare detta funzione d'attivazione (di cui tratteremo successivamente).



Consideriamo inizialmente un problema di classificazione in due classi C_1 e C_2 . In tal caso la funzione discriminante $y(\mathbf{x})$ può decidere a che classe appartiene il vettore $\mathbf{x}$ per esempio nel modo seguente:

$\mathbf{x} \in C_1$ se $y(\mathbf{x}) > 0$

$\mathbf{x} \in C_2$ se $y(\mathbf{x}) < 0$

La funzione discriminante più semplice è

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w^T x} + \mathrm{w}_0$$

dove $\mathbf{w} = (w_1, ...w_d)$ è un vettore d-dimensionale di pesi e w_0 è un termine costante (bias). Per semplicità di notazione nel seguito scriveremo $y(\mathbf{x})$ anziché $y(\mathbf{x}, \mathbf{w})$.

L'interpretazione geometrica della funzione discriminante è la seguente:

il limite decisionale $y(\mathbf{x}) = 0$ corrisponde ad un iperpiano di dimensione (d-1) all'interno dello spazio d-dimensionale delle $\mathbf{x}$. Ciò implica che due generici punti $\mathbf{x^A}$e $\mathbf{x^B}$giacenti su tale piano hanno $y(\mathbf{x^A}) = 0 = \mathrm{y}(\mathbf{x^B})$ e quindi, per definizione di $y(\mathbf{x})$, si ha $\mathbf{w^T}(\mathbf{x^B} - \mathbf{x^A}) = 0$. Questo vuol dire che $\mathbf{w}$ è normale ad ogni vettore dell'iperpiano e quindi definisce l'orientazione del limite decisionale. Osserviamo inoltre che la distanza dell'iperpiano $\{\mathbf{x} : y(\mathbf{x}) = 0\}$ dall'origine è data da

$$l = \frac{|w_0|}{||\mathbf{w}||}$$

Dunque se ne conclude che il bias w_0 determina la posizione dell'iperpiano di delimitazione delle aree di classificazione nello spazio delle $\mathbf{x}$.

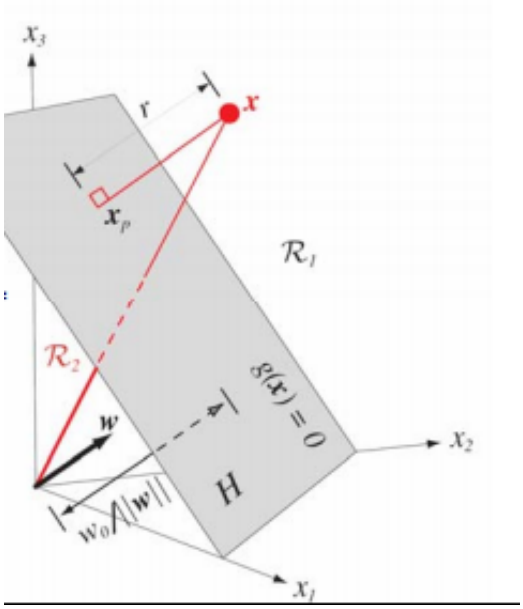


Figura3.1.1: Interpretazione geometrica della funzione discriminante per d=3 ($y(x)$=$g(x)$).

Spesso si utilizza una notazione più omogenea per y(x) cioè

$y(\mathbf{x}) = \tilde{\mathbf{w}}^{\mathbf{T}}\tilde{\mathbf{x}}$

con $\tilde{\mathbf{w}} = (\mathrm{w}_0,\mathbf{w})$ e $\tilde{\mathbf{x}} = (\mathrm{x}_0,\mathbf{x})$ con $x_0 \equiv 1$. La funzione discriminante viene quindi rappresentata nelle reti neurali come in figura 3.1.2: gli input $x_1, x_2, ..., x_d$ vengono indicati con dei nodi di un grafo, connessi tramite archi pesati all'output $y(\mathbf{x})$. Il termine noto w_0 viene rappresentato come peso di $x_0$fissato al valore 1 (indicato in nero)

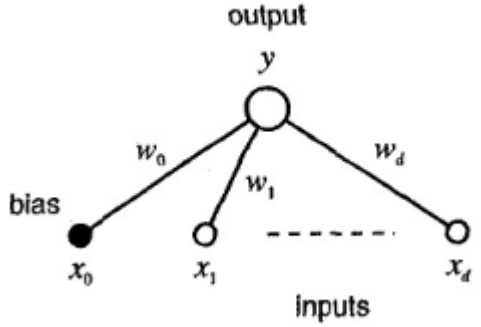


Figura3.1.2: Rappresentazione di una funzione discriminante lineare come un diagramma di rete neurale.

Quanto appena illustrato per due classi può essere esteso a più classi usando una funzione discriminante $y_k(\mathbf{x})$ per ogni C_k,e cioè::

$$y_k(\mathbf{x}) = \mathbf{w_k^T x} + w_{k0}$$

Un nuovo punto $\mathbf{x}$ verrà assegnato alla classe C_k se $y_k(\mathbf{x}) > y_j(\mathbf{x})$ per ogni $j \neq k$.

Il limite di separazione tra le classi C_k e C_j sarà dato da $y_k(\mathbf{x}) = y_j(\mathbf{x})$ ossia dall'iperpiano

$$(\mathbf{w_k} - \mathbf{w_j})^{\mathrm{T}}\mathbf{x} + (w_{k0} - w_{j0}) = 0$$

Analogamente al caso di due classi si osserva quindi che la direzione normale all'iperpiano di separazione è data dalla differenza tra i vettori dei pesi e che la distanza dall'origine è

$$l = \frac{|w_{k0} - w_{j0}|}{||\mathbf{w_k} - \mathbf{w_j}||}$$

Tale funzione discriminante multiclasse può essere rappresentata da un diagramma di rete neurale come mostrato in Figura 3.1.3

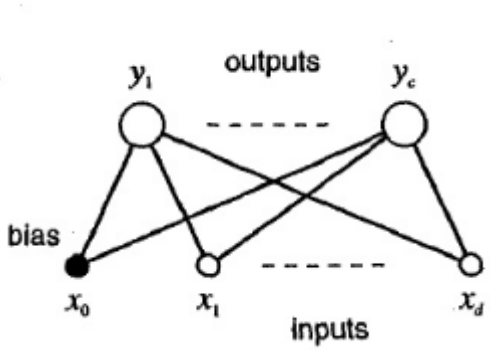


Figura3.1.3: Rappresentazione di una funzione discriminante lineare multiclasse come un diagramma di rete neurale. I cerchi in alto rappresentano le funzioni $y_k(\mathbf{x})$ e sono talvolta chiamate *processing units*.

Possiamo esprimere l'output $y_k(\mathbf{x})$ come:

$$y_k(\mathbf{x}) = \sum_{i=1}^{d} w_{ki}x_i + w_{k0}$$

oppure, introducendo l'extra input $x_0 = 1$, come:

$$y_k(\mathbf{x}) = \sum_{i=0}^{d} w_{ki}x_i$$

Una volta definiti i pesi w_{ki} ottimi, un nuovo vettore verrà classificato applicandovi la funzione sopra descritta e assegnando il vettore alla classe con il massimo valore di attivazione. Le regioni di classificazione così definite sono insiemi convessi (Figura 3.1.4), dato che, per ogni coppia di punti $\mathbf{x^A}$e $\mathbf{x^B}$ nella stessa regione R_k, un generico punto $\hat{\mathbf{x}}$ che giace sul segmento che li unisce è descritto dall'equazione

$$\hat{\mathbf{x}} = \alpha\mathbf{x^A} + (1-\alpha)\mathbf{x^B} \text{ con } 0 \leq \alpha \leq 1$$

Applicando dunque la k-esima funzione discriminante a tale punto avremo

$$y_k(\hat{\mathbf{x}}) = \alpha y_k(\mathbf{x^A}) + (1-\alpha)y_k(\mathbf{x^B})$$

Poiché abbiamo supposto $\mathbf{x^A}, \mathbf{x^B} \in R_k$ vale $y_k(\mathbf{x^A}) > y_j(\mathbf{x^A})$ e $y_k(\mathbf{x}^B) > y_j(\mathbf{x^B})$ per ogni $j \neq k$, si ricava che $y_k(\hat{\mathbf{x}}) > y_j(\hat{\mathbf{x}})$ per ogni $j \neq k$. Ciò vuol dire che ogni punto $\hat{\mathbf{x}}$ che giace sul segmento che unisce due punti di R_k appartiene esso stesso a R_ke quindi $R_k$deve essere un insieme convesso.

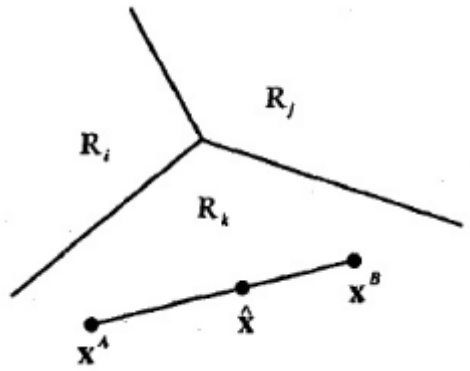


Figura3.1.4: Esempio di limiti decisionali prodotti tramite una funzione discriminante lineare multiclasse. Per ogni coppia di punti $\mathbf{x^A}$e $\mathbf{x^B}$, entrambi nella stessa regione R_k, ogni punto $\hat{\mathbf{x}}$ che giace sul segmento che li unisce appartiene esso stesso a R_k. Le regioni sono insiemi convessi.

3.2 Funzione di attivazione

Finora abbiamo trattato come funzione discriminante una semplice combinazione lineare delle variabili di input. In realtà questa funzione può essere generalizzata in molti modi per esempio tramite una funzione non lineare $g : \mathbb{R} \to \mathbb{R}$. Per il problema di discriminazione tra due classi, per esempio, avremo:

$$y(\mathbf{x}, \mathbf{w}) = g(\mathbf{w^T x} + w_0)$$

Tale g è detta *funzione di attivazione* ed è generalmente monotona. Spesso si utilizzano funzioni di tipo sigmoide, cioè funzioni caratterizzate da un andamento ad S (sigma in greco). Alcuni di possibili esempi di funzioni di attivazione sono mostrati in Figura 3.2.1.

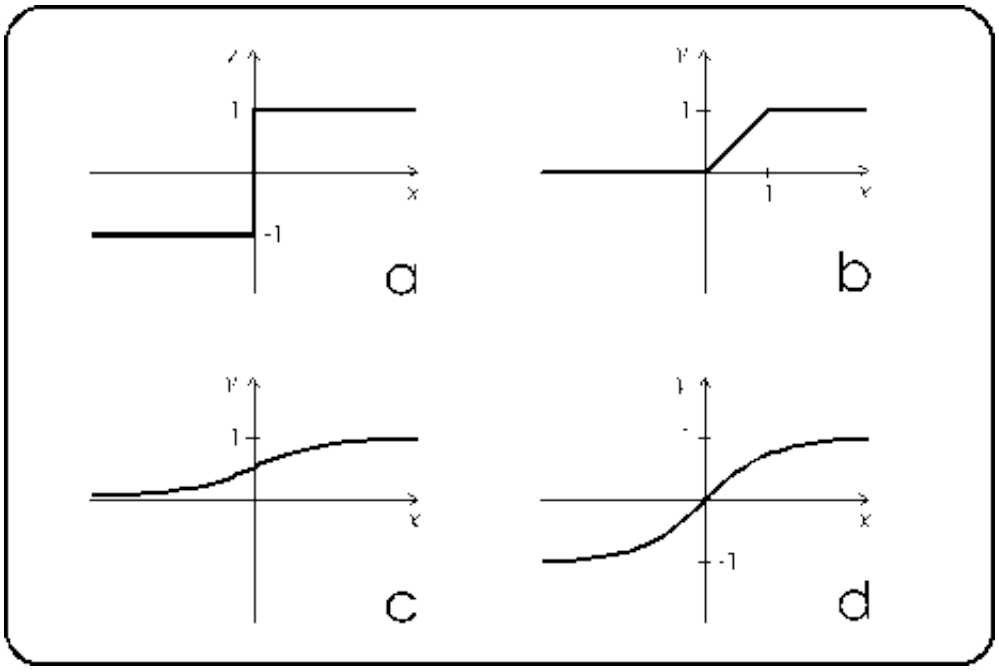


Figura3.2.1: Alcune funzioni di attivazione: a) funzione gradino che può assumere valori bipolari (-1, 1), come quella in figura, o valori binari (0, 1). b) funzione lineare con saturazione c) funzione sigmoide logistica d) funzione sigmoide simmetrica

La più importante tra le funzioni sigmoidee è la *funzione sigmoide logistica.* Questa funzione è molto interessante perché consente di interpretare l'output della funzione discriminante come una probabilità a posteriori.

Consideriamo infatti il problema della classificazione in due categorie e supponiamo che le densità di probabilità condizionate siano date da una gaussiana con matrice di covarianza uguale per le due classi ($\Sigma_1 = \Sigma_2 = \Sigma$), cioè

$$p(\mathbf{x}|C_k) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left[-\frac{1}{2}(\mathbf{x}-\mu_\mathbf{k})\Sigma^{-1}(\mathbf{x}-\mu_k)\right] \qquad (1)$$

Per il teorema di Bayes la probabilità a posteriori è $P(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)P(C_k)}{p(\mathbf{x})}$ con, nel caso in esame, $p(\mathbf{x}) = p(\mathbf{x}|C_1)P(C_1) + p(\mathbf{x}|C_2)P(C_2)$ e quindi

$$P(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)P(C_k)}{p(\mathbf{x}|C_1)P(C_1) + p(\mathbf{x}|C_2)P(C_2)} = \frac{1}{1+\exp(-a)} = g(a) \qquad (2)$$

dove

$$a = \ln \frac{p(x|C_1)P(C_1)}{p(x|C_2)P(C_2)} \qquad (3)$$

e

$$g(a) \equiv \tfrac{1}{1+\exp(-a)}$$

è la funzione sigmoide logistica (Figura 3.2.2).

Inserendo la probabilità condizionata gaussiana della formula (1) nella (3) otteniamo

$$a = \mathbf{w^T x} + w_0$$

con $\mathbf{w} = \Sigma^{-1}(\mu_\mathbf{1} - \mu_\mathbf{2})$
e $w_0 = -\frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + \ln\frac{P(C_1)}{P(C_2)}$

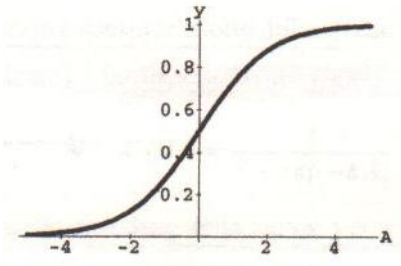


Figura3.2.2: Plot della funzione sigmoide logistica $g(a) \equiv \frac{1}{1+\exp(-a)}$

La più semplice funzione di attivazione è invece la funzione gradino:

$$g(a) = \begin{cases} 0 & a < 0 \\ 1 & a \geq 0 \end{cases}$$

Questa funzione è interessante per la sua interpretazione biologica (da cui deriva il nome di funzioni di attivazione): gli input x_i rappresentano il livello di attività degli altri neuroni connessi al neurone in esame, i pesi w_i

rappresentano la forza delle interconnessioni (cioè delle sinapsi) e il termine noto w_0 è il valore di soglia, superato il quale il neurone si attiva.

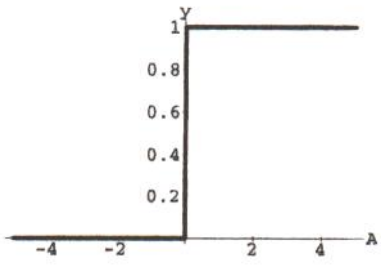


Figura3.2.3: Plot della funzione gradino

3.3 Un limite delle reti neurali monostrato: separabilità lineare.

Come abbiamo visto nella Sezione 3.1 la funzione discriminante di una rete monostrato definisce un iperpiano di divisione tra regioni che sono convesse. Esistono alcuni problemi di classificazione in cui però le regioni non sono di questo tipo, cioè non hanno la proprietà di separabilità lineare.

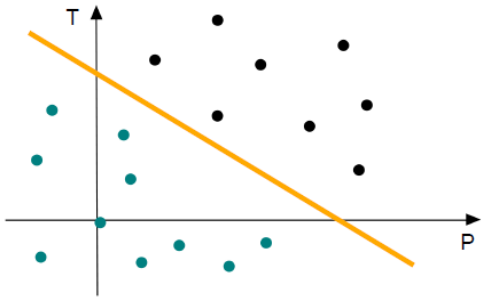


Figura3.3.1: Un problema separabile linearmente

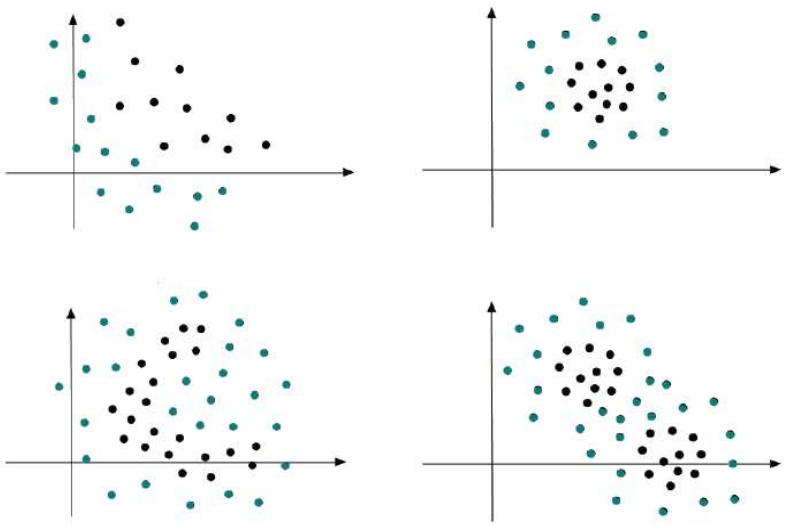
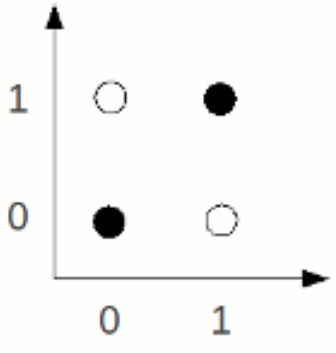


Figura3.3.2: Esempi di problemi non linearmente separabili

I dati non linearmente separabili sono caratterizzati dal fatto che un piccolo cambiamento in input (quindi punti poco distanti nella rappresentazione grafica) causa un gran cambiamento nell'output. Un esempio molto citato in letteratura è quello dello XOR illustrato nella seguente figura:

X	Y	O
0	0	0
0	1	1
1	0	1
1	1	0



Come si vede, il cambiamento di un solo bit, per esempio da (0,0) a (0,1), causa un netto cambiamento nell'output: il primo punto ha come output 0 (classe bianca in figura) e il secondo ha output 1 (classe nera). Risulta inoltre evidente che non è possibile tracciare un'unica linea nel piano bidimensionale per separare i punti bianchi dai punti neri.

Per risolvere questo problema ed essere in grado di mappare correttamente funzioni come lo XOR dobbiamo utilizzare reti a più livelli (Figu-

ra3.3.6). Possono essere considerate rappresentazioni di reti di questo tipo tutti i diagrammi feed-forward, cioè i diagrammi che non contengono anelli all'indietro. Questo infatti assicura che l'uscita del network possa essere calcolata come una funzione esplicita e non ricorsiva degli ingressi.

Come mostra la Figura 3.3.3 una rete multistrato è la concatenazione di reti monostrato come quelle di cui abbiamo parlato finora: ogni livello è basato sulla combinazione lineare delle variabili del livello precedente, trasformato attraverso funzioni di attivazione non lineari.

Le unità interne, cioè quelle che non generano l'output finale del network, vengono dette unità nascoste (*hidden units*)

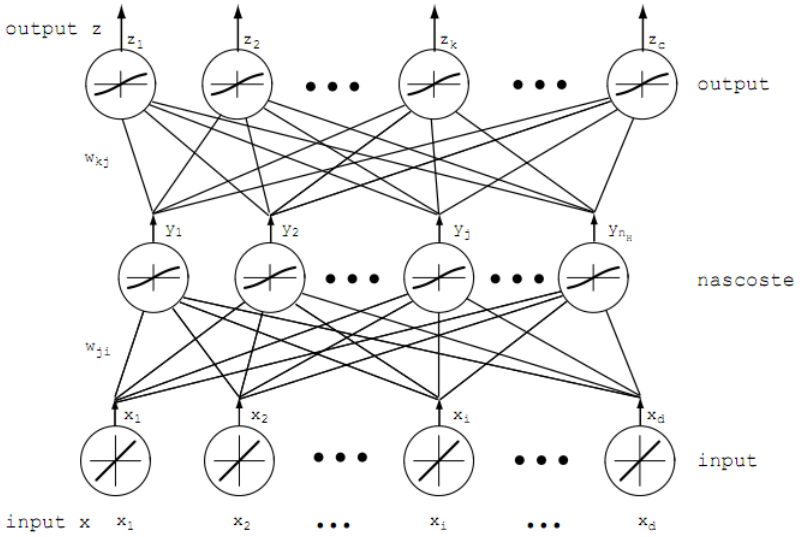


Figura3.3.3: Esempio di diagramma multistrato di una rete neurale

Analizzando per esempio la più semplice rete multistrato, cioè quella avente due strati di pesi adattabili come in Figura 3.3.4, avremo dunque che gli output del diagramma avranno la forma:

$$y_k(\mathbf{x}) = \tilde{g}(\sum_{j=0}^{M} w_{kj}^{(2)} g(\sum_{i=0}^{d} w_{ji}^{(1)} x_i))$$

dove

$\tilde{g}(.)$ è la funzione di attivazione dell'ultimo livello, che può essere anche differente dalla funzione di attivazione $g(.)$ del livello nascosto.

$w_{ji}^{(1)}$indica il peso di primo livello dall'input i-esimo alla j-esima unità nascosta;

$w_{kj}^{(2)}$indica il peso di secondo livello tra la j-esima unità nascosta e l'output k-esimo.

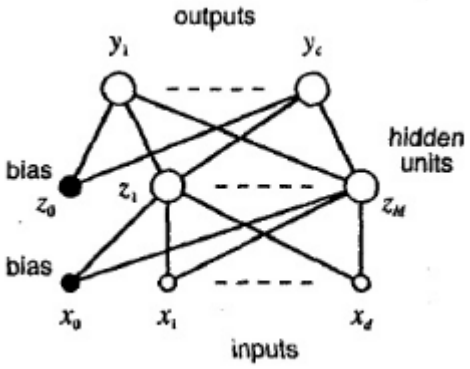


Figura3.3.4: Esempio di diagramma a due livelli di pesi adattabili.$z_j = g(\sum_{i=0}^{d} w_{ji}^{(1)} x_i))$

Le figure seguenti mostrano come le reti multistrato riescano a dividere dati non linearmente separabili:

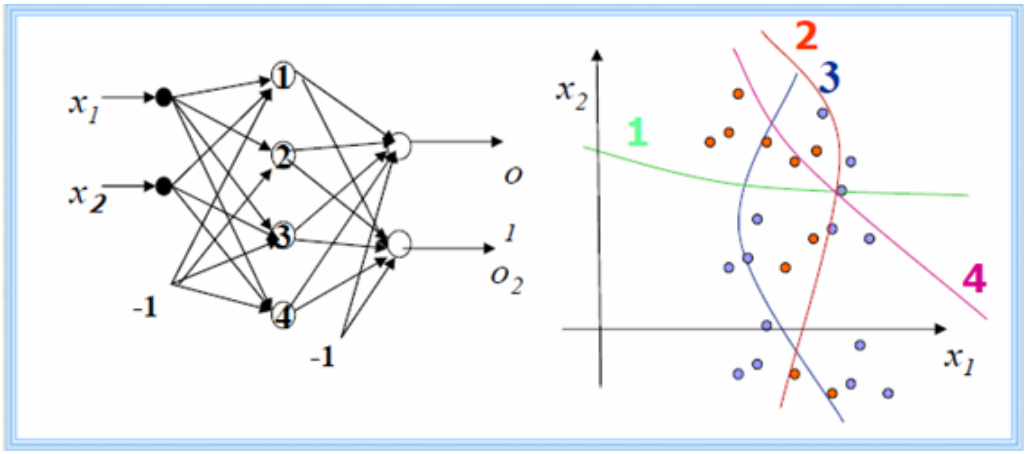


Figura3.3.5: Esempio di classificazione realizzata da una rete multistrato. Gli strati nasconsti (*hidden*) partizionano lo spazio in regioni e gli strati di output combinano i contributi delle varie regioni

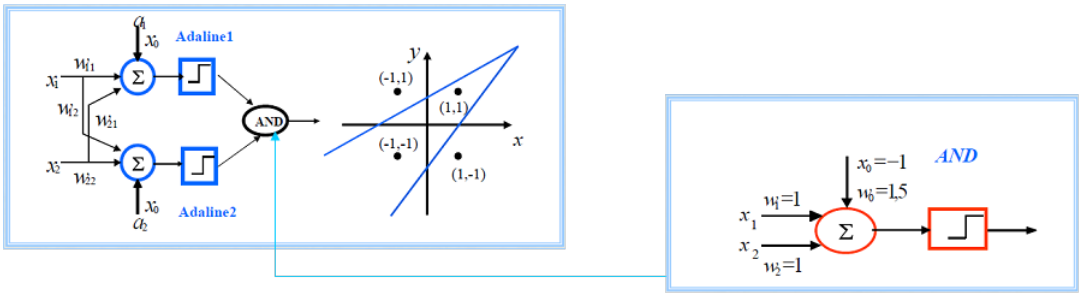


Figura3.3.6: Una possibile soluzione del problema dello XOR tramite rete a due livelli.

4 L'algoritmo di apprendimento Backpropagation

Abbiamo detto che le reti multistrato sono in grado di trattare un numero più grande di funzioni rispetto alle reti a singolo strato. Tuttavia è evidente che lo sforzo necessario per trovare la combinazione ottimale dei pesi cresce significativamente per l'aumento del numero di parametri in gioco e soprattutto per la struttura più complessa della rete (altamente non lineare).

Di seguito ci occuperemo dunque del problema (già accennato nella sezione 2) di minimizzazione dell'errore attraverso un apprendimento supervisionato. Illustreremo in particolare la soluzione tramite l'algoritmo Backpropagation che è ad oggi uno degli algoritmi più studiati ed utilizzati nelle reti neurali.

Per l'illustrazione di tale algoritmo utilizzeremo un approccio grafico in modo da rendere i risultati più generali e più intuitivi rispetto a quelli dell'approccio analitico.

4.1 Riduzione del problema di apprendimento al calcolo del gradiente

L'algoritmo Backpropagation si basa sul metodo della discesa del gradiente per trovare il minimo della funzione d'errore rispetto ai pesi.

La tecnica della discesa del gradiente si basa sul fatto che il gradiente indica la direzione di massima crescita (decrescita se considerato nel verso opposto). Nell'implementazione base, si inizia scegliendo un punto a caso nello spazio multidimensionale (primo punto) e valutando il gradiente in tale

punto. Si procede quindi scegliendo un altro punto (secondo punto) nella direzione di massima decrescita indicata dall'opposto del gradiente (antigradiente). Se la funzione al secondo punto ha un valore inferiore al valore calcolato al primo punto, la discesa può continuare seguendo l'antigradiente calcolato al secondo punto, altrimenti il passo di spostamento viene ridotto e si riparte. L'algoritmo si ferma non appena si raggiunge un minimo locale.

Poiché il nostro algoritmo di apprendimento si basa sul calcolo del gradiente della funzione d'errore, dobbiamo assicurarci che essa sia continua e differenziabile. Pertanto la funzione gradino non sarà una buona funzione d'attivazione (poiché introduce una discontinuità), e si utilizzerà piuttosto una funzione sigmoide (del genere visto nel paragrafo 3.2).

Una funzione d'attivazione differenziabile come queste, infatti, rende differenziabile l'intera funzione calcolata dalla rete neurale (assumendo che la funzione discriminante ad ogni nodo sia la somma degli input), poiché la rete non fa che comporre funzioni differenziabili. Ne consegue che l'errore sarà esso stesso differenziabile.

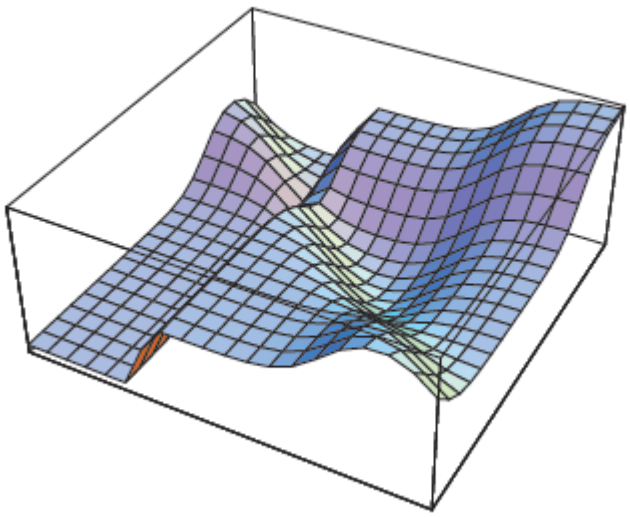


Figura4.1.1 Minimo locali e globali per la funzione d'errore. Se il primo punto della regola del gradiente viene scelto troppo vicino ad un minimo locale l'algoritmo potrebbe non convergere mai al minimo globale.

Il punto chiave dell'algoritmo Backpropagation è il calcolo del gradiente in maniera efficiente.

Per prima cosa dunque dobbiamo estendere la nostra rete in modo che essa calcoli automaticamente la funzione d'errore

$$E = \frac{1}{2}\sum_{i=1}^{N}(o^i - t^i)^2 = \sum_{i=1}^{N} E_i$$

dove t^i è il target value dell'i-esimo elemento del training set $\{(\mathbf{x^1}, t^1), ..., (\mathbf{x^N}, t^N)\}$ e $o^i = y(\mathbf{x^i}, \mathbf{w})$ è l'output prodotto dalla rete quando le viene presentato il valore di allenamento $\mathbf{x^i}$.

La Figura 4.1.2 mostra una rete neurale estesa che dà come output il contributo E_i di $\mathbf{x^i}$ all'errore complessivo E (per generalità in figura viene illustrato il caso multi-output in cui o e t sono vettori)

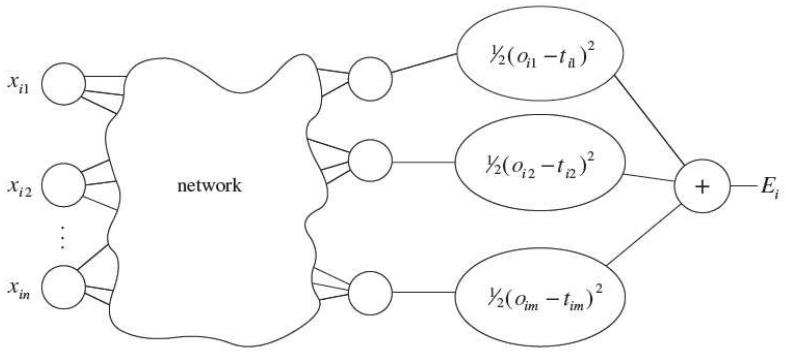


Figura4.1.2 Una rete neurale multistrato estesa al calcolo della funzione d'errore per l'i-esimo input.

Tale estensione va fatta per ogni t^i e la funzione d'errore totale si calcola sommando i risultati $E_1 + ... + E_N = E$. Osserviamo che i pesi della rete sono gli unici parametri che possiamo modificare per rendere l'errore quadratico E più piccolo possibile.

Poiché E viene calcolata dalla rete estesa come composizione delle funzioni dei nodi essa è continua e differenziabile rispetto ai pesi $w_1, ...w_l$.

Sfrutteremo dunque la Backpropagation per calcolare il gradiente

$$\nabla E = (\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, ..., \frac{\partial E}{\partial w_l})$$

e aggiorneremo i pesi usando l'incremento

$$\Delta w_i = -\gamma \frac{\partial E}{\partial w_i} \text{ per } i = 1, ..., l$$

dove γ è la *costante di apprendimento* che definisce ad ogni iterazione la lunghezza del passo che usiamo per muoverci nella direzione di decrescita indicata dall'antigradiente.

4.2 B-diagram

Cerchiamo ora di capire come opera l'algoritmo Backpropagation per calcolare le derivate, partendo da casi semplici.

Poiché una rete neurale è equivalente ad una complessa concatenazione di funzioni, ci aspettiamo che la regola della catena per il calcolo della derivata di funzioni composte giochi un ruolo importante nel calcolo del gradiente. Teniamo conto di questo fatto dando ai nodi della rete una particolare struttura: ogni nodo si immagina suddiviso in due parti, una destra e una sinistra. A destra si trova la funzione associata ad ogni nodo e a sinistra la sua derivata. Questo tipo di rappresentazione viene detta *B-diagram* (per Backpropagation-diagram).

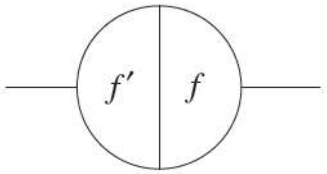


Figura4.2.1 Le due parti di un'unità computazionale in un B-diagram

Si osservi che la funzione discriminante può essere divisa dalla funzione di attivazione dividendo ogni nodo del B-diagram in due parti, come mostra la figura 4.2.2: il primo nodo fa la somma degli input e il secondo vi applica la funzione di attivazione s. Nella parte sinistra dei nodi sono presenti le derivate: la derivata di s è s' e la derivata parziale di una somma di n termini rispetto ad ognuno di essi è 1.

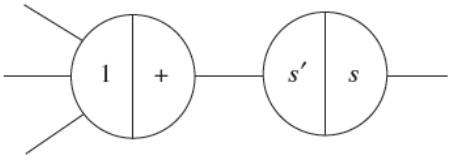


Figura4.2.2 Separazione della funzione discriminante (somma pesata degli input al primo nodo) e funzione di attivazione (s del secondo nodo)

La rete viene quindi valutata in due fasi:

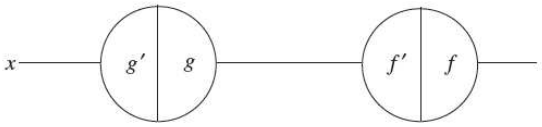
1. fase *feed-forward*: l'informazione viene da sinistra e ogni nodo calcola la funzione a destra e la derivata a sinistra rispetto ai dati in ingresso x. Entrambi i risultati vengono memorizzati nel nodo ma solo il risultato ottenuto della parte destra viene trasmesso in avanti lungo la rete;

2. fase *backpropagation*: l'intera rete viene fatta funzionare all'inverso, quindi da destra verso sinistra. Si pone 1 come valore di input e vengono utilizzati gli altri dati, moltiplicando i valori che erano stati memorizzati nella parte sinistra dei nodi. Il risultato è la derivata della funzione.

Vediamo come esempi le principali funzioni prese singolarmente (cioè fuori dal contesto specifico delle reti neurali).

4.2.1 Composizione di funzioni

Consideriamo un B-diagram di due nodi come illustrato in figura:



Dato un input scalare x la fase feed-forward calcola le derivate delle due funzioni nei punti corrispondenti a tale input (quindi in x la prima e in $g(x)$ la seconda). Questi dati vengono memorizzati nei nodi, ma solo l'informazione di destra viene propagata: l'output di questa fase sarà dunque la composizione delle due funzioni.

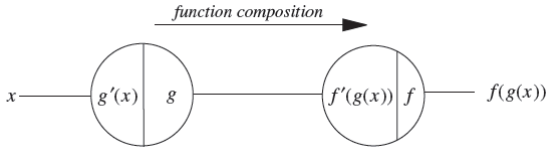


Figura4.2.1.1: Risultato della fase feed-forward: calcola la composizione

La fase backpropagation invece parte da destra prendendo come input 1 e implementa la regola della catena. Il risultato di questa fase, infatti, viene calcolato con le funzioni memorizzate nella parte sinistra dei nodi come mostrato in figura 4.2.1.2

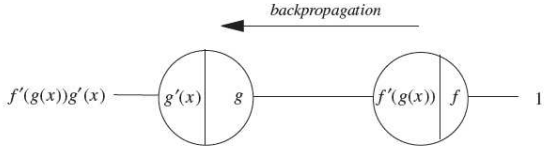


Figura4.2.1.2: Risultato della fase backpropagation: calcola la derivata secondo la regola della catena

4.2.2 Addizione

Dato un input x, la somma di due funzioni $f_1(x)$ e $f_2(x)$ viene calcolata nella fase di feed-forward come mostrato in figura:

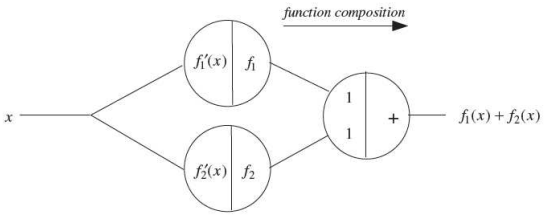


Figura4.2.2.1: Risultato della fase feed-forward: calcola la somma

La fase backpropagatin calcola la derivata della somma, che è la somma delle derivate, come illustrato di seguito

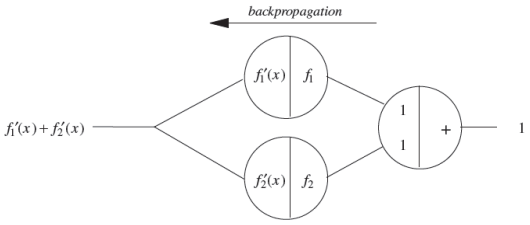
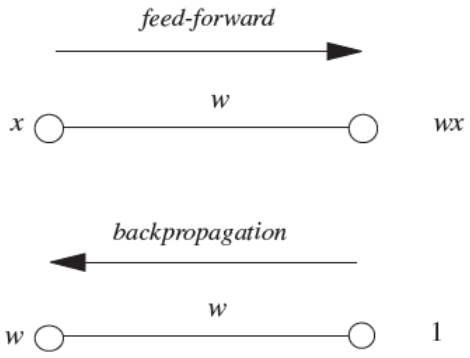


Figura4.2.2.2: Risultato della fase backpropagation: calcola la derivata della somma

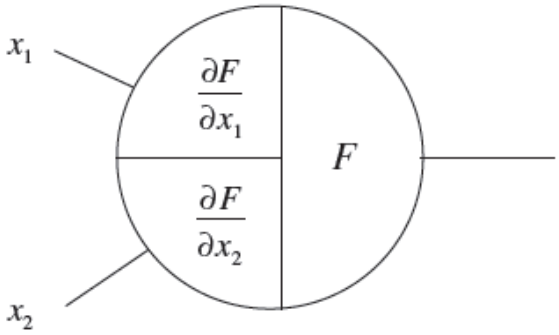
4.2.3 Lati pesati

I lati pesati possono essere visti come una composizione o, in maniera più semplice, come una moltiplicazione: nella fase feed-forward l'informazione x in input viene moltiplicata per w ottenendo wx, dove x ew sono valori scalari.

Nella fase backpropagation è il valore 1 proveniente da destra ad essere moltiplicato per il peso. Notiamo che il risultato della backpropagation è comunque la derivata poiché w è la derivata di wx rispetto ad x.



Finora abbiamo considerato funzioni di una variabile scalare x, ma i risultati trovati possono essere estesi a funzioni di più variabili: in tal caso la parte sinistra del nodo sarà divisa in più parti, ognuna contenente una derivata parziale della funzione.



L'algoritmo Backpropagation funziona correttamente anche in questo caso: la fase feed-forward rimane invariata eccetto per il fatto che vengono memorizzate più derivate; nella fase backpropagation, invece, si individuano più cammini, ognuno dei quali genera la derivata della F rispetto a tale componente.

4.3 L'algoritmo Backpropagation applicato a reti neurali multistrato

Come abbiamo detto, il nostro fine è utilizzare l'algoritmo Backpropagation per il calcolo del gradiente della funzione d'errore. Nel seguito considereremo il caso multi-output in cui $\mathbf{o^i}$ e $\mathbf{t^i}$ sono dei vettori. Tuttavia, per semplificare la

notazione, supporremo di avere un unico campione di esempio e quindi un'unica coppia $(\mathbf{o}, \mathbf{t})$. In Figura 4.3.1 rappresentiamo la rete multistrato estesa in questo caso: la parte destra elabora la deviazione quadratica $\frac{1}{2}(o_j^{(2)} - t_j)^2$ per l'j-esima componente del vettore di output e la parte sinistra ne memorizza la derivata $(o_j^{(2)} - t_j)$. L'output $o_j^{(2)}$ è ottenuto applicando la funzione sigmoide s (nel nodo precedente) all'output del network originale.

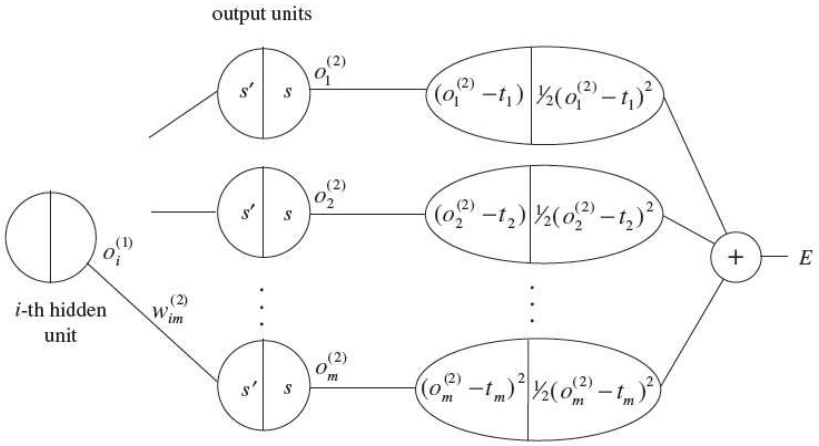


Figura4.3.1: Rete multistrato estesa per il calcolo dell'errore

Se vi fossero N campioni di esempio, la funzione d'errore totale si calcolerebbe creando N reti come quella appena illustrata e sommandone i risultati.

Dopo aver inizialmente impostato dei pesi casuali per la rete possiamo applicare l'algoritmo Backpropagation che in questo caso specifico consiste in 4 passi:

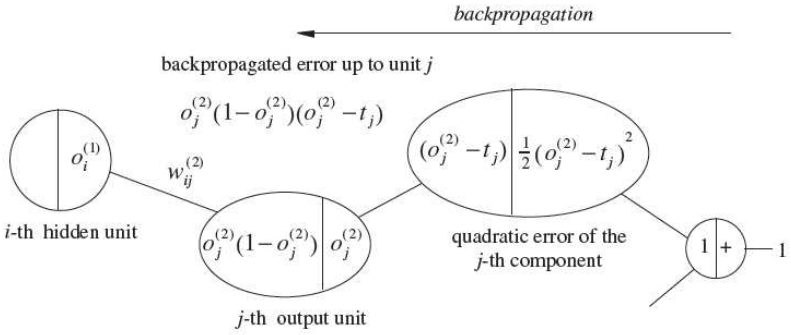
1. fase feed-forward
2. Backpropagation sul livello di uscita
3. Backpropagation sul livello nascosto
4. aggiornamento dei pesi

L'algoritmo termina quando il valore della funzione d'errore non migliora significativamente (minimo locale).

Vediamo ora le diverse fasi più nel dettaglio:

Fase feed-forward: La rete elabora i vettori di output $\mathbf{o}^{(1)}$ e $\mathbf{o}^{(2)}$ e ogni unità memorizza il valore della sua funzione e della sua derivata in tali punti. Solo i risultati della parte destra dei nodi vengono propagati e in uscita viene calcolato l'errore E.

Backpropagation sul livello di uscita: Si segue il cammino da destra verso sinistra per generare la derivata parziale $\frac{\partial E}{\partial w_{ij}^{(2)}}$, mostrato in figura:



Moltiplicando i valori memorizzati nella parte sinistra di ogni nodo troviamo il *backpropagated error* δ_j che è quindi

$$\delta_j^{(2)} = \mathrm{o_j^{(2)}}(1 - \mathrm{o_j^{(2)}})(\mathrm{o_j^{(2)}} - \mathrm{t_j})$$

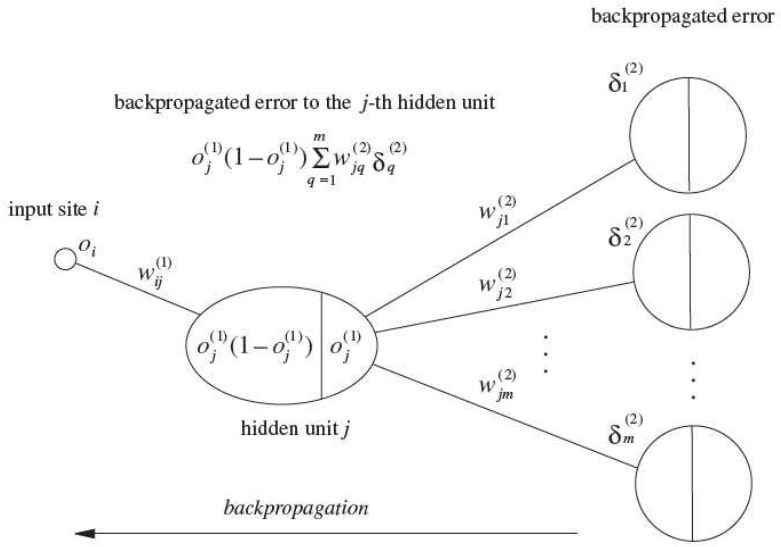
Ora osserviamo che nella fase di backpropagation w_{ij} è variabile mentre o_i è una costante quindi

$$\frac{\partial E}{\partial w_{ij}} = o_i \frac{\partial E}{\partial o_i w_{ij}} = o_i \delta_j$$

e dunque

$$\frac{\partial E}{\partial w_{ij}^{(2)}} = o_i^{(1)} \delta_j^{(2)} = o_i^{(1)} [\mathrm{o_j^{(2)}}(1 - \mathrm{o_j^{(2)}})(\mathrm{o_j^{(2)}} - \mathrm{t_j})]$$

Backpropagation sul livello nascosto: Questa fase calcola $\frac{\partial E}{\partial w_{ij}^{(1)}}$. Ogni unità j del livello nascosto è connessa a tutte le unità del livello di uscita tramite lati pesati, e quindi deve tener conto del *backpropagated error* di ognuna di queste come illustrato in figura:



Il *backpropagated error* $\delta_j^{(1)}$ si ottiene in maniera analoga a quanto visto precedentemente tenendo conto degli errori $\delta_q^{(2)}$ del livello di uscita pesati, cioè

$$\delta_j^{(1)} = \mathrm{o_j^{(1)}}(1 - \mathrm{o_j^{(1)}}) \sum_{\mathrm{q=1}}^{\mathrm{m}} \mathrm{w_{jq}^{(2)}} \delta_{\mathrm{q}}^{(2)}$$

e dunque la derivata che stavamo cercando sarà

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \delta_j^{(1)} o_i.$$

Il *backpropagated error* può essere calcolato analogamente per un numero arbitrario di livelli nascosti e l'espressione per le derivate parziali di E mantiene la stessa forma analitica.

Aggiornamento dei pesi: Questa è la fase di apprendimento vero e proprio. Dopo aver calcolato le derivate parziali i pesi della rete vengono aggiornati nella direzione dell'antigradiente. Una costante di apprendimento γ definisce di quanto ci si muove in tale direzione. Le correzioni da applicare ai pesi sono date da:

$$\Delta w_{ij}^{(2)} = -\gamma o_i^{(1)} \delta_j^{(2)} \text{ per } i = 0, 1...k; j = 1...m$$

e

$$\Delta w_{ij}^{(1)} = -\gamma o_i \delta_j^{(1)} \text{ per } i = 0, 1...n; j = 1...k$$

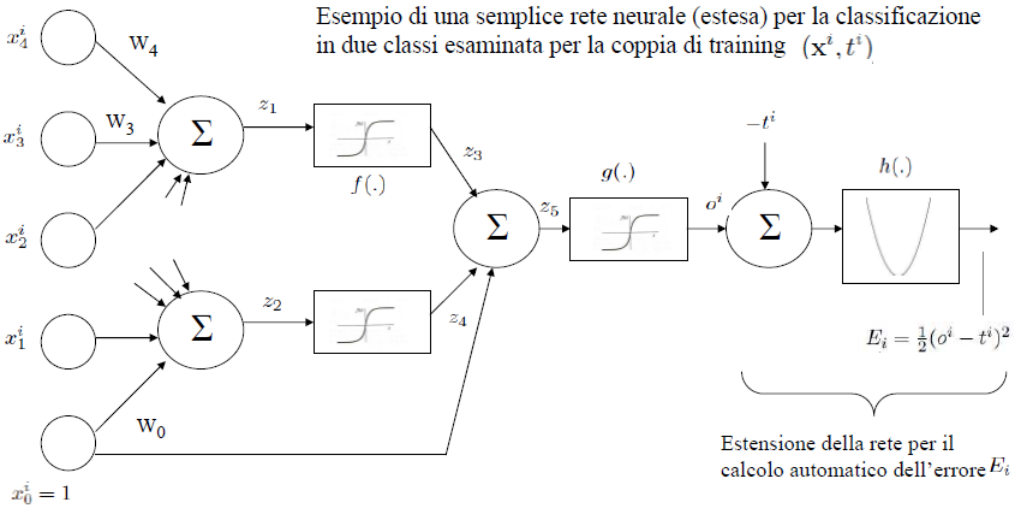
usando la convenzione che $o_0^{(1)} = o_0 = 1$.

E' molto importante che l'aggiornamento dei pesi avvenga solo quando il *backpropagated error* è stato calcolato per tutte le unità del network, altrimenti la direzione di spostamento non corrisponderebbe più a quella associata all'antigradiente.

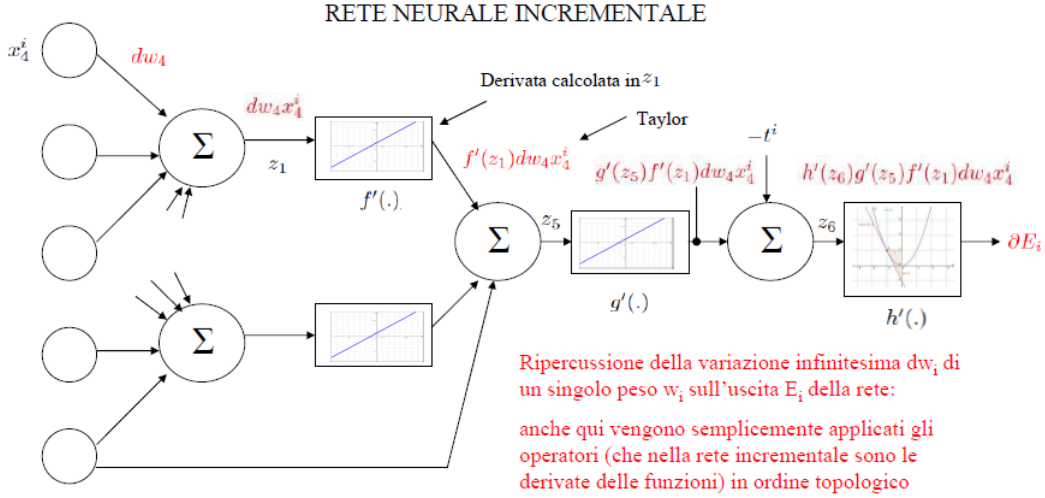
4.4 Esempio

Nel seguito viene illustrato un semplice esempio per mostrare perchè l'algoritmo Backpropagation è particolarmente efficiente nel calcolo del gradiente. Vengono esaminate soltanto le operazioni relative al calcolo dell'errore e delle sue derivate parziali (fasi feed-forward e backpropagation), con riferimento ad un singolo punto $\mathbf{x}^i$ del training set.

Per rendere più intuitiva la trattazione si è qui introdotto il concetto di "rete neurale incrementale" che rappresenta la linearizzazione della rete nell'intorno del punto di lavoro.



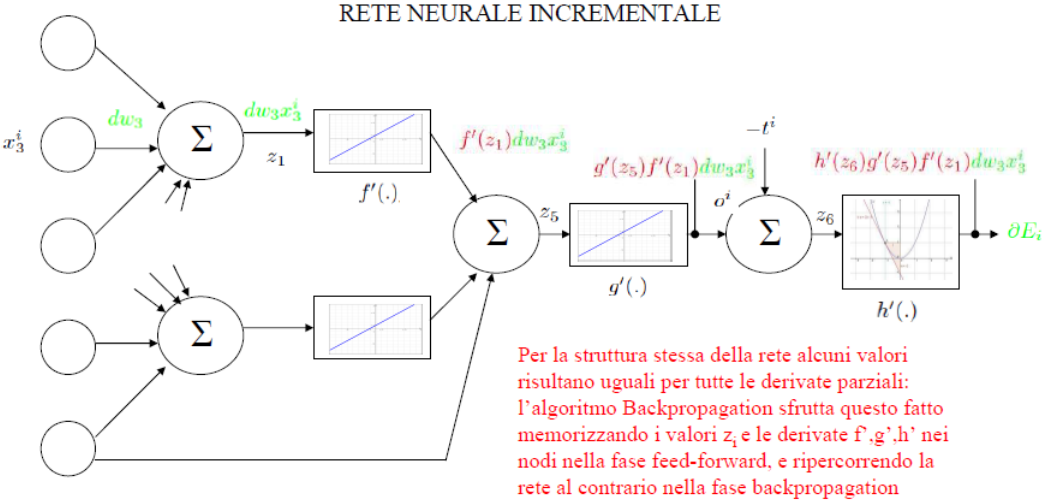
RETE NEURALE INCREMENTALE



Ripercussione della variazione infinitesima dw_i di un singolo peso w_i sull'uscita E_i della rete:

anche qui vengono semplicemente applicati gli operatori (che nella rete incrementale sono le derivate delle funzioni) in ordine topologico

RETE NEURALE INCREMENTALE



Per la struttura stessa della rete alcuni valori risultano uguali per tutte le derivate parziali: l'algoritmo Backpropagation sfrutta questo fatto memorizzando i valori z_i e le derivate f',g',h' nei nodi nella fase feed-forward, e ripercorrendo la rete al contrario nella fase backpropagation

5 Conclusioni: pregi, difetti e altre applicazioni delle reti neurali

5.1 Pregi

- Per come sono costruite, le reti neurali lavorano **in parallelo** e sono quindi in grado di trattare molti dati mentre nei calcolatori tradizionali ciascun dato viene elaborato individualmente e in successione. Si tratta in sostanza di un sofisticato sistema di tipo statistico dotato di una **buona immunità al rumore**: se alcune unità del sistema dovessero funzionare male, la rete nel suo complesso avrebbe delle riduzioni di prestazioni ma difficilmente andrebbe incontro ad un blocco del sistema. I software di ultima generazione dedicati alle reti neurali richiedono comunque buone conoscenze statistiche. Da un punto di vista industriale, risultano efficaci quando si dispone di dati storici che possono essere trattati con gli algoritmi neurali. Ciò è di interesse per la produzione perché permette di estrarre dati e modelli senza effettuare ulteriori prove e sperimentazioni.

- L'elaborazione delle reti neurali è **distribuita** su molti elementi, ovvero vi sono molti neuroni che si occupano della stessa operazione.

- Le reti neurali non devono essere programmate per svolgere un compito, bensì **imparano autonomamente** in base all'esperienza o con l'aiuto di un agente esterno.

5.2 Difetti

- I modelli prodotti dalle reti neurali, anche se molto efficienti, non sono spiegabili in linguaggio simbolico umano: i risultati vanno accettati "così come sono", da cui la definizione inglese delle reti neurali come sistemi **black box**.

- Come per qualsiasi algoritmo di modellazione, anche le reti neurali sono efficienti solo per **alcuni tipi di variabili** e queste vanno scelte con cura. Per esempio, non sono in grado di trattare in modo efficiente variabili di tipo categorico con molti valori diversi.

- Necessitano di una fase di addestramento del sistema che ne ottimizzi i pesi e questa fase può richiedere molto **tempo** se il numero dei campioni di training e delle variabili analizzate è molto grande.
- Non esistono teoremi o modelli che permettano di definire la rete ottima, quindi la riuscita di una rete **dipende molto dall'esperienza del creatore.**

5.3 Utilizzi

Le reti neurali vengono solitamente usate in contesti dove i dati possono essere parzialmente errati oppure in cui non esistano modelli analitici in grado di affrontare il problema. Vediamo brevemente qualche esempio.

5.3.1 Diagnostica in medicina

La medicina è stato uno dei primi settori ad utilizzare le reti neurali per migliorare la qualità delle diagnosi su malattie in genere e sui tumori in particolare. Questo perché, in medicina, l'attività diagnostica molte volte non può essere ricondotta ad un problema deterministico. Basti pensare che un clinico esperto è in grado di risolvere un quesito diagnostico ma talora non ha alcuna idea del come ci sia riuscito. Egli ha certamente applicato le regole apprese durante il corso di studi, ma verosimilmente un grosso contributo alla sua decisione diagnostica lo hanno dato le numerose esperienze simili che ha dovuto affrontare in anni ed anni di pratica clinica.

È quindi evidente che anche non avendo conoscenze "propedeutiche" è possibile, per una rete neurale particolarmente efficiente come quella umana, riuscire a categorizzare tramite l'esperienza (che altro non è che un tipo di statistica umana) e a dare diagnosi attendibili.

Applicazioni diagnostiche delle reti neurali sono, ad esempio:

- ricerca automatica delle lesioni cancerose a partire direttamente dalle lastre radiologiche
- scelta di trattamento in ortodonzia tramite l'analisi dei dati di una cartella clinica (dati cefalometrici e dentali del paziente facilmente ricavabili da una radiografia)
- ricerca di pattern funzionali e/o strutturali in proteine e acidi nucleici (bioinformatica).

5.3.2 Controllo di qualità su produzioni industriali

Le reti neurali possono costituire programmi in grado di fornire ad impianti industriali le soluzioni desiderate nell'ambito del controllo di qualità della produzione, sostituendo i metodi tradizionali di osservazione manuale di ogni singolo prodotto o a campione, per deciderne il livello qualitativo.

Di fronte a una grande massa di prodotti realizzati (e quindi di dati da elaborare), un metodo basato sulle reti neurali è in grado di classificarli: basterà far osservare al programma vari esempi di prodotti il cui standard qualitativo sia già stato riconosciuto in un senso o nell'altro, e la rete neurale stessa costruirà al suo "interno" un modello di tolleranze per poter giudicare i successivi prodotti, associando magari il tutto ad un sistema automatico in gradi di individuare i pezzi e a scartare i refusi.

5.3.3 Applicazione finanziaria

Lo scopo di gran parte di queste applicazioni e quello di realizzare progetti speculativi tramite trading, prevalentemente di breve o brevissimo periodo, di attività finanziarie quotate in borsa: principalmente azioni, tassi di cambio, futures, commodities.

Le reti neurali possono essere inoltre utilizzate per la gestione dei portafogli, della stima dei modelli di curva dei rendimenti, della valutazione dei titoli obbligazionari e azionari, delle strategie di trading, di copertura e di arbitraggio.

5.3.4 Previsioni meteorologiche

La potenza computazionale di una rete neurale in ambito meteorologico sta nel fatto che permette di ottenere soluzioni per analogia, risolvendo problemi la cui dinamica può non essere perfettamente conosciuta. In ambito meteorologico, infatti, latmosfera viene studiata secondo regole prestabilite, ma talvolta su scala locale intervengono altri fattori sconosciuti.

Le previsioni tramite reti neurali, a differenza delle tradizionali che si basano su modelli deterministici, possono adattarsi e suggerire previsioni del tempo molto attendibili nell'immediato.

Ad esempio un gruppo del Politecnico di Losanna assieme all'Università dell'Aquila, ha utilizzato le reti neurali per riconoscere l'insorgere del "blocco meteorologico", un fenomeno che si manifesta come una specie di quiete meteorologica riguardante una vasta regione sulla quale, ad esempio, staziona

per diverse settimane una zona di alta pressione. Il blocco meteorologico ha delle conseguenze importanti in quanto può causare notevoli danni economici (siccità, nebbia, inquinamento) e da un punto di vista scientifico è uno dei problemi teorici più affascinanti. La rete neurale è stata in questo caso addestrata a riconoscere quali sono i sintomi che caratterizzano l'insorgere del fenomeno utilizzando come prima base di conoscenza le mappe di pressione dell'Atlantico del Nord e dell'Europa dal 1949 al 1992. Le prestazioni della rete sono risultate equivalenti e in alcuni casi migliori del metodo tradizionale di riconoscimento del blocco. Attualmente si sta lavorando ad una vera e propria previsione su scala più ridotta (Italia centrale), per il momento utilizzando i dati di dieci anni ottenuti con modelli di previsione. Il Politecnico di Losanna ha inoltre realizzato, in collaborazione con l'Istituto svizzero di meteorologia, un primo tentativo di previsione della nuvolosità all'aeroporto di Zurigo. In tal caso la rete è stata addestrata a correlare delle osservazioni strumentali (temperatura, vento, ecc.) con la presenza delle nuvole quantificata in termini di copertura e di altezza. I dati di apprendimento si riferiscono ad un anno di osservazioni. In questa fase la rete ha riconosciuto correttamente circa l'88% di casi che non aveva mai visto. Prove in campo hanno mostrato delle eccellenti prestazioni con un punteggio di circa il 70%.

5.3.5 Applicazioni ai veicoli e per scopi militari

Una nuova area di applicazioni emergente è quella dei veicoli autonomi come sommergibili, veicoli terrestri su ruote o cingolati, aerei. Un sistema di visione artificiale può sia supportare un pilota di questi veicoli in varie situazioni, come può addirittura (nel caso di veicoli fully-autonomous) occuparsi dell'intera navigazione. In questo caso è importante saper riconoscere gli ostacoli e riuscire a produrre una mappa della zona circostante. Esempi in questa area sono i warning-system nelle automobili, i sistemi per l'atterraggio automatico degli aerei o i sistemi per la guida automatica di autovetture. Quest'ultima tecnologia anche se studiata e prodotta non ha ancora raggiunto tuttavia un costo accettabile per essere lanciata sul mercato.

Le applicazioni militari sono probabilmente una delle più grandi aree che sfrutta i benefici della visione artificiale, anche se solo una piccola parte del lavoro svolto in questo ambiente viene reso pubblico. Ovvi esempi possono essere il sistema di guida e puntamento dei missili.

Riferimenti bibliografici

[1] Bishop, C.M. (1995) *Neural Network for Pattern Recognition*; Oxford University Press

[2] Rojas R. (1996) *Neural Networks: a Systematic Introduction*; Springer

[3] Kulkarni A.D. (1994) *Artificial Neural Networks for Image Understanding*; VNR Computer Library, New York

[4] McCulloch, W. S. e Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, 5, 115137.

[5] Russell S. e Norvig P. (2010) *Artificial Intelligence: a Modern Approach*; Pearson

[6] Aguzzi A. (2001) Reti Neurali: le principali applicazioni pratiche; Punto informatico; Gruppo edizioni Master; *http://punto-informatico.it/95237_2/PI/News/reti-neurali-principali-applicazioni-pratiche.aspx*

[7] Visconti G. (1997) Corriere della scienza. Nuovo approccio alle previsioni metereologiche; Corriere della Sera del 13 aprile 1997, p.24; *http://archiviostorico.corriere.it/1997/aprile/13/Reti_neurali_studiano_cielo_poi_co_0_970413801.shtml*