Identificazione Di Modelli Attraverso Reti Neurali

Professore
A. Casavola

Allievo
Gullì Daniel
matr. 88531

INTRODUZIONE

Il concetto di intelligenza non è mai stato ben definito. La maggior parte dei tentativi per chiarirlo usano espressioni a loro volta imprecise ed ambigue. Molti non ritengono di poter attribuire il predicato "intelligente" ad alcuna macchina e limitano tale proprietà - per lo più per definizione - solo all'uomo. Tale argomentazione non è trascurabile, ma non getta alcuna luce sul problema. Più utile è la seguente definizione: l'intelligenza è la facoltà di adattarsi ad un mondo che cambia. Quanto più questa facoltà di adattamento è spiccata e molteplice, tanto più il sistema è intelligente.
Secondo questa definizione sono intelligenti non solo gli uomini, ma tutti gli esseri viventi ed anche alcune macchine.
I sistemi di intelligenza artificiale vengono spesso messi a confronto con le facoltà intellettuali dell'uomo. E così i critici giungono soddisfatti alla conclusione che il computer è una macchina stupida che sa calcolare in modo eccezionalmente veloce, incapace però di creare o realizzare alcunché.
Quando si verifica il contrario si pongono allora nuove mete e quelle raggiunte non vengono più considerate. Così si era affermato che il computer non avrebbe mai imparato a giocare decentemente a scacchi. Poiché invece l'ha fatto, si dice allora che non saprà mai comporre una sinfonia. Se poi ciò un giorno accadesse, si pretenderà che sappia ridere ad una battuta...
Effettivamente, le prime realizzazioni di intelligenza artificiale si erano poste degli obiettivi troppo elevati per le limitate capacità dei nostri computer: per esempio si credeva che per fare una traduzione bastasse fornire i vocabolari della lingua sorgente e di quella di destinazione e fissare quindi la corrispondenza tra i vocaboli equivalenti delle due lingue, oltre a impostare in un programma delle semplici regole di costruzione delle frasi.
I risultati furono piuttosto deludenti. E' famosa la frase "lo spirito è forte, ma la carne è debole" che, tradotta in russo e ritradotta nella lingua originale, suona all'incirca "la vodka è buona, ma la bistecca è immangiabile".
Queste considerazioni possono anche essere divertenti ma non vanno certo al cuore del problema. Non si vuole infatti creare una macchina "ad immagine e somiglianza" dell'uomo, ma più semplicemente aumentare le capacità dello strumento computer.
Di conseguenza è del tutto irrilevante che la macchina non sia capace di imitare l'uomo in una sua determinata attività.
Considerato che il computer può essere usato non solo per l'elaborazione numerica ma anche per quella simbolica, si sono poste infatti sin dagli anni '50 le due seguenti domande fondamentali:

- può uscire da un computer di più di quanto ci si è messo in termini di programma?
- può una macchina in qualche modo "pensare"?

Alla prima domanda si rispose affermativamente già nel 1960. Si era scritto un programma per computer che dopo un certo periodo di addestramento giocava a dama meglio dei suoi creatori e che ancora oggi riesce a non farsi battere dai migliori giocatori.
La seconda domanda portò ad una discussione senza fine sul concetto del pensare. Per evitare l'infruttuosa disputa il matematico inglese Turing propose nel 1950 il suo famoso test: un osservatore è collegato tramite un terminale ad altri due terminali, dei quali si sa soltanto che ad uno è addetto un uomo, all'altro una macchina. Se una persona, che intrattiene il dialogo sia con l'uomo che con la macchina attraverso il terminale non riesce a distinguerli, allora bisognerebbe concedere anche alla macchina la facoltà di "pensare".
La realizzazione di una macchina che superi il test di Turing è un traguardo che non può essere mai raggiunto, perché il pensiero umano è improntato anche di relazioni sensitive e sociali che restano proscritte ad un computer. E così siamo di nuovo al punto di partenza: vale veramente la pena di confrontarne il comportamento con quello dell'uomo?

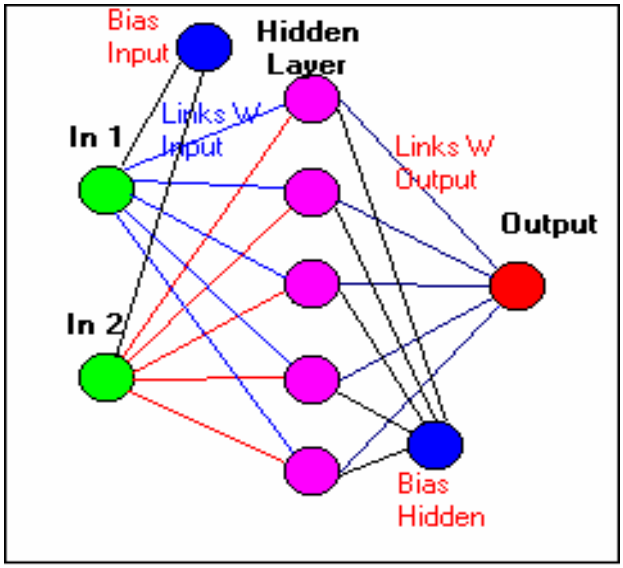
Mentre il gioco della dama o degli scacchi si svolge secondo regole fisse e perciò può essere ridotto, almeno in linea di principio, ad un programma, ciò non vale per la maggior parte dei settori della conoscenza di cui si occupa l'intelligenza artificiale. Uno dei suoi compiti principali è l'elaborazione della conoscenza, e cioè il ricavare delle conclusioni da determinati fatti. Ciò detto, si possono seguire due vie:

- l'approccio psicologico e cioè realizzando sul computer modelli del comportamento con cui l'uomo risolve i problemi, sperando così di capire il funzionamento del cervello (modello funzionale);
- l'approccio ingegneristico e cioè utilizzando qualsiasi metodo che produca risultati positivi (modello realizzativo).

Si consideri l'esempio del volo: l'imitazione del volo degli uccelli è rimasto infruttuoso, mentre le ricerche sistematiche su modelli in gallerie del vento hanno portato alla teoria della aerodinamica e quindi sia alla spiegazione del meccanismo del volo degli uccelli che alla realizzazione delle macchine per volare.
La velocità dei moderni computer é misurabile in nanosecondi mentre quella dei nostri neuroni lo é in millisecondi. Dal nostro tempo di reazione, valutabile in circa mezzo secondo, per riconoscere un'immagine o un testo, si può dedurre un numero di circa 100 passi di elaborazione che coinvolgono però un numero enorme di cellule neurali. Perciò, l'efficacia del sistema biologico è chiaramente attribuibile all'elaborazione parallela. Non sappiamo però nulla sulle leggi di concatenamento, di elaborazione e di riduzione dell'informazione.
Se si esclude quindi l'imitazione del comportamento umano e si limita il campo applicativo a particolari settori specialistici già oggi si può attribuire al computer un comportamento intelligente.
Alcuni esempi storici mostrano le possibilità dell'intelligenza artificiale.
Una pietra miliare fu posta dal programma SHRDLU. Questo simula un robot ad un braccio che, in un mondo costituito dai pezzi di una scatola di costruzioni, può muovere cubetti, piramidi, scatole ed altri pezzi disposti su un piano.
Può eseguire comandi in lingua naturale, effettuando una serie complessa di manipolazioni, può rispondere alle domande "come, dove e perché", può immagazzinare informazioni, discutere i suoi piani ed eseguirli.
In questo caso si può parlare di comprensione perché la conoscenza del piccolo mondo in cui deve agire il robot è rappresentata internamente.
IL MACSYMA, un esperto di matematica, in continuo sviluppo ed impiego dal 1968, realizza differenziazioni ed integrazioni simboliche meglio di specialisti umani.
IL DENDRAL, progettato da Feigenbáum nel 1969, determina la struttura molecolare a partire dai dati forniti da uno spettrografo di massa e raggiunge e supera in alcuni casi le capacità di grandi esperti di chimica.
MYCIN, sviluppato sin dall'inizio degli anni '70, produce diagnosi mediche per determinate infezioni batteriche e presenta proposte di terapie con relativo dosaggio di antibiotici.
Le sue capacità sono ben al di sopra di quelle di un comune medico e possono essere paragonate solo a quelle di alcuni specialisti universitari.
I sistemi esperti si sono cimentati anche con le realtà industriali. IL PROSPECTOR, impiegato per la ricerca di minerali strategici, ha scoperto un giacimento di molibdeno nello stato di Washington valutato oltre 100 milioni di dollari.
Ai nostri giorni, è possibile costruire calcolatori ed altre macchine che sono in grado di effettuare una varietà di ben definiti compiti con una rapidità e precisione che non sono alla portata degli esseri umani. Nessun uomo può invertire una matrice 100 per 100 o risolvere una equazione differenziale alle derivate parziali alla velocità di una workstation. Tuttavia, vi sono molti problemi che le macchine non possono risolvere in modo soddisfacente, mentre sono alla portata degli animali superiori o persino di pesci o insetti. Si pensi ad esempio alle capacità di memorizzazione e riconoscimento di un dato percorso, alle possibilità di controllo degli arti per la locomozione, e così via. Ancor oggi non è facile dire quale sia il segreto del cervello animale. Si può però ossservare

che i singoli costituenti del cervello, i neuroni biologici, sono meno veloci dei componenti elementari delle macchine. Non si può quindi spiegare la capacità di elaborazione degli animali in termini di velocità e di elaborazione dei componenti elementari. L'opinione corrente è che tali rapidità sia in qualche modo determinate dall'elevato livello di interconnessione tra neuroni elementari; il cervello è infatti un vero e proprio coacervo di neuroni variamente intercorressi. Si stima che vi siano 10^{12} neuroni nel cervello di un essere umano adulto, ciascuno dei quali può essere collegamento fino a 10^5 neuroni circostanti. Nella corteccia cerebrale, la connettività media (cioè il numero medio di neuroni a cui è collegato un singolo neurone) è 10^3.
Stimolati da queste osservazioni, è nato il tentativo di mimare il funzionamento del cervello mediante le cosiddette reti neurali artificiali, o semplicemente reti neurali (NN = Neural Network). Vedremo ora la costituzione di alcune semplici reti neurali artificiali ed il loro impiego come «approssimanti universali» di funzioni non lineari. Successivamente, discuteremo il loro uso per scopi di identificazione e controllo di sistemi dinamici.

Reti Neurali



Quando si crea un programma convenzionale per svolgere un dato compito (applicazione), è necessario comprendere a fondo il problema e la sua soluzione. Bisogna quindi implementare, per ogni possibile applicazione, le strutture dati che la rappresentano, all'interno del computer, con una sintassi legata ad un linguaggio per computer. E' necessario poi scrivere un apposito programma, più o meno complesso, per gestire o analizzare queste strutture dati. Vi sono però casi in cui il problema che il programma deve risolvere non è formalizzabile in un algoritmo matematico di risoluzione, o perchè ci sono troppe variabili, o perchè semplicemente il problema sfugge alla normale comprensione umana. I programmi applicativi spaziano quindi tra due estremi: da una parte i problemi strutturati, che sono completamente definiti, e dall'altra i problemi casuali, che sono del tutto indefiniti e la cui soluzione dipende interamente dall'addestramento a base di esempi di una "Rete neurale". Le reti neurali basate su esempi rappresentano quindi un metodo per destreggiarsi nell'ampio territorio dei problemi casuali non strutturati.

Il neurone standard

Il neurone artificiale elementare, che in seguito chiameremo semplicemente neurone, è un sistema con un certo numero, diciamo q, di ingressi, u1, , u_2 .. , u_q, ed una uscita, y. Tutti gli ingressi e

l'uscita sono variabili reali. Gli ingressi vengono combinati linearmente e formano il cosiddetto segnale di attivazione

$$s = \theta_0 + \theta_1 u_1 + \theta_2 u_2 + ... + \theta_q u_q$$

dove $\theta_0, \theta_1, \theta_2, ... \theta_q$ sono dei parametri reali che pesano diversamente i vari ingressi (e per questo vengono detti pesi); il parametro θ_0 prende il nome di soglia o parametro di polarizzazione (bias). Va da sé che è possibile rendere questo parametro formalmente indistinguibile dagli altri introducendo un ulteriore ingresso u_0 :

$$s = \theta_0 u_0 + \theta_1 u_1 + \theta_2 u_2 + ... + \theta_q u_q$$

pur di porre $u_0 = 1$.

A valle del nodo sommatore che fornisce s, si effettua un'elaborazione non lineare, ottenendo il segnale di uscita:

$y = \sigma(s)$

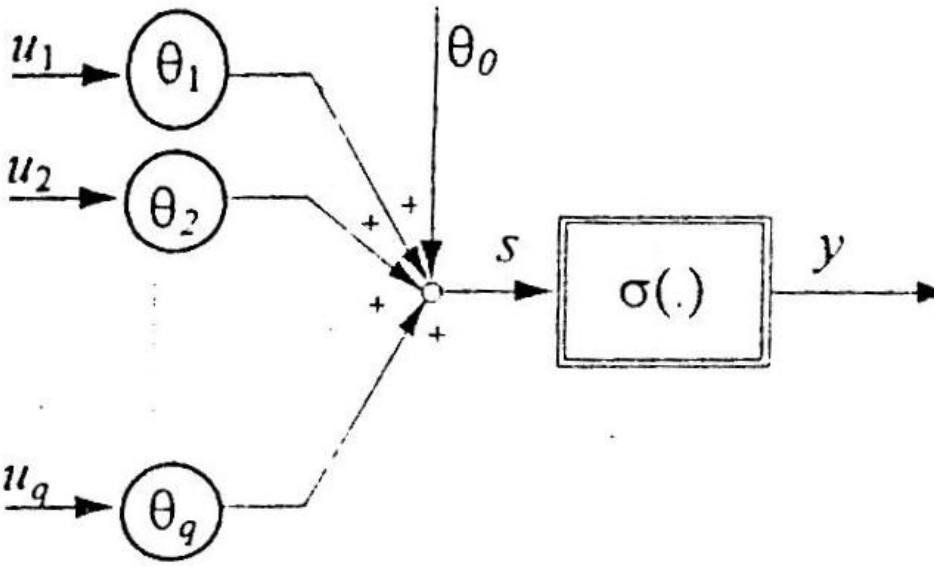
dove $\sigma(\cdot)$ è una funzione non lineare denominata funzione di attivazione. Di norma $\sigma(\cdot)$ è una funzione reale di variabile reale, monotona crescente e dotata di asintoto inferiore

$$\lim_{s \to -\infty} \sigma(s) = \sigma_{\inf}$$

e asintoto superiore

$$\lim_{s \to +\infty} \sigma(s) = \sigma_{\sup}$$

Per queste ragioni, $\sigma(\cdot)$ prende il nome di funzione sigmoidale o di sigmoide. Il neurone puo quindi essere rappresentato come in figura.



Nel suo complesso, il neurone è dunque una trasformazione da $\Re^q$ in $\Re$, elaborando gli ingressi $u_1, u_2, ..., u_q$ a formare l'uscita y. L'espressione analitica corrispondente è:

$$y = \sigma\left(\theta_0 + \theta_1 u_1 + \theta_2 u_2 + ... + \theta_q u_q\right)$$

che si può sinteticamente scrivere

$$y = \sigma\left(\theta_0 + \theta^T \varphi\right) \quad (1)$$

pur di definire il vettore dei pesi

$$\theta = \left[\theta_0, \theta_1, \theta_2, ... \theta_q\right]^T$$

e quello delle osservazioni di ingresso

$$\varphi = \left[u_1, u_2, ..., u_q\right]^T$$

Naturalmente, invece della (1) si può anche scrivere:

$$y = \sigma\left(\theta_E{}^T \varphi_E\right)$$

dove

$$\theta_E = \left[\theta_0, \theta_1, \theta_2, ... \theta_q\right]^T$$
$$\varphi_E = \left[1, u_1, u_2, ..., u_q\right]^T$$

sono denominati vettori estesi dei pesi e degli ingressi.

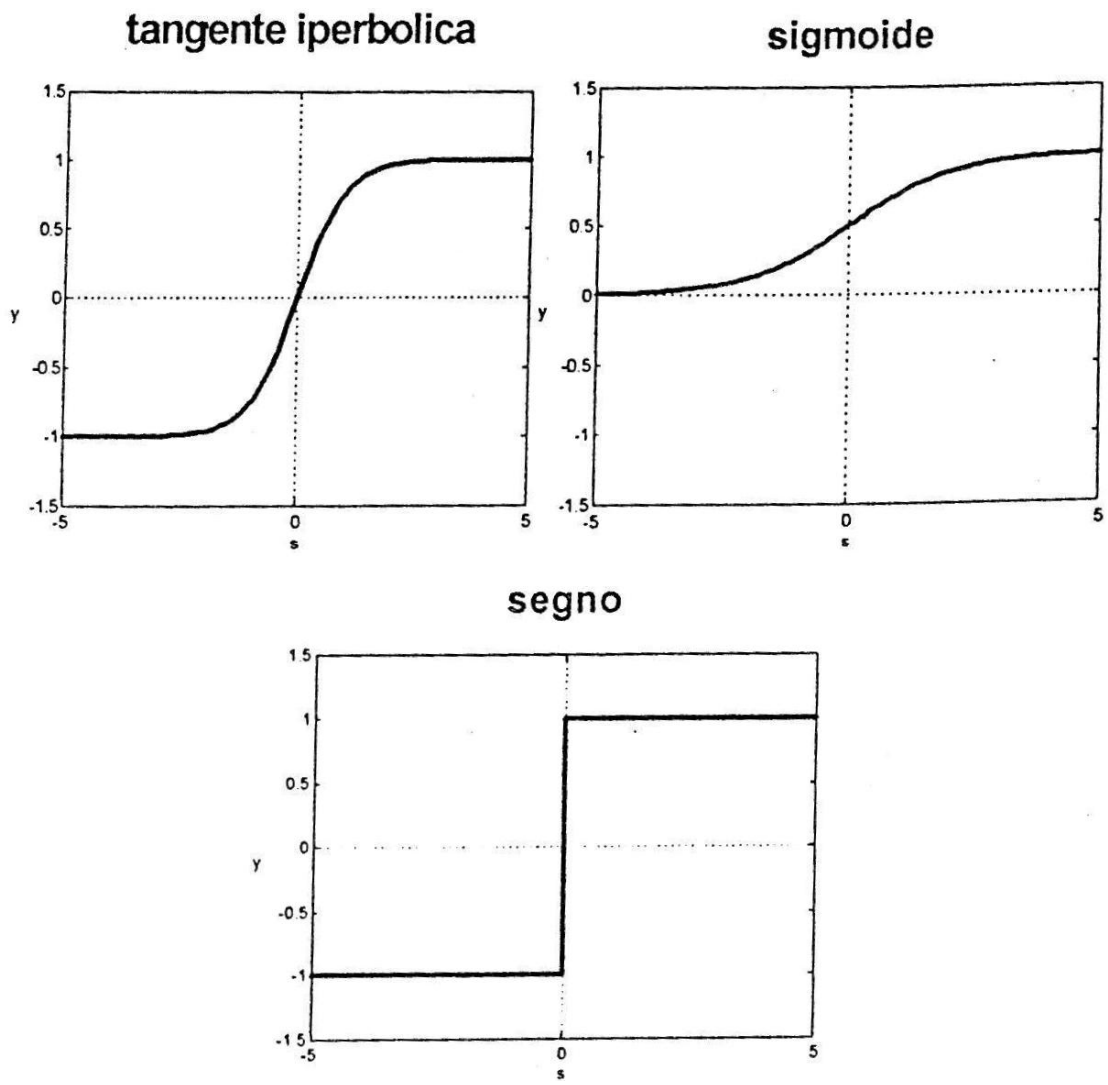
Quanto al «sigmoide» $\sigma(\cdot)$, diverse sono le scelte possibili. Tra queste le più frequenti sono:

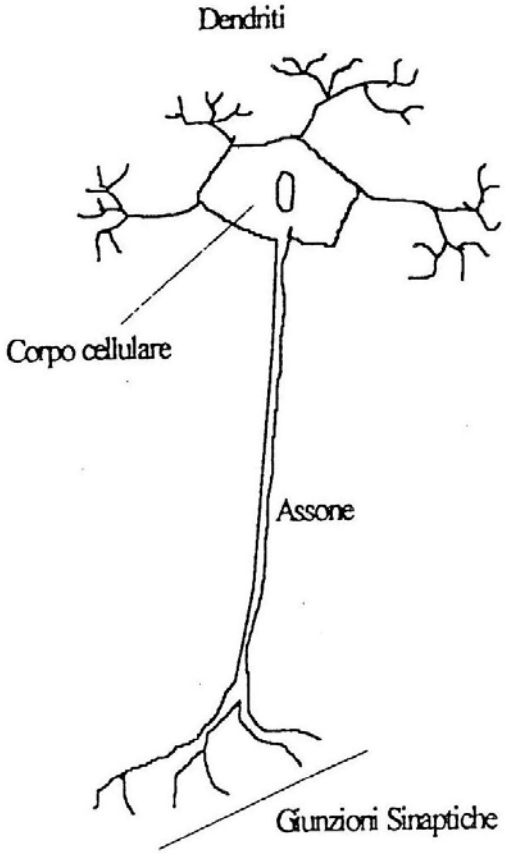
a) $\sigma(s) = \dfrac{1}{1+e^{-s}}$

b) $\sigma(s) = \tanh(s)$

c) $\sigma(s) = \begin{cases} 1 & s \geq 0 \\ -1 & s < 0 \end{cases}$

i cui andamenti sono rappresentati nella figura seguente.





Nota

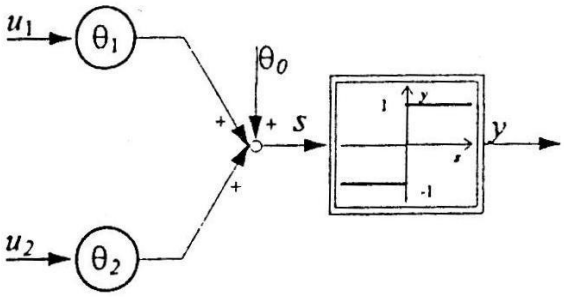
Il neurone biologico (rappresentato nella figura preceddente) è costituito da un corpo cellulare (o soma), che riceve segnali elettrici di ingresso dai suoi dentriti per mezzo di ioni. L'interno del corpo cellulare, infatti, è carico negativamente rispetto al fluido extracellulare; i .segnali che arrivano al neurone attraverso ioni sodio (Na+) diminuiscono la differenza di potenziale tra l'interno e l'esterno della cellula fino ad arrivare ad un punto in cui il neurone emette un impulso («scarica»). Il fenomeno di scarica si propaga dai dendriti al corpo cellulare attraverso gli assoni ed attraverso questi ultimi perviene alla giunzione sinaptica, ché costituisce il legame tra un neurone e l'altro. Il segnale prodotto dal neurone è il risultato di una quantità di stimoli che pervengono da altri neuroni attraverso gli assoni. A loro volta gli stimoli derivano da opportune azioni di vaglio e selezione delle informazioni in arrivo attraverso gli organi di senso. L'informazione globale così mediata produce eventualmente la «scarica» del neurone. Ciò corrisponde ad un fenomeno non lineare (la «scarica») attivato dall'azione integrata di diversi stimoli. Questo modo di operare è, per così dire, stilizzato dal neurone artificiale appena introdotto. Le variabili di ingresso u; corrispondono agli stimoli esterni, i parametri θ, sono i pesi attribuiti agli stimoli, pesi che potranno essere positivi o negativi, di valore assoluto più o meno elevato a seconda della significatività dell'informazione apportata da uno stimolo; la globalità degli stimoli pesati viene composta a dare un segnale globale sintetico, detto segnale di attivazione. Nella rappresentazione artificiale del neurone, tale segnale è s(t). Questa variabile che, per così dire, sintetizza tutti gli stimoli da cui è sollecitato il neurone, viene poi elaborata alfine di produrre delle decisioni, oppure alfine di produrre altri segnali che serviranno a loro volta da stimolo per altri neuroni. Per esempio; si potrebbe prendere la decisione seguente: se il segnale di attivazione è troppo basso,allora il neurone inibisce ogni stimolo in uscita. Se invece è sufficientemente elevato, il neurone invia una scarica. Nella schematizzazione artificiale, questo corrisponderebbe a prendere come $\sigma(s)$ la funzione segno:

$$\sigma(s) = \begin{cases} 1 & s \geq \varepsilon \\ -1 & s < \varepsilon \end{cases}$$

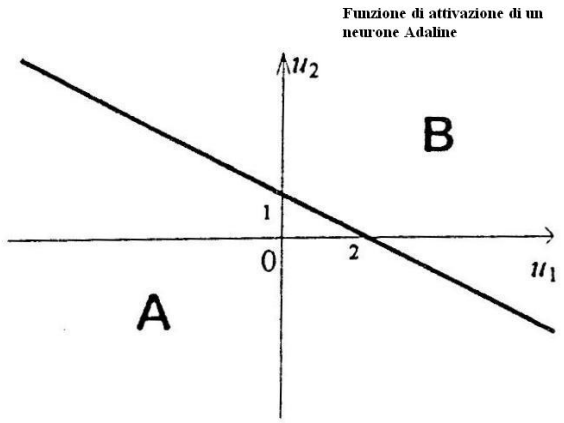
Si spiega così la consolidata tradizione di chiamare la $\sigma(s)$ funzione di attivazione.

Esempio 4.2.1 (Adaline)

Si consideri il neurone rappresentato nella seguente figura, caratterizzato da due ingressi e da una non linearità «segno»:



Il neurone Adaline



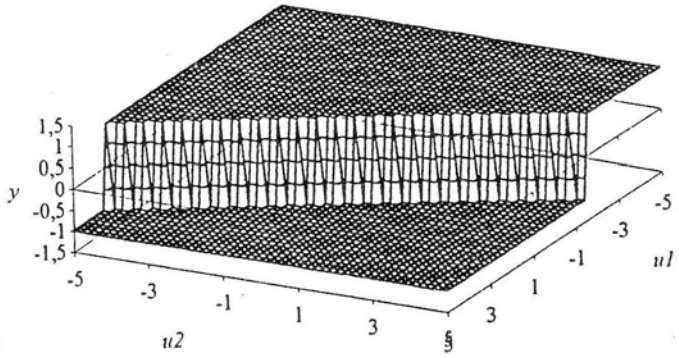
Questo neurone è noto come Adaline (da Adaptive Linear Element). L'uscita può assumere solo il valore +l oppure -l; precisamente

$$y = \begin{cases} 1 & se \quad s \geq 0 \\ -1 & se \quad s < 0 \end{cases}$$

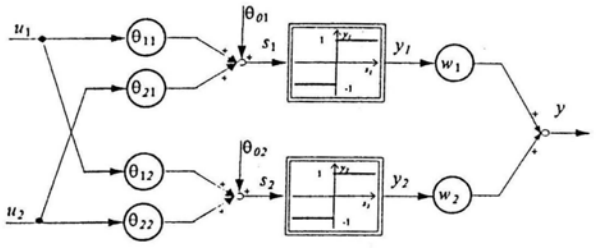
Dato che $s = \theta_0 + \theta_1 u_1 + \theta_2 u_2 + ... + \theta_q u_q$, la condizione che determina la separazione tra i due modi di uscita:

$$\theta_0 + \theta_1 u_1 + \theta_2 u_2 + ... + \theta_q u_q = 0$$

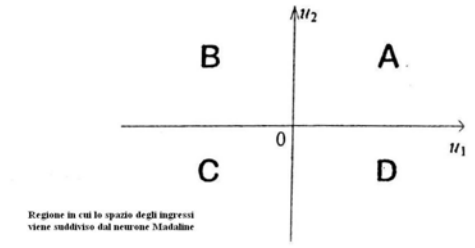
Nel piano (u_l, u_2), questa equazione definisce una retta, la cui posizione dipende dai valori attribuiti ai parametri θ_0, θ_1 *e* θ_2. Ad ogni semipiano definito dalla retta corrisponde un valore della variabile di uscita y come indicato in figura. Se il particolare valore assunto dalla coppia di ingresso (u_l, u_2) appartiene alla regione A, allora y=+1 , altrimenti y=-1.



Si noti il diverso ruolo giocato dai parametri $\theta_0, \theta_1 \; e \; \theta_2$, variando $\theta_1 \; e \; \theta_2$ si altera la pendenza della separatrice; mediante θ_0 si può invece ottenere un effetto di traslazione. Da quanto detto,è evidente che Adaline può essere utile come riconoscitore. L'uscita binaria consente infatti di distinguere immediatamente il semipiano in cui è situata la coppia dei valori d'ingresso. Il riconoscitore Adaline si limita alla possibilità di separare $\Re^2$ soltanto in due semipiani; per ottenere separatrici più complicate si ricorre a reti Adaline ,chiamate Madaline.



Il neurone Madaline



Regione in cui lo spazio degli ingressi viene suddiviso dal neurone Madaline

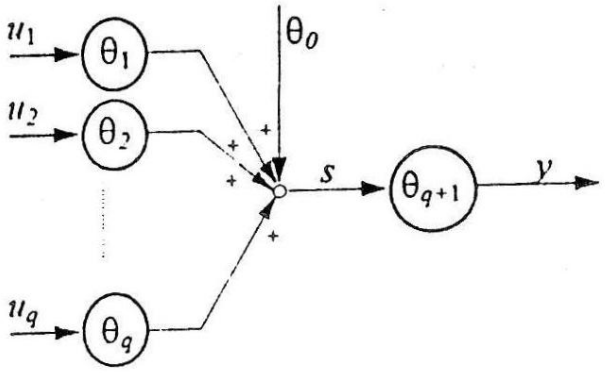
Esempio 4.2.2 (Madaline)
Si consideri l'insieme di due neuroni Adaline interconnessi come indicato in figura. Questa rete è nota come Madaline, così come, più in generale tutte le reti ottenute componendo delle Adaline prendono tale nome. Le due rette generate dai due neuroni Adaline dividono il piano (u_1, u_2) in quattro parti che in questo caso coincidono con i quattro quadranti. La funzione $y=f(u_1, u_2)$ determinata dalla rete è rappresentata nella figura precedente. La funzione complessiva del neurone Madaline è dunque quella di riconoscere a quale delle quattro regioni (in cui il piano di ingresso è diviso dalle due rette) il punto determinato dalla coppia delle variabili di ingresso appartenga.

Nota
Per completezza, conviene considerare anche i cosiddetti «neuroni lineari». Si tratta di neuroni in. cui la funzione di attivazione e in realtà lineare. In effetti il neurone lineare non è che un sistema che effettua una combinazione lineare degli ingressi:

$$y = \theta_{q+1}\theta_0 + \theta_{q+1}\theta_1 u_1 + \theta_{q+1}\theta_2 u_2 + ... + \theta_{q+1}\theta_q u_q$$

e perciò potrebbe essere chiamato semplicemente «combinatore lineare», con pesi della combinazione dati dal prodotto, $\theta_{q+1}\theta_i$.

Con questa definizione possiamo dire che la rete rappresentata nella seguente figura è quindi costituita da due strati, il primo composto da due neuroni non lineari, e il secondo da un unico neurone lineare



Il neurone lineare

Apprendimento di una rete

La capacità di «approssimazione universale» delle reti neurali rende questi sistemi ideali per l'identificazione non lineare a scatola nera. Il problema della messa a punto di una rete che realizzi compiutamente l'approssimazione desiderata di una data funzione è però critico, tanto più quanto maggiore è la complessità della rete. Di norma, si opera ad architettura fissata; si stabilisce cioè a priori il numero dei neuroni e la distribuzione delle relative interconnessioni. Anche la funzione sigmoidale viene di solito prescelta. La messa a punto della rete si riduce allora all'identificazione dei parametri che la caratterizzano. Nel gergo neurale, il procedimento d'identificazione, prende il nome di addestramento o di apprendimento. Si parla di *apprendimento* quando l'identificazione è effettuata al fine di imporre un determinato e prestabilito comportamento alla rete; quando invece il comportamento desiderato viene replicato esponendo la rete a dati ingresso-uscita sperimentalmente rilevati, si parla di *addestramento*. Nel seguito, ci riferiremo a quest'ultimo contesto, che è il classico ambito di lavoro dei metodi di identificazione. La differenza sostanziale tra l'addestramento di una rete neurale e il problema dell'identificazione di un modello lineare (per esempio ARX o ARMAX) risiede nell'elevato numero di parametri delle reti neurali di uso corrente. Mentre un modello ARMAX con 30 parametri è considerato un modello assai complesso, una rete con 100 pesi è una rete di bassa complessità. Il fatto che una rete neurale sia caratterizzata da un così elevato numero di parametri ha almeno due effetti importanti, che contraddistinguono il problema di neuroidentificazione dal problema dell'identificazione finora visto. Innanzitutto, la messa a punto dei parametri avviene di norma utilizzando più volte gli stessi dati di ingresso-uscita. L'addestramento viene cioè effettuato per iterazioni successive, ciascuna delle quali viene effettuata elaborando sempre i medesimi dati disponibili, per esempio N coppie di dati d'ingresso e uscita:

$$\left\{\left(u_1(t), u_2(t), ..., u_q(t)\right), y(t) \middle| t = 1,2,..., N\right\}$$

al fine di aggiornare i parametri. Ogni iterazione prende il nome di epoca. Non è infrequente che l'addestramento completo di una rete di un centinaio di parametri (esempio rete standard con 2 ingressi e un'uscita e 25 neuroni), tarata su qualche centinaio di dati (esempio N=300), richieda

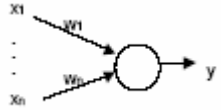
qualche migliaio di epoche (esempio 10.000 epoche). Il secondo importante effetto è che, per l'identificazione, è necessario basarsi su algoritmi semplici. Altrimenti, tenuto conto dell'elevato numero di parametri e di epoche, il tempo di calcolo potrebbe diventare insostenibile. L'algoritmo più usato verrà esposto nel prossimo paragrafo.

Apprendimento Supervisionato

Possiamo osservare che il corretto funzionamento della rete neurale dipende dall'architettura della rete (cioè dal numero di strati e dal numero di neuroni per strato), dalla funzione di attivazione dei neuroni e dai pesi. I primi due parametri sono fissati prima della fase di addestramento. Il compito dell'addestramento è quindi quello di aggiustare i pesi in modo che la rete produca le risposte desiderate. Uno dei modi più usati per permettere ad una rete di imparare è l'*apprendimento supervisionato*, che prevede di presentare alla rete per ogni esempio di addestramento la corrispondente uscita desiderata. Di solito i pesi vengono inizializzati con valori casuali all'inizio dell'addestramento. Poi si cominciano a presentare, uno alla volta, gli esempi costituenti l'insieme di addestramento (*training set*). Per ogni esempio presentato si calcola l'errore commesso dalla rete, cioè la differenza tra l'uscita desiderata e l'uscita effettiva della rete. L'errore è usato per aggiustare i pesi. Il processo viene di solito ripetuto ripresentando alla rete, in ordine casuale, tutti gli esempi del training set finché l'errore commesso su tutto il training set (oppure l'errore medio sul training set) risulta inferiore ad una soglia prestabilita. Dopo l'addestramento la rete viene testata controllandone il comportamento su un insieme di dati, detto *test set*, costituito da esempi non utilizzati durante la fase di training. La fase di test ha quindi lo scopo di valutare la capacità di generalizzazione della rete neurale. Diremo che la rete ha imparato, cioè è in grado di fornire risposte anche per ingressi che non le sono mai stati presentati durante la fase di addestramento. Ovviamente le prestazioni di una rete neurale dipendono fortemente dall'insieme di esempi scelti per l'addestramento. Tali esempi devono quindi essere rappresentativi della realtà che la rete deve apprendere e in cui verrà utilizzata. L'addestramento è in effetti un processo ad hoc dipendente dallo specifico problema trattato.

Delta rule

Riferiamoci al neurone rappresentato di seguito:



La regola più usata per aggiustare i pesi di un neurone è la *delta rule* o *regola di Widrow-Hoff*. Sia $x = (x_1, \ldots, x_n)$ l'ingresso fornito al neurone. Se *t* ed *y* sono, rispettivamente, l'uscita desiderata e l'uscita neurale, l'errore δ è dato da

$$\delta = t - y.$$

La delta rule stabilisce che la variazione del generico peso Δw_i è:

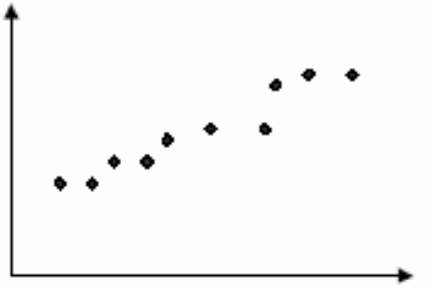
$$\Delta w_i = \eta \delta x_i$$

dove η è un numero reale compreso tra 0 e 1 detto *learning rate*. Il learning rate determina la velocità di apprendimento del neurone. La delta rule modifica in maniera proporzionale all'errore solo i pesi delle connessioni che hanno contribuito all'errore (cioè che hanno $x_i \neq 0$). Al contrario, se xi=0, wi non viene modificato poiché non si sa se ha contribuito all'errore. Il nuovo valore dei pesi è quindi:

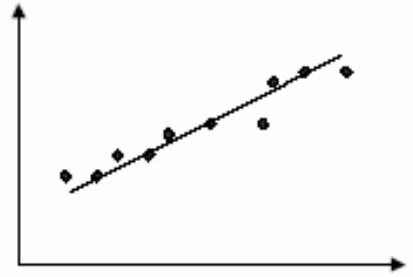
$$w_i = w_i + \Delta w_i$$

Esempio di addestramento

Molti problemi, per poter essere risolti, richiedono di adattare una retta o una curva ai dati che si hanno a disposizione. Consideriamo, ad esempio, un insieme di punti nel piano che seguono l'andamento di una linea retta ma non appartengono esattamente ad alcuna linea retta.



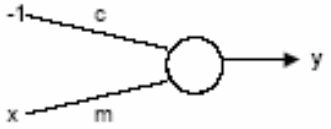
Possiamo interpolare i punti mediante una retta, calcolata, ad esempio usando il *metodo dei minimi quadrati*. Tale metodo ci permette di calcolare la retta che minimizza la somma degli errori quadratici per tutti i punti. Per ogni punto, l'errore è la distanza del punto dalla retta.



La retta disegnata può essere utile per ricavare (o prevedere) valori della variabile dipendente (rappresentata sull'asse delle ordinate) in corrispondenza di valori della variabile indipendente per cui non sono state fatte misurazioni. Considerando che l'equazione di una retta in forma esplicita è:

$$y = mx + c$$

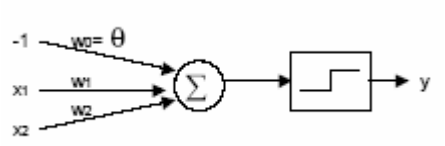
dove *y* ed *x* sono variabili, *m* la pendenza e *c* il punto in cui la retta incontra l'asse *y*, possiamo calcolare *m* e *c* con il metodo dei minimi quadrati risolvendo le equazioni ottenute uguagliando a 0 le derivate parziali (rispetto ad *m* e *c*, rispettivamente) della somma degli errori quadratici.
In alternativa al metodo dei minimi quadrati si può usare una rete neurale che approssima una retta. In ingresso alla rete vengono presentati i punti da approssimare con la linea retta e si lascia alla rete il compito di apprendere. Possiamo usare una rete (rappresentata nella figura seguente) costituita da un neurone di ingresso ed un neurone di uscita con funzione di attivazione lineare. Dato che la rete deve stimare *m* e *c*, *m* e *c* costituiscono i pesi. Tali pesi saranno inizializzati in modo casuale. Gli esempi che costituiscono il training set sono le coppie (*x*, *y*) delle coordinate dei punti da approssimare con la linea retta; ovvero l'ascissa rappresenta l'ingresso e l'ordinata l'uscita desiderata. Il peso *c* è la soglia, quindi è associato ad un ingresso costantemente uguale a -1.



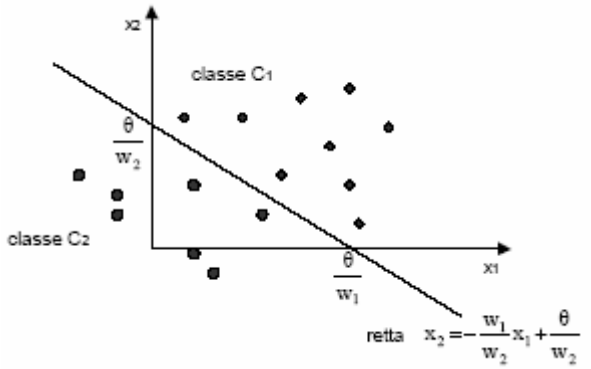
La rete viene addestrata usando la delta rule con un certo learning rate (ad esempio, $\eta = 0.1$). I punti del training set saranno presentati un certo numero di volte, finché l'errore commesso dalla rete non scende al di sotto di una soglia prestabilita. L'uscita della rete produrrà una retta del tutto simile a quella generata col metodo dei minimi quadrati. Diremo che la rete ha imparato perché dandole in ingresso l'ascissa di un punto non usato durante l'addestramento la rete produrrà in uscita l'ordinata corrispondente.

Classificazione

In molte applicazioni si incontrano problemi di classificazione di un insieme di oggetti, cioè occorre associare ogni oggetto alla classe corretta. Supponiamo di voler classificare in due classi distinte oggetti rappresentati mediante punti nel piano. Se le due classi sono linearmente separabili, possiamo usare una rete neurale che approssima una retta di separazione tra le due classi. Un oggetto sarà quindi classificato rappresentandolo come punto nel piano ed assegnandolo a quella delle due classi individuata dal semipiano in cui cade il punto. Tale classificazione può essere facilmente ottenuta addestrando una rete neurale con due ingressi ed un solo neurone di uscita con funzione di attivazione a soglia:



Tale rete è un esempio semplice di **perceptron**, costituito da più ingressi confluenti in un neurone di uscita con funzione di attivazione a soglia. Consideriamo, ad esempio, la figura seguente:



Possiamo associare la classe C1 all'insieme di stimoli per cui la rete risponde con y=1 e C2 all'insieme di stimoli per cui la rete risponde con y=0, cioè:

$$x \in C_1 \quad \text{se} \quad y=1$$
$$x \in C_2 \quad \text{se} \quad y=0$$

Nel piano (x_1, x_2) degli ingressi della rete le classi C1 e C2 sono rappresentate da due semipiani separati dalla retta

$$x_2 = -\frac{w_1}{w_2}x_1 + \frac{\theta}{w_2}.$$

Vediamo come è possibile addestrare il perceptron in modo che sia in grado di classificare correttamente i punti del piano. Dopo aver predisposto un opportuno training set, si fa uso della delta rule eseguendo i passi seguenti:

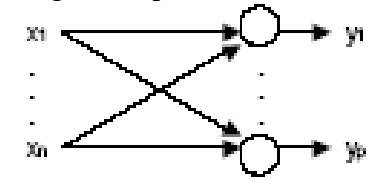
1. si inizializzano i pesi w_i con valori casuali;
2. si presenta alla rete un ingresso x_k insieme al valore t_k desiderato in uscita;
3. si calcola la risposta y_k della rete e si aggiornano i pesi mediante la delta rule;
4. si ripete il ciclo dal passo 2, finchè la risposta della rete non risulti soddisfacente.

Con riferimento all'ultima figura, osserviamo che il processo di apprendimento, modificando i pesi w_1, w_2 e θ, non fa altro che modificare la posizione e la pendenza della retta di separazione tra le due classi. Il processo termina quando la retta separa correttamente le due classi. Infine, osserviamo che la rete considerata nell'esempio è un perceptron con due ingressi perché gli oggetti da classificare sono rappresentati come punti in $\Re^2$. Tali punti sono separati da una retta. In generale,

se gli oggetti da classificare sono vettori *n*-dimensionali, gli ingressi del perceptron sono *n* e le due classi sono separate da un iperpiano in $\Re^n$.

Aggiornamento dei pesi e convergenza della delta rule

Consideriamo la seguente rete con *n* ingressi e *p* uscite:



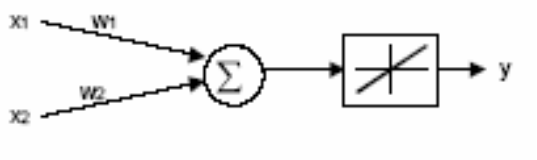
Siano t_j e o_j, rispettivamente, l'uscita desiderata e l'uscita effettiva del neurone *j*. L'errore E_k commesso dalla rete sull'esempio *k* può essere definito come:

$$E_k = \frac{1}{2}\sum_{j=1}^{P}\left(t_j - o_j\right)^2 \quad \textbf{(2)}$$

(il fattore ½ è stato introdotto per semplificare le notazioni) per cui l'errore globale commesso dalla rete su tutto il training set costituito da *m* esempi è:

$$E=\sum_{k=1}^{m} E_k$$

Per illustrare il razionale della delta rule, verrà preso in esame un caso semplice di comportamento lineare; anche se analiticamente tale situazione può essere affrontata con metodi diretti, faremo riferimento alla procedura iterativa della delta rule. Consideriamo allora un singolo neurone, rappresentato nella figura seguente, con due ingressi, soglia = 0 e funzione di attivazione lineare (cioè l'uscita coincide con l'ingresso: f(a)=a).



Tale neurone può modellare una qualunque linea retta passante per l'origine. Per un neurone lineare ed un singolo esempio, l'errore diventa:

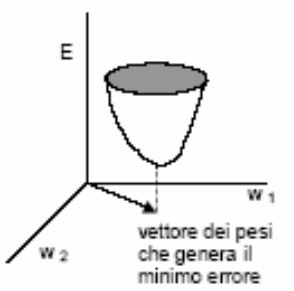
$$E=\frac{1}{2}(t-a)^2$$

Considerando che $a=X_1W_1+X_2W_2$e svolgendo i calcoli otteniamo:

$$E=\frac{1}{2}[t^2-2ta+a^2]=$$

$$=\frac{1}{2}[t^2-2t(x_1w_1+x_2w_2)+x_1^2w_1^2+2x_1w_1x_2w_2+x_2^2w_2^2]$$

Abbiamo ricavato che l'errore E, nel caso di neuroni lineari, è un paraboloide nello spazio dei pesi. In generale, per altri tipi di neuroni, sarà un'ipersuperficie nello spazio dei pesi. Prima di iniziare l'addestramento, i pesi sono inizializzati a valori casuali quindi il punto che rappresenta lo stato iniziale della rete può trovarsi ovunque sulla superficie dell'errore (in generale non coinciderà con il punto di minimo di tale superficie). Durante l'addestramento i pesi dovranno essere modificati in modo da far muovere lo stato della rete lungo una direzione, che la delta rule individuerà essere quella di massima pendenza, della superficie dell'errore in modo da minimizzare l'errore globale.



Ricordiamo la definizione della delta rule con cui aggiustare i pesi:

$$\Delta w_{ij} = \eta \delta_j x_i \qquad \delta_j = (t_j - o_j)$$

dove t_j è l'uscita desiderata dal neurone j, o_j l'uscita effettiva, x_i il segnale proveniente dal neurone i, η il learning rate e w_{ij} la variazione del peso sulla connessione da i a j. Vogliamo dimostrare che, aggiornando i pesi mediante la delta rule, l'apprendimento converge verso una configurazione dei pesi che minimizza l'errore quadratico globale. Osserviamo che per dimostrare la convergenza della delta rule basterebbe dimostrare che tale regola è riconducibile alla forma

$$\Delta w_{ij} = -\frac{\partial E}{\partial w_{ij}}$$

la quale varierebbe i pesi in modo da favorire la diminuzione dell'errore.
Infatti:
-se E cresce all'aumentare di w_{ij} cioè:

$$\frac{\partial E}{\partial W_{ij}} > 0$$

allora w_{ij} viene diminuito per contrastare la crescita di

$$E \ (\Delta w_{ij} < 0)$$

-se E diminuisce all'aumentare di w_{ij}

$$(\text{cioè } \frac{\partial E}{\partial w_{ij}} < 0)$$

allora w_{ij} viene aumentato per favorire la diminuzione di

$$E \ (\Delta w_{ij} > 0).$$

Per semplicità, consideriamo un neurone lineare con uscita definita da

$$o_j = \sum_i x_i w_{ij}$$

Esprimiamo la derivata dell'errore rispetto ad un peso come prodotto di due quantità, la prima delle quali esprime il cambiamento dell'errore in funzione dell'uscita di un neurone, la seconda riguarda il cambiamento dell'uscita rispetto ad un peso:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial w_{ij}}$$

dalla (2) e dalla definizione di δ_j otteniamo

$$\frac{\partial E}{\partial o_j} = -\delta_j$$

Inoltre

$$\frac{\partial o_j}{\partial w_{ij}} = x_i.$$

Di conseguenza

$$-\frac{\partial E}{\partial w_{ij}} = \delta_j x_i$$

A questo punto, inserendo il learning rate, otteniamo proprio la delta rule. Con riferimento alla figura che mostra la superficie dell'errore, osserviamo che il processo di apprendimento può essere interpretato come una discesa su tale superficie lungo la linea di massima pendenza, individuata appunto dal gradiente:

$$-\nabla E=\left(-\frac{\partial E}{\partial w_{ij}}\right).$$

Il learning rate η rappresenta quindi la rapidità di discesa di E sulla superficie. È importante scegliere il valore giusto per η: un valore troppo piccolo può comportare un apprendimento troppo lento, mentre un valore troppo elevato può provocare oscillazioni dell'errore intorno al minimo. La soluzione tipicamente adottata è quella di stabilire un valore alto di η (prossimo a 1) all'inizio dell'addestramento e diminuire tale valore mano a mano che si procede con l'apprendimento

$$\frac{\partial E}{\partial o_j}=-\delta_j$$

Problemi lineari e non lineari

Un problema di classificazione in cui si devono separare in due classi i punti appartenenti ad un certo insieme si dice *lineare* se una linea (in due dimensioni) o un iperpiano (in *n* dimensioni) possono separare correttamente tutti i punti. In caso contrario, il problema si dice *non lineare*. Abbiamo visto che un problema lineare può essere risolto usando un perceptron. Infatti, un perceptron con *n* ingressi è in grado di rappresentare un iperpiano *n*-dimensionale. Quindi, un perceptron è in grado di risolvere problemi linearmente separabili in cui gli ingressi devono essere catalogati in due differenti classi separabili tramite una retta (perceptron a due ingressi), un piano (perceptron a tre ingressi), o un iperpiano (perceptron a *n* ingressi). Un tipico problema non lineare è l'or esclusivo (XOR). Tale operatore, infatti, produce un'uscita solo quando uno solo degli ingressi vale 1, altrimenti dà 0. Non esiste alcuna retta che separi i punti (0,1) e (1,0) dai punti (0,0) e (1,1). Per risolvere questo problema si hanno due possibilità:

1) si ricorre a particolari funzioni di uscita non lineari
2) si usano reti con più strati

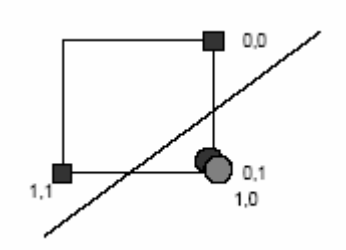
Nel primo approccio si utilizza una rete con *n* ingressi ed un neurone di uscita la cui funzione di uscita è scelta in modo appropriato. Un esempio di funzione di uscita non lineare adatta per i nostri scopi è la seguente:

$$y=(x_1-x_2)^2=\begin{cases}1 \text{ se } x_1\neq x_2\\ 0 \text{ se } x_1=x_2\end{cases}$$

Ovviamente, questo approccio può essere difficile da perseguire qualora la funzione non lineare richiesta sia difficile da individuare. Nel secondo approccio, si usa una rete con uno o più strati nascosti in modo da modellare due o più rette per separare i dati. Tale rete è detta *multilayer perceptron*. Nel caso dello XOR la rete può essere quella già vista precedentemente e contenente due neuroni nascosti che rappresentano due rette e un neurone di uscita che combina le informazioni prodotte dalle due rette. Con riferimento alla rete già vista per risolvere lo XOR, consideriamo la seguente tabella che riporta gli ingressi ai neuroni nascosti e le relative uscite.

		ingresso strato nascosto		uscita strato nascosto	
x_1	x_1	neurone 1	neurone 2	neurone 1	neurone 2
1	1	-0.5	-1.5	0	0
1	0	0.5	-0.5	1	0
0	1	0.5	-0.5	1	0
0	0	1.5	0.5	1	1

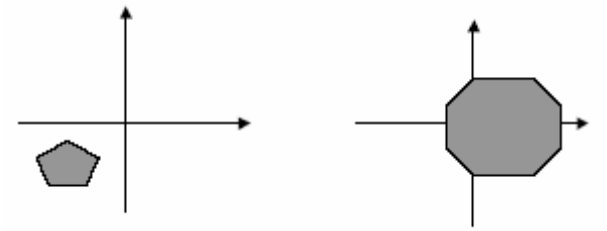
Possiamo osservare che gli ingressi alla rete risultano trasformati in uscita dallo strato nascosto: il primo livello di pesi (tra lo strato di ingresso e lo strato nascosto) ha spostato il punto originale (0,1) nel punto (1,0). Notiamo anche che (0,0) e (1,1) sono stati scambiati tra loro. La figura seguente rappresenta l'uscita dallo strato nascosto.



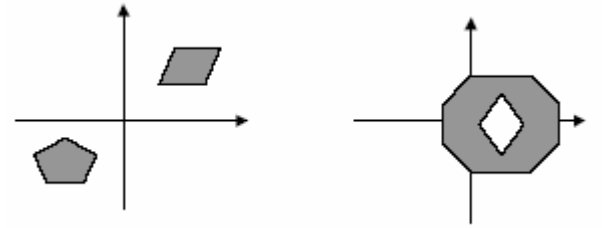
Anche il secondo livello di pesi (che connettono lo strato nascosto con lo strato di uscita) modella una retta. Affinché la rete produca gli output desiderati, occorre quindi che le configurazioni degli input al secondo livello di pesi siano separabili da una retta, come confermato dalla figura precedente. I pesi che definiscono la retta di separazione disegnata nella figura precedente sono quelli già visti quando abbiamo parlato per la prima volta del problema dello XOR.

Osservazioni

Usando reti con uno strato nascosto è possibile formare regioni decisionali convesse nello spazio degli ingressi. Dato che ogni neurone di uno strato separa due regioni, il numero di lati della regione è minore o uguale al numero di neuroni dello strato nascosto. Ad esempio, possiamo ottenere le seguenti regioni in grigio:



Con due strati nascosti possiamo realizzare regioni decisionali complesse,ad esempio:



La maggior parte dei problemi è risolvibile adottando reti con al massimo due strati nascosti. Osserviamo che la formazione di regioni decisionali complesse è possibile solo se si adottano funzioni di uscita non lineari. Infatti una rete multistrato con neuroni lineari è equivalente ad una rete con uno strato di ingresso ed uno strato di uscita. Ad esempio, con riferimento ad una rete con uno strato nascosto e neuroni lineari, siano n, p e q il numero di neuroni dei tre strati di ingresso, nascosto e di uscita, rispettivamente. Ricordando che, di solito, la matrice di connessione tra lo strato i e lo strato j si indica con W_{ji}, indichiamo con W_{21} e W_{32} le matrici dei pesi relative alla coppia di strati (ingresso-nascosto) e (nascosto-uscita), rispettivamente. W_{21} e W_{32} hanno dimensioni $\tilde{p}n$ e $\tilde{q}p$, rispettivamente. L'uscita dallo strato nascosto è data da $Y=W_{21}X$, essendo X il vettore di ingresso. L'uscita dallo strato di uscita è $Z=W_{32}Y=W_{32}W_{21}X$. Definendo $W = W_{32}W_{21}$ otteniamo $Z=WX$, cioè la rete considerata è equivalente ad una rete senza strati nascosti con connessioni espresse da una matrice W, di dimensione qn, che è il prodotto delle due matrici date.

BACKPROPAGATION

La rete multistrato che abbiamo adottato precedentemente per risolvere il problema dello XOR è stata costruita definendo i pesi appropriati. Ovviamente, ciò che ci interessa trovare è un algoritmo di addestramento supervisionato che permetta alla rete multistrato di trovare da sola i pesi. Il problema incontrato nell'addestramento delle reti multistrato è il seguente: volendo adottare un meccanismo di aggiornamento dei pesi simile alla delta rule (in cui l'errore è calcolato come differenza tra l'uscita desiderata e l'uscita effettiva di ciascun neurone) si riesce ad aggiornare solo i pesi relativi ai neuroni di uscita, ma non quelli relativi ai neuroni degli strati nascosti. Infatti, mentre per lo strato di uscita si conosce l'uscita desiderata (tale uscita viene data come secondo elemento delle coppie che costituiscono gli esempi del training set), niente si sa dell'uscita desiderata dai neuroni nascosti. Questo problema è stato risolto dopo molti anni di disinteresse per le reti neurali (in quanto non si riusciva ad addestrarle) solo nel 1986, quando fu introdotto l'algoritmo di *backpropagation*. Tale algoritmo prevede di calcolare l'errore commesso da un neurone dell'ultimo strato nascosto propagando all'indietro l'errore calcolato sui neuroni di uscita collegati a tale neurone. Lo stesso procedimento è poi ripetuto per tutti i neuroni del penultimo strato nascosto, e così via. L'algoritmo di backpropagation prevede che, per ogni esempio del training set, i segnali viaggino dall'ingresso verso l'uscita al fine di calcolare la risposta della rete. Dopo di che c'è una seconda fase durante la quale i segnali di errore vengono propagati all'indietro, sulle stesse connessioni su cui nella prima fase hanno viaggiato gli ingressi, ma in senso contrario, dall'uscita verso l'ingresso. Durante questa seconda fase vengono modificati i pesi. I pesi sono inizializzati con valori casuali. Come funzione di uscita non lineare dei neuroni della rete si adotta in genere la funzione sigmoide (l'algoritmo richiede che la funzione sia derivabile). Tale funzione produce valori tra 0 e 1. L'algoritmo di backpropagation usa una generalizzazione della delta rule. Nel seguito useremo *net* per indicare la somma pesata degli ingressi di un neurone. Possiamo esprimere la derivata dell'errore come segue:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

Poniamo:

$$\delta_j = -\frac{\partial E}{\partial net_j}$$

(osserviamo che questa definizione coincide con la delta rule. Lì, infatti, avevamo

$$\delta_j = -\frac{\partial E}{\partial o_j}$$

perché consideravamo neuroni lineari, cioè $o_i = net_i$). Riscriviamo l'equazione precedente:

$$\delta_j = -\frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}$$

Poiché l'errore commesso sul *k*-esimo esempio del training set è:

$$E_k = \frac{1}{2}\sum_j (t_j - o_j)^2$$

abbiamo

$$\frac{\partial E}{\partial o_j} = -(t_j - o_j)$$

Per la funzione di attivazione *f* (di solito, la funzione logistica) l'uscita è:

$$o_j = f(net_j)$$

e quindi:

$$\frac{\partial o_j}{\partial net_j} = f'(net_j)$$

da cui

$$\delta_j = (t_j - o_j) f'(net_j)$$

Essendo

$$net_j = \sum_i x_i w_{ij}$$

si ha:

$$\frac{\partial net_j}{\partial w_{ij}} = x_i$$

per cui, tornando alla formula di partenza risulta

$$\frac{\partial E}{\partial w_{ij}} = -(t_j - o_j) f'(net_j) x_i = -\delta_j x_i$$

Quindi, applicando il learning rate, abbiamo la seguente formula per la modifica dei pesi, sulla base della delta rule generalizzata:

$$\Delta w_{ij} = \eta \delta_j x_i$$

L'errore δ_J è calcolabile per un neurone di uscita, ma non per un neurone nascosto perché, come già detto, non conosciamo la sua uscita desiderata. Comunque, un neurone nascosto può essere adattato in modo proporzionale al suo contributo all'errore sullo strato successivo (verso l'uscita della rete). Fissando l'attenzione su un neurone dell'ultimo strato nascosto, possiamo dire che l'errore commesso da tale neurone può essere calcolato come somma degli errori commessi da tutti i neuroni di uscita collegati a tale neurone nascosto. Il contributo di ciascuno di tali errori dipende, ovviamente, sia dalla dimensione dell'errore commesso dal relativo neurone di uscita sia dal peso sulla connessione tra il neurone nascosto e il neurone di uscita. In altri termini, un neurone di uscita con un grosso errore contribuisce in maniera notevole all'errore di ogni neurone nascosto a cui è connesso con un peso elevato. Per un neurone nascosto l'errore è dato da:

$$\delta_j = f'(net_j) \sum_s \delta_s w_{js}$$

dove *s* è l'indice dei neuroni dello strato che trasmette all'indietro l'errore.
Come detto, una funzione spesso usata è la funzione logistica

$$f(net_j) = \frac{1}{1 + \exp(-net_j)}$$

La derivata di tale funzione è:

$$f'(net_j) = \frac{\exp(-net_j)}{(1 + \exp(-net_j))^2}$$

$$= \frac{1}{1 + \exp(-net_j)} \left(1 - \frac{1}{1 + \exp(-net_j)} \right)$$

$$= f(net_j)[1 - f(net_j)]$$

Osservazione
Abbiamo già osservato, parlando del perceptron, che il learning rate η non deve avere valori troppo bassi né troppo alti per evitare, rispettivamente, tempi di addestramento troppo lunghi o oscillazioni dell'errore. Esistono due tecniche per risolvere questo problema. La prima è la stessa vista per il perceptron e prevede di variare η nel tempo. La seconda prevede di ridurre la probabilità di oscillazione dei pesi usando un termine α, detto *momentum*, che è una costante di proporzionalità (compresa tra 0 e 1) alla precedente variazione dei pesi. La legge di apprendimento diventa quindi:

$$\Delta w_{ij}(n+1) = \eta \delta_j o_i + \alpha \Delta w_{ij}(n).$$

In questo modo, il cambiamento dei pesi per l'esempio *n*+1 dipende dal cambiamento apportato ai pesi per l'esempio *n*.

Algoritmo di backpropagation

Dato un training set costituito da *m* esempi (Xk,Tk), l'addestramento di una rete multistrato, con funzioni di trasferimento sigmoidi, avviene tramite i seguenti passi:

1) si inizializzano i pesi con valori casuali (in genere, con valori non troppo elevati);
2) si presenta un ingresso X_K e si calcolano le uscite

$$o_j = \frac{1}{1+\exp(-net_j)}$$

di tutti i neuroni della rete;
3) dato Tk, si calcolano l'errore δ_j e la variazione dei pesi w ij per ogni neurone dello strato di uscita:

$$\delta_j = (t_j - o_j) f'(net_j) = (t_j - o_j) o_j (1 - o_j)$$

4) partendo dall'ultimo strato nascosto e procedendo all'indietro, calcolare

$$\delta_j = f'(net_j) \sum_s \delta_s w_{js} = o_j (1 - o_j) \sum_s \delta_s w_{js}$$

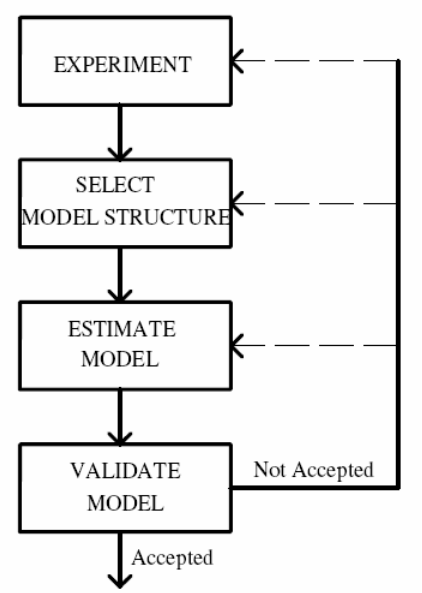
5) per tutti gli strati aggiornare i pesi:

$$\Delta w_{ij}(n+1) = \eta \delta_j o_i + \alpha \Delta w_{ij}(n)$$

6) ripetere il processo dal punto 2 finchè non si siano presentati tutti gli *m* esempi del training set;
7) si calcola l'errore medio sul training set (oppure l'errore globale); se l'errore è al di sotto di una soglia prefissata, l'algoritmo termina (si è raggiunta la convergenza), altrimenti si ripete un intero ciclo di presentazione del training set.

Un ciclo di presentazione degli esempi del training set è detto *epoca*. Esistono due modalità di applicazione dell'algoritmo di backpropagation: nella modalità *batch* i pesi sono aggiornati dopo aver presentato alla rete tutti gli esempi del training set, nella modalità *on-line* (o *incrementale*) i pesi sono aggiornati dopo la presentazione di ogni esempio. Nell'algoritmo precedente si è fatto riferimento a quest'ultima modalità.

Identificazione del sistema



Ci sono 4 passi fondamentali per l'indentificazione del sistema dinamico:

1) Experiment
2) Seleziona la struttura del modello
3) Stima i parametri del modello
4) Validazione del modello

Una volta acquisito il data set si seleziona la struttura del modello, ciò è molto complicato per un sistema non lineare. Non basta scegliere soltanto l'insieme dei regressori ma anche l'architettura della NN (Neural Network), l'approccio usato è quello descritto da Norgaard. Quando una particolare struttura della NN è stata selezionata è necessario scegliere inoltre il numero dei segnali precedenti da usare nel repressore cioè l'ordine del modello, se non è presente un rumore che agisce sul sistema o il livello di questo rumore è molto bassa esiste un funzione fornita da Matlab che determina l'ordine del modello:

```
OrderIndices=lipschit(u1s,y1s,1:7,1:7);
```

Sono presenti alcune funzioni che generano una NN in base al modello specificato.

Nonlinear System Identification	
nnarmax1	Identify a neural network ARMAX (or ARMA) model (linear noise filter).
nnarmax2	Identify a neural network ARMAX (or ARMA) model.
nnarx	Identify a neural network ARX (or AR) model.
nnarxm	Identify a multi-output neural network ARX (or AR) model.
nnigls	Iterated generalized least squares training of multi-output NNARX models
nniol	Identify a neural network model suited for I-O linearization type control.
nnoe	Identify a neural network Output Error model.
nnssif	Identify a neural network state space innovations form model.
nnrarmx1	Recursive counterpart to NNARMAX1.
nnrarmx2	Recursive counterpart to NNARMAX2.
nnrarx	Recursive counterpart to NNARX.

Le funzioni usano Levenberg-Marquardt method o recursive prediction error method e quindi devono passare gli stessi parametri alle funzioni marq orpl quando noi invochiamo:

```
[W1,W2,NSSEvec]=nnoe(NetDef,NN,W1,W2,trparms,y,u);
```

NetDef: stringa che definisce l'architettura della NN

```
NetDef=['HHHHHHHHHH';'L---------'];
```

specifica un rete con 10 neuroni con attivazione a tangente iperbolica e un'unica uscita con attivazione lineare.

```
NN=[na nb nc];
```

NN contiene l'ordine del modello. W1 e W2 contengono i pesi iniziali della rete, mentre:

```
trparms=settrain;
trparms=settrain(trparms,'maxiter',300,'D',1e-4,'skip',10);
```

setta i parametri di default dell'algoritmo di stima della NN. Dopo aver tarato la NN (Neural Network) tale sistema deve essere validato, il più comune metodo per la validazione è indagare i residui (errore di predizione) attraverso una cross-validazione sul data set. Le funzioni nnvalid e ifvalid includono le funzioni di autocorrelazione dei residui e la cross-corrrelazione tra il segnale di controllo e i residui. Queste funzioni visualizzano tutto attraverso istogrammi mostrando la distribuzione dei residui. Tuttavia un modello lineare è estratto da NN ad ogni istante ciò è chiamato local istantanous linearization technique (vedi Norgaard).

```
[yhat,NSSE]=nnvalid('nnoe',NetDef,NN,w1,w2,y1,u1);
```

u,y sono i test set control mentre yhat è la predizione a un passo prodotta da NN mentre NSSE è il criterio di valutazione del test set (chiamato anche test error). La funzione indaga xcorrel indaga il numero di high-order crosscorrelation functions (Billings). Per selezionare la struttura del modello si deve risolvere un problema servo lineare per individuare l'ordine del modello lineare estratto dalla NN per identificare l'ordine della NN, quindi in questo caso il problema lineare è un problema servo per la NN, inoltre esiste un ulteriore funzione nnprune che cerca di eliminare i rami della NN che sono superflui.

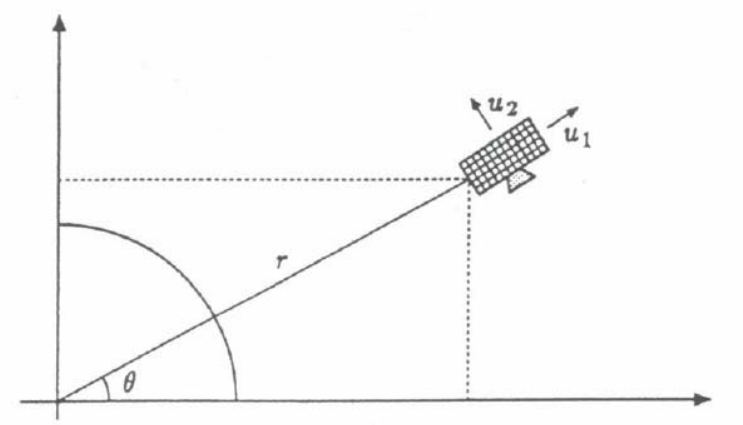
Simulazione

Abbiamo generato le nostre coppie ingresso uscita attraverso le seguenti istruzioni:

```
clear
global Kg
global ur
global uo
global c
global rumore
global y
global uc
uc=[];
y=[];
rumore=[];
ur=0;
uo=0;
c=0;
%global R
%global omega
W=1;
R=1;
omega=W;
A=[0 1 0 0
  3*W^2 0 0 2*W*R
  0 0 0 1
  0 -2*W/R 0 0];
B=[0 0
  1 0
  0 0
  0 1/R];
Tint=[0 10];
%Kgain= acker(A,B(:,2),[-1 -2 -3 -4]);
Kgain= place(A,B,[-1 -2 -3 -4]);
Kg=-Kgain;
[T,XX]=ode45('Simul',Tint,[0;0;0;0]);
plot(T,XX);
```

dove la funzione simul è la seguente:

```
function [XD]=Simul(t,X)
global Kg
global ur
global uo
global c
global rumore
global y
global uc
t
omega=1;
R=1;
k=omega*omega*R*R*R;
if(t-c>=0.1)
   ur=0.1*rand(1);
   uo=0.1*rand(1);
   v=[ur;uo];
   rumore=[rumore v];
   %c=c+0.1;
end
u1=Kg(1,1)*X(1)+Kg(1,2)*X(2)+Kg(1,3)*X(3)+Kg(1,4)*X(4)+ur;
u2=Kg(2,1)*X(1)+Kg(2,2)*X(2)+Kg(2,3)*X(3)+Kg(2,4)*X(4)+uo;
u=[u1;u2];
if(t-c>=0.1)
   uc=[uc u];
end
g1=X(2);
g2=(X(1)+R)*(X(4)+omega)^2-k/((X(1)+R)^2)+u(1);
g3=X(4);
g4=-2*(X(4)+omega)*X(2)/(X(1)+R)+(u(2)/(X(1)+R));
XD=[g1;g2;g3;g4];
if(t-c>=0.1)
   y=[y XD];
   c=c+0.1;
end
```



Questo modello simula il moto di un satellite che orbita attorno alla terra ad un'altezza R e che presenta una velocità angolare (ω), il sistema in considerazione è non lineare è stato linearizzato attorno a un punto di equilibrio.

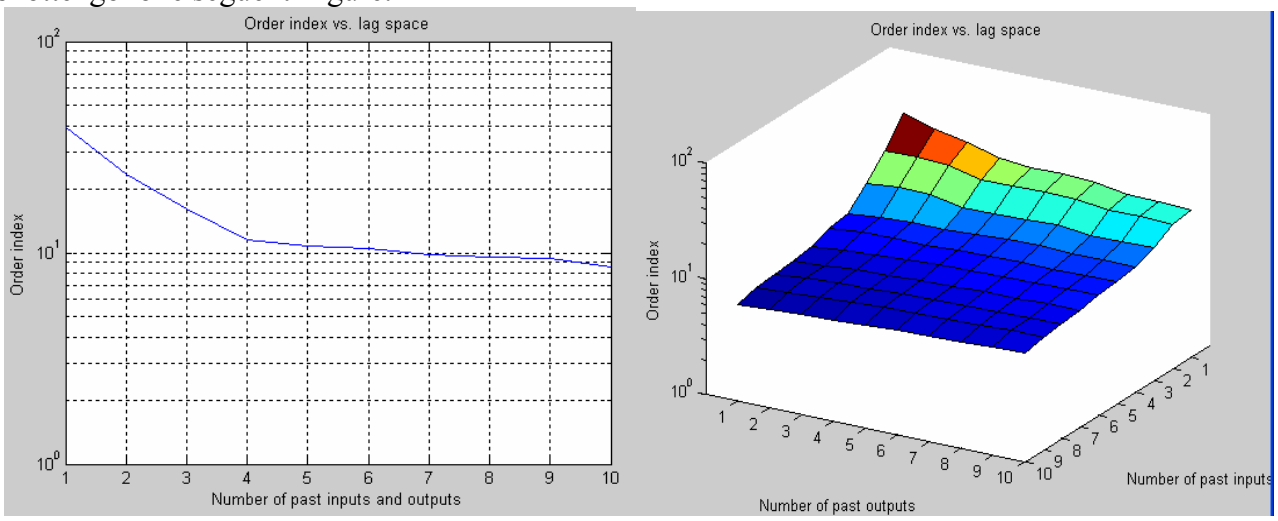
$$\ddot{r}(t) = r(t)\dot{\theta}^{2}(t) - \frac{k}{r(t)^{2}} + u_1(t)$$

$$\ddot{\theta} = \frac{-2\dot{\theta}(t)\dot{r}(t)}{r(t)} + \frac{1}{r(t)}u_2(t)$$

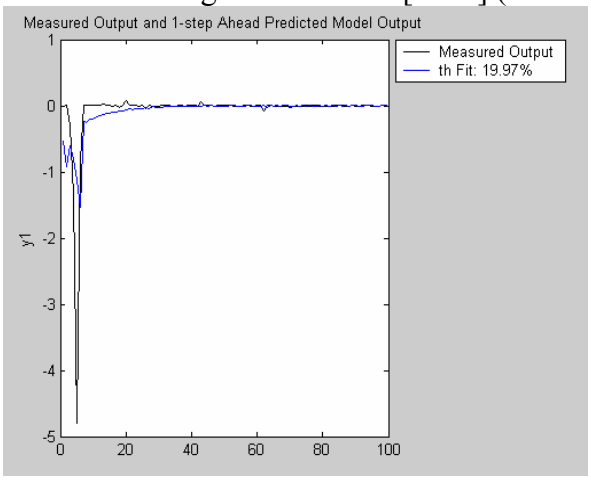
Abbiamo calcolato il vettore k che ci permette di realizzare una retroazione statica dallo stato, il controllore così ottenuto è stato usato per controllare il sistema non lineare allora a tale sistema (modello non lineare controllato attraverso una retroazione statica dallo stato) è stato applicato rumore bianco di varianza unitaria e media nulla, così facendo abbiamo ottenuto le nostre coppie ingresso uscite che saranno usate per allenare la nostra rete neurale. Per costruire la nostra rete neurale abbiamo bisogno di conoscere l'ordine del modello che si vuole approssimare attraverso una rete neurale, per far ciò si deve risolvere un servo problema lineare. Utilizzando le seguenti istruzioni possiamo avere un'idea dell'ordine del modello che stiamo cercando di approssimare:

```
clear
load dati.dat;
u1=dati(1,:);
y1=dati(2,:);
clear dati
load datiVal.dat;
u2=datiVal(1,:);
y2=datiVal(2,:);
clear datiVal
[u1s,uscales]=dscale(u1);
[y1s,yscales]=dscale(y1);
u2s=dscale(u2,uscales);
y2s=dscale(y2,yscales);
OrderIndices=lipschit(u1s,y1s,1:10,1:10);
NN=[9 9 1];%ordine
th=oe([y1' u1'],NN);
present(th);
figure(3)
compare([y2' u2'],th,1);
figure(4);
resid([y2' u2'],th);
```

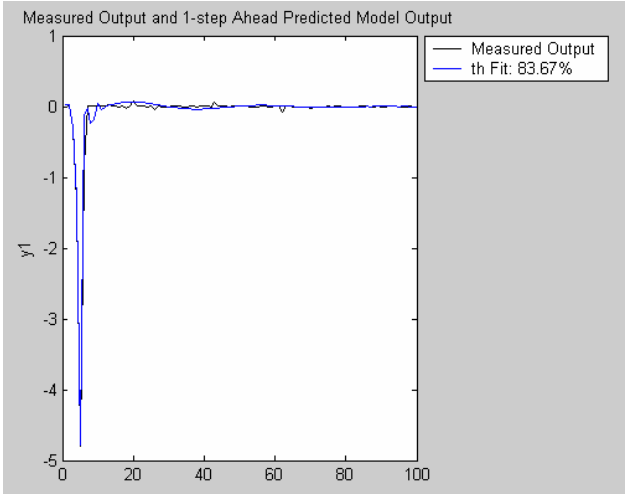
si ottengono le seguenti figure:



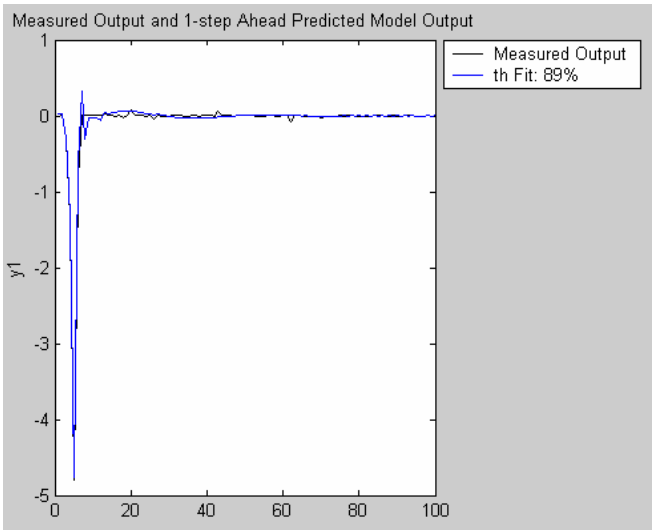
l'immagine a destra è poco significativa mentre quella a sinistra fa vedere come aumentando l'ordine del modello l'indice di perdita diminuisce, il test successivo che andremo a compiere è quello di comparare l'uscita dell'impianto con quella simulata dal sistema, cominciamo con un sistema, che presenta il seguente ordine di grandezza NN=[2 2 1] (na nb nc), si ottiene:



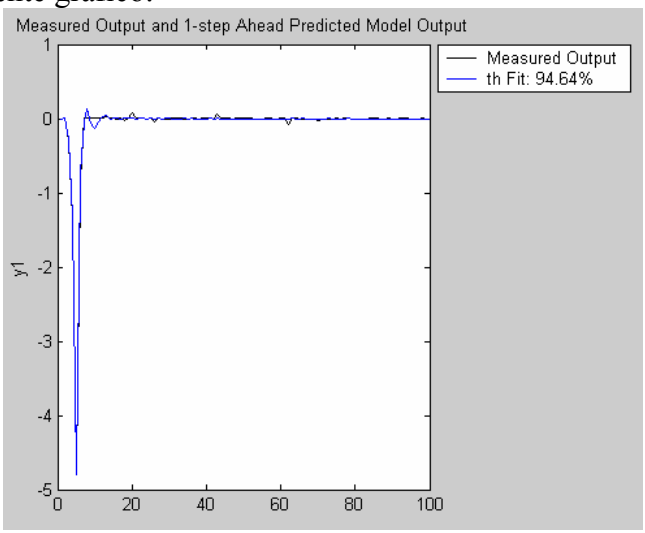
il seguente grafico in cui come si può notare si ha una fit che sfiora il 20%, allora si incrementa ulteriormente l'ordine del modello sperando che la fit del sistema aumenti, il nuovo ordine di grandezza è $NN = [7\ 7\ 1]$ per cui si ha:



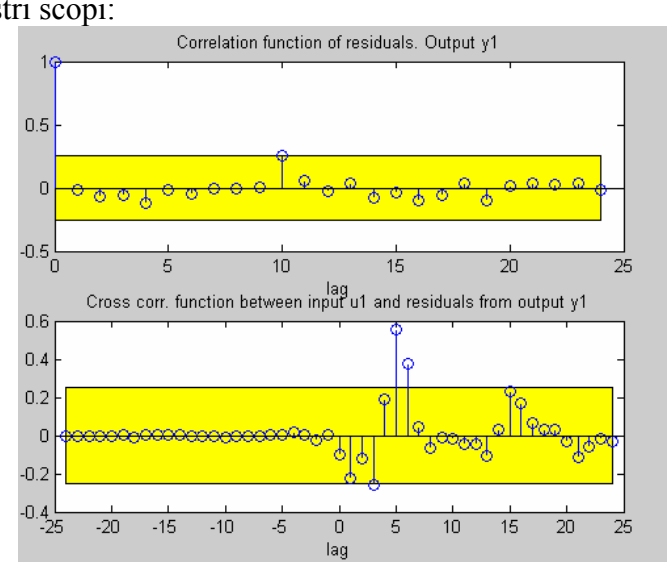
che la fit ha un valore di quasi del 84%, tale valore è gia un ottimo valore per la fit, si aumenta ancora l'ordine del modello sperando che la fit del sistema aumenti ancora il nuovo ordine del modello è NN=[9 9 1], per cui si ha il seguente grafico in cui la fit è del 89%:



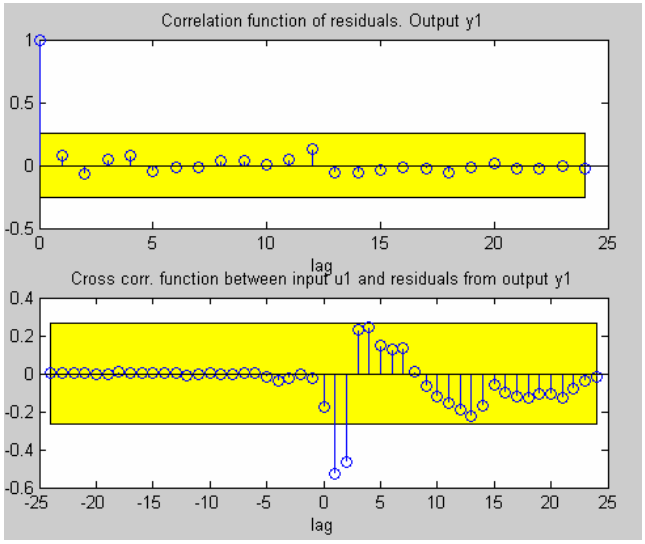
Non ancora contenti si aumenta ulteriormente l'ordine del modello per cui NN=[10 10 1], per tale sistema si ottiene il seguente grafico:



per quest'ultimo valore del vettore NN si realizza la fit maggiore (quasi 95%) ma se si analizza l'autocorrelazione e la crosscorrelazione sui residui si nota che quest'orine di grandezza è incompatibile, per i nostri scopi:



infatti presenta degli andamenti anomali nella crosscorrelazione, ciò è meno evidente se si analizzano i residui per NN=[9 9 1]: per cui si ha il seguente grafico:



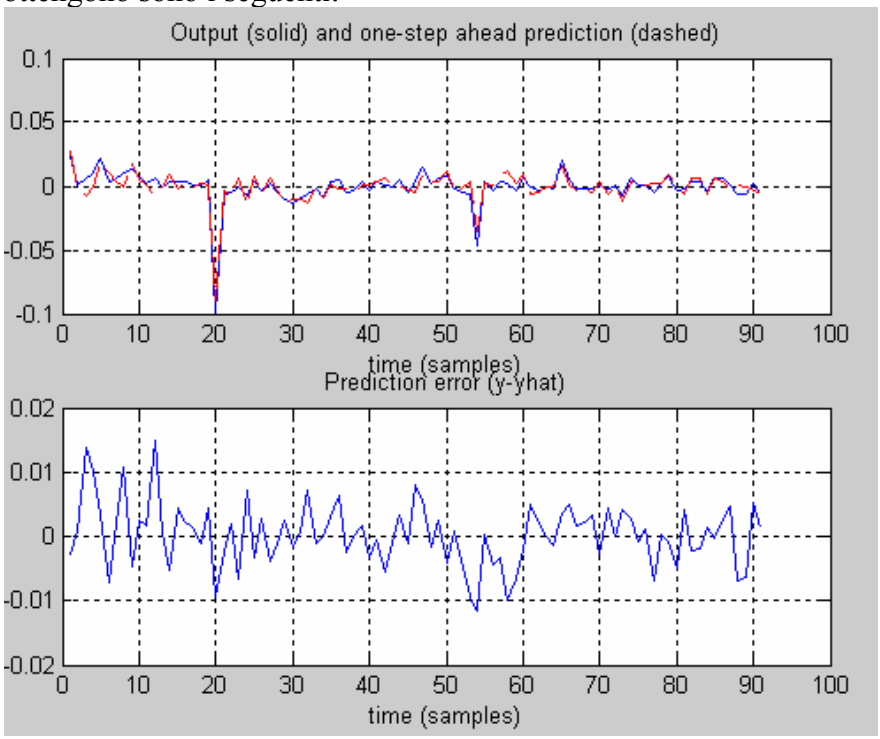
da ciò si deduce che l'orine del nostro modello è NN=[9 9 1]. Fissato l'ordine del nostro modello si procederà alla fase successiva scegliere il modello della rete neurale poiché la sua struttura è formata da due strati uno nascosto formato da 10 neuroni con sigmoide di attivazione a tangente iperbolica e un unico strato di uscita ad attivazione lineare. Il modello che si è scelti è quello dell'Output Error, in cui il modello assume la seguente forma:

$$\hat{y}(t|\theta) = g\left(\hat{y}(t-1|\theta),\ldots,\hat{y}(t-n_a|\theta),u(t-n_k),\ldots,u(t-n_b-n_k+1)\right)$$

utilizzando le seguenti istruzioni si cerca di allenare la nostra NN (Neural Network):

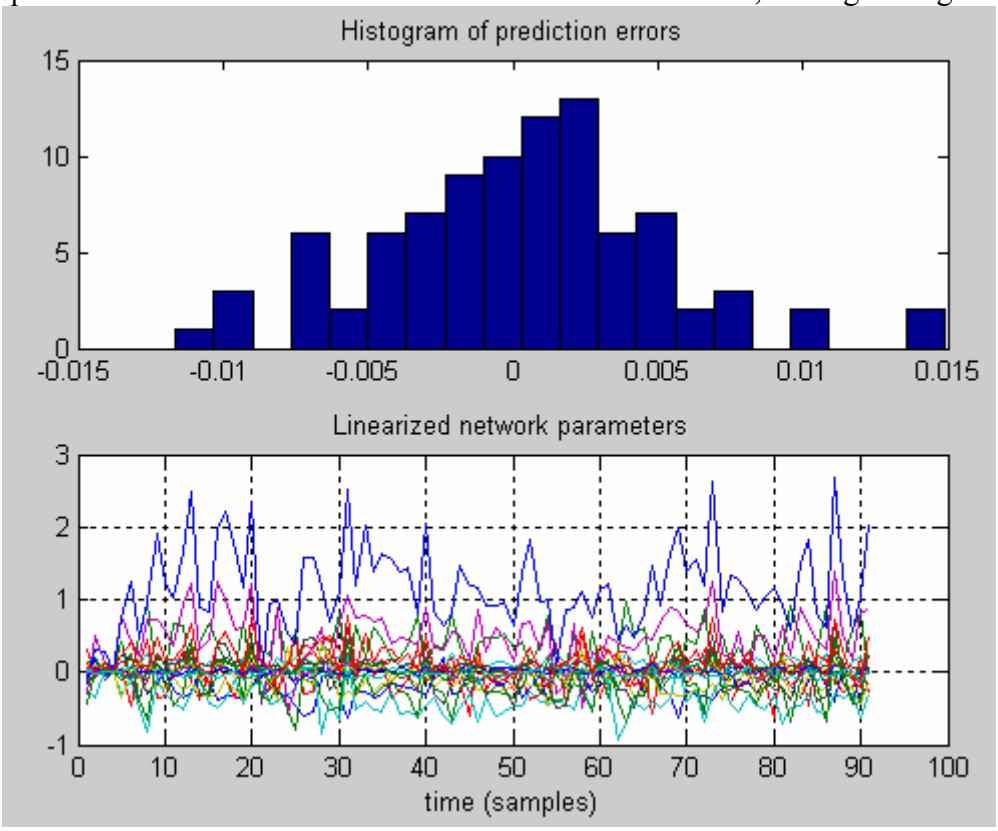
```
NetDef=['HHHHHHHHHH';'L---------'];
trparms=settrain;
trparms=settrain(trparms,'maxiter',300,'D',1e-4,'skip',10);
[W1,W2,NSSEvec]=nnoe(NetDef,NN,[],[],trparms,y1s,u1s);
[w1,w2]=wrescale('nnoe',W1,W2,uscales,yscales,NN);
[yhat,NSSE]=nnvalid('nnoe',NetDef,NN,w1,w2,y1,u1);
```

i risultati che si ottengono sono i seguenti:

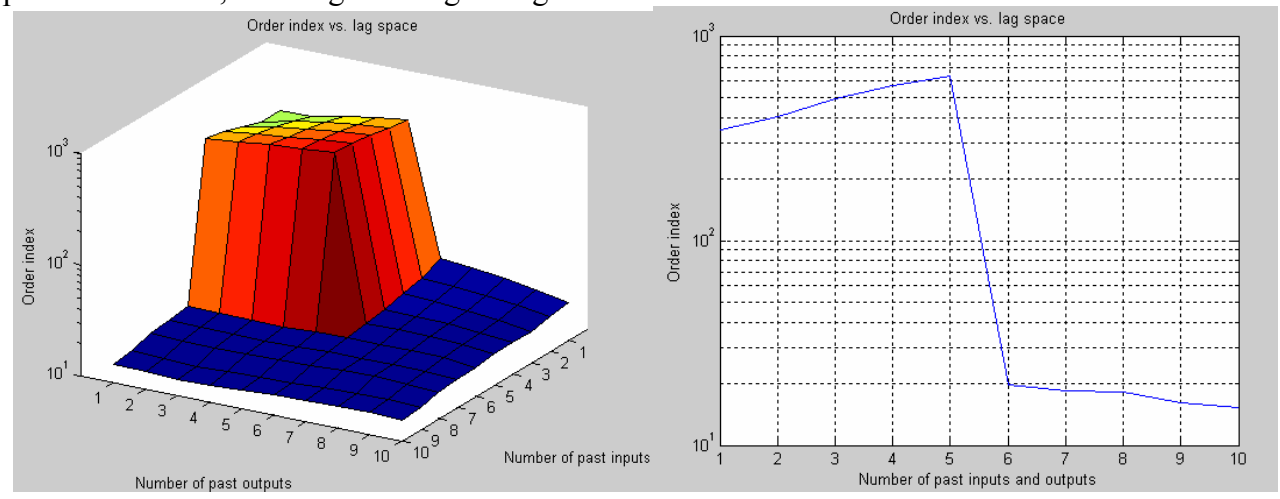


in questo primo grafico in blu è rappresentata l'uscita del sistema mentre in rosso la predizione ad un passo come si può notare sono due cose completamente diverse cioè la rete neurale che è stata costruita non è in grado di riprodurre il sistema non lineare l'altra figura nel grafico precedente

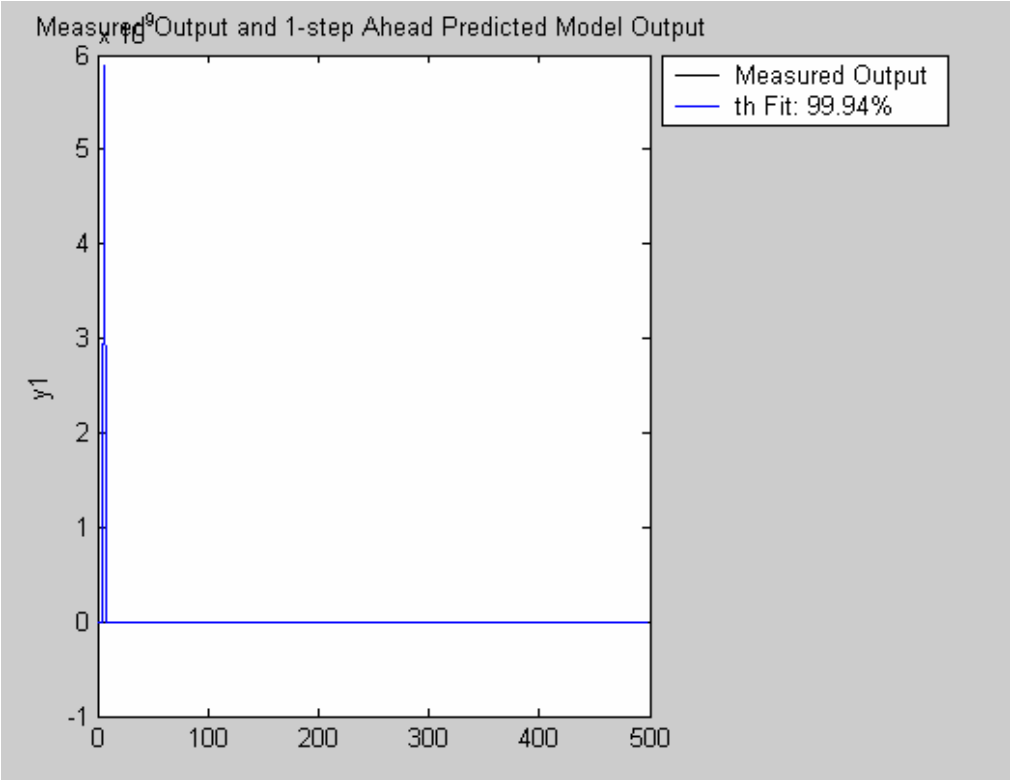
raffigura l'errore di predizione che è dello stesso orine di grandezza della nostra uscita ciò significa che l'uscita predetta non coincide con l'uscita del modello non lineare, dal seguente grafico:



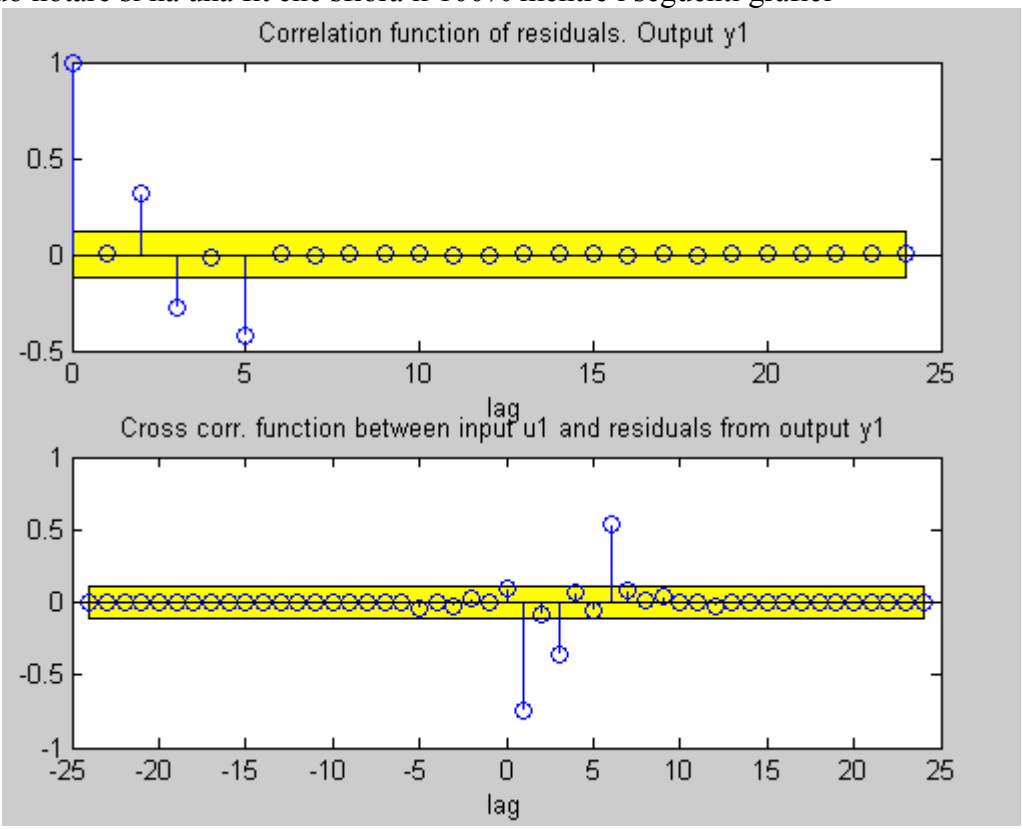
si può notare come è distribuito l'errore di predizione ma si nota anche che i parametri della nostra rete neurale sono instabili, cioè non convergono ad alcun valore costante. Da ciò si deduce che il numero di campioni che abbiamo usato per allenare la nostra rete neurale è insufficiente, quindi si estende il nostro data set di campioni che passa da cento coppie a cinquecento. Allora si devono ripetere le stesse operazioni fin qui effettuate, quindi prima di tutto dobbiamo risolvere il servo problema lineare, si ottengono i seguenti grafici:



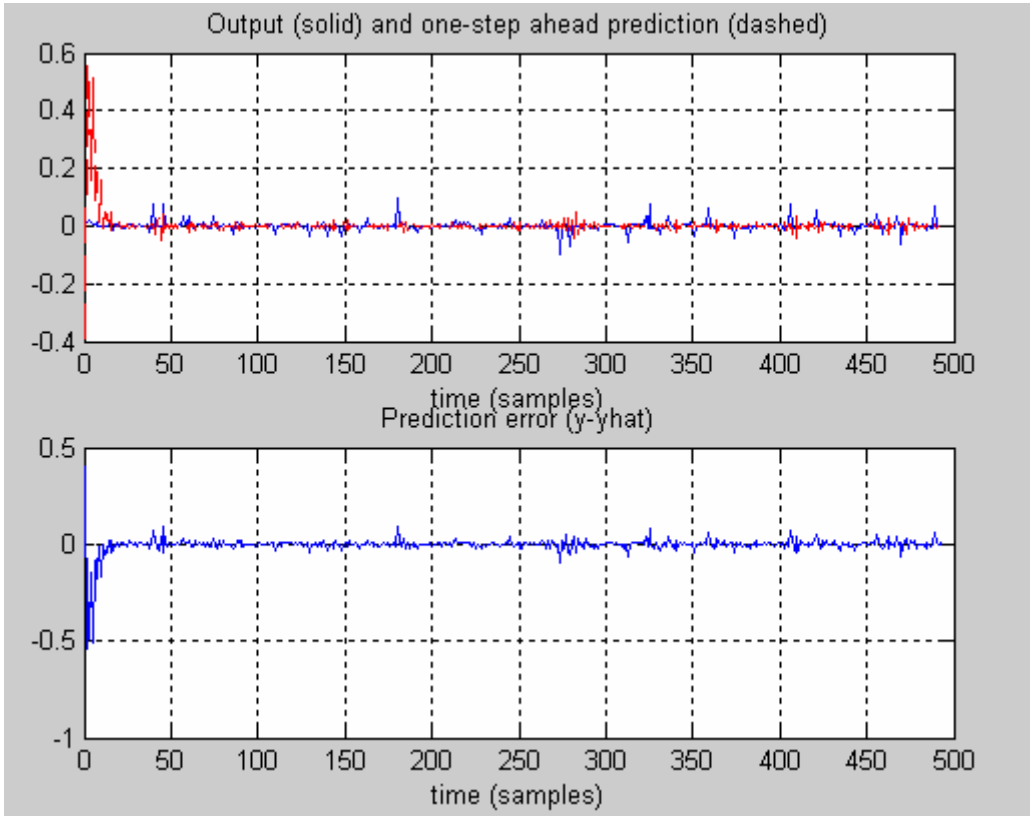
da questi grafici si deduce che l'ordine del modello è NN=[6 6 1] poiché applicando il principio di parsimonia per tale valore si ha il più basso valore dell'indice di perdita. Se per tale modello si analizza sia la fit che la funzione di crosscorrelazione tra u e y e la funzione di autocorrelazione dei residui, si hanno i seguenti grafici:



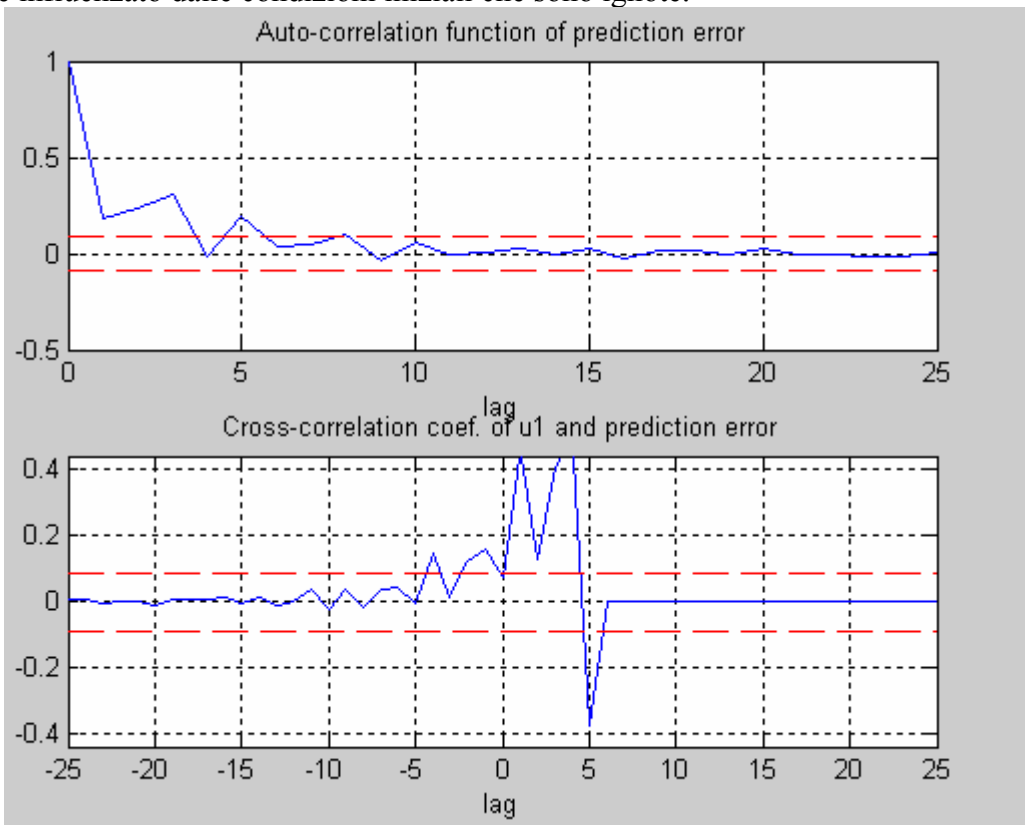
come si può notare si ha una fit che sfiora il 100% mentre i seguenti grafici



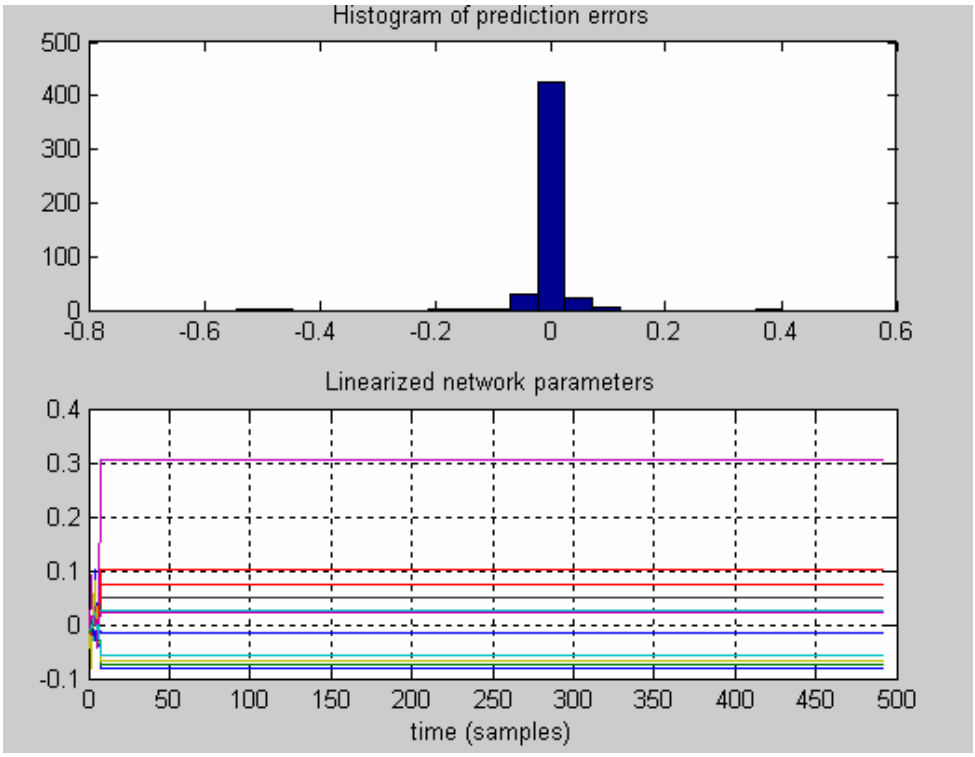
sono stati ignorati, anche se la funzione di autocorrelazione che quella di crosscorrelazione non sono ciò che di meglio si può chiedere, poiché si desidera che gli errori che si commettono durante il calcolo dei residui siano non correlati tra di loro e anche non correlati con il segnale d'ingresso, si è scelto che questo sarà il nostro modello, allora $NN = [6\,6\,1]$. Adesso non ci resta che allenare la nostra rete e validarla con un altro data set di elementi di ugual lunghezza (500) solo per congruenza, poiché ciò non è necessario. I risultati che si ottengono sono i seguenti:



come si può notare l'errore che si commette è trascurabile se si esclude il transitorio iniziale che è fortemente influenzato dalle condizioni iniziali che sono ignote.



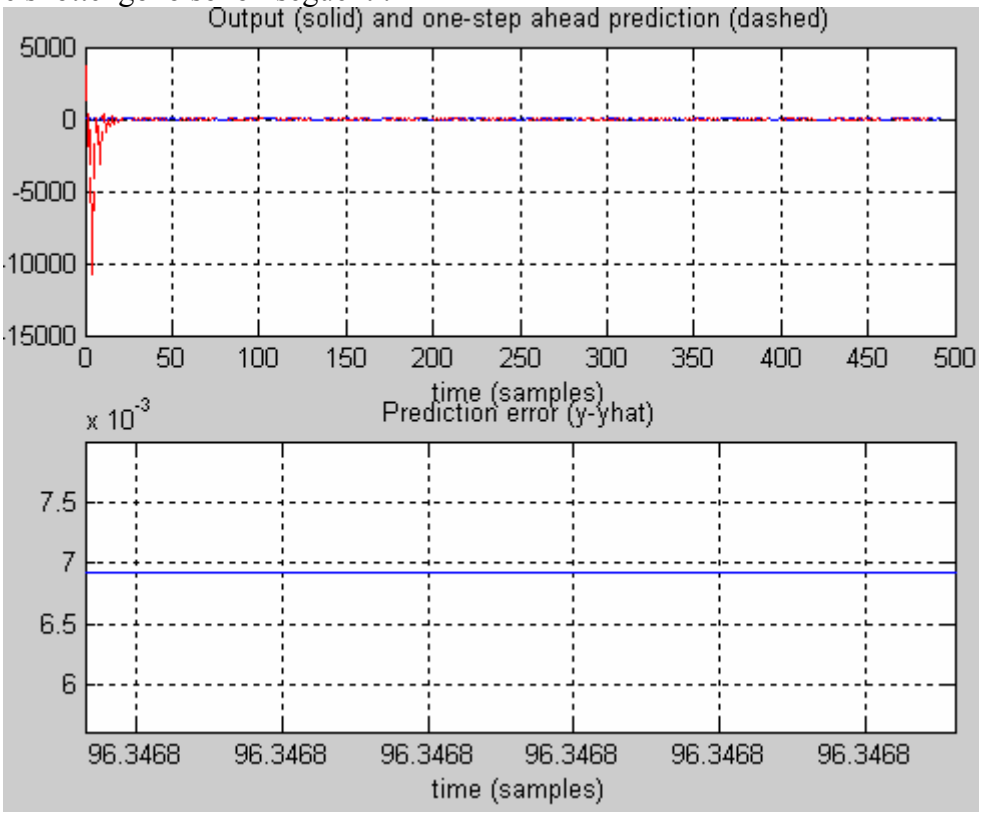
Queste sono le funzioni di auto cross correlazione che sono molo simili a quelle del sistema lineare, mentre per quanto riguarda l'errore di predizione che si commette e la convergenza dei parametri della rete neurale sono rappresentate nel seguente grafico:



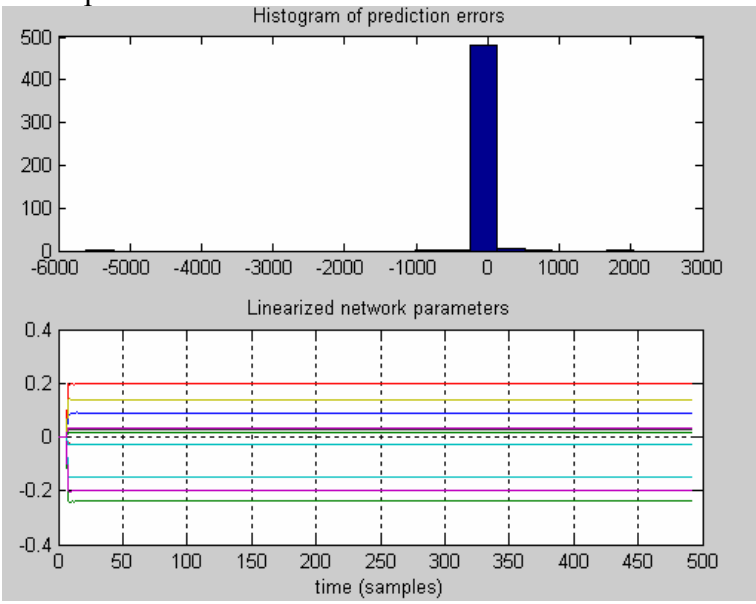
come si può notare l'errore di predizione è distribuito come una normale di media zero e varianza 0.1 che è la stessa del rumore che agisce sul sistema, mentre come si può vedere i parametri convergono molto velocemente, quindi il test successivo è inserire una nuovo data set in ingresso alla rete neurale e confrontare l'uscita simulata con quella misurata sottoponendo alla rete neurale opportuni valori dell'ingresso oppure se ciò non accade, per realizzare ciò si utilizza la seguente istruzione che non fa altro che validare la rete neurale:

```
[yhat,NSSE]=nnvalid('nnoe',NetDef,NN,w1,w2,y2,u2);
```

I risultati che si ottengono sono i seguenti:



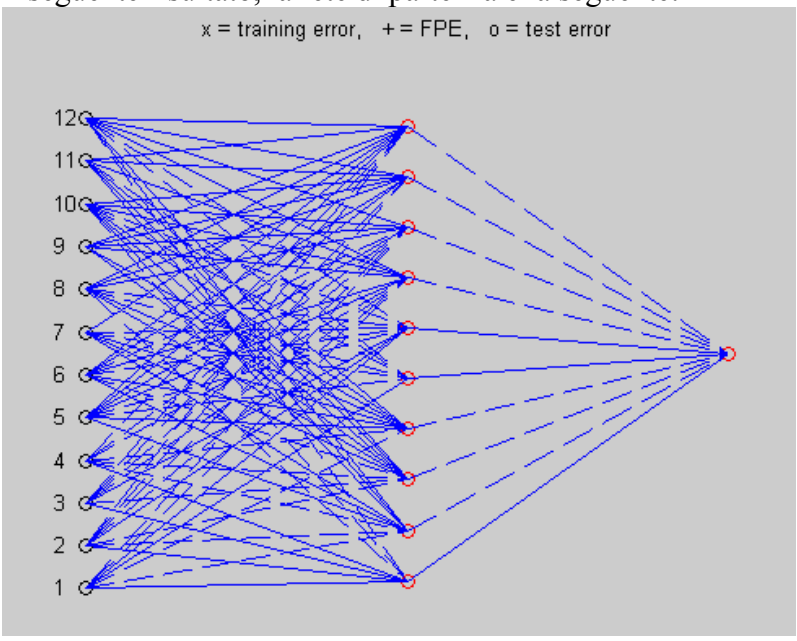
come si vede l'uscita simulata segue fedelmente quella simulata tranne in un breve transitorio iniziale, tale fenomeno è da imputarsi allo stato iniziale dell'impianto che è incognito, mentre a regime l'errore di predizione che si commette è dell'ordine di 10^{-3}, se si analizza l'istogramma dell'errore di predizione si può notare:



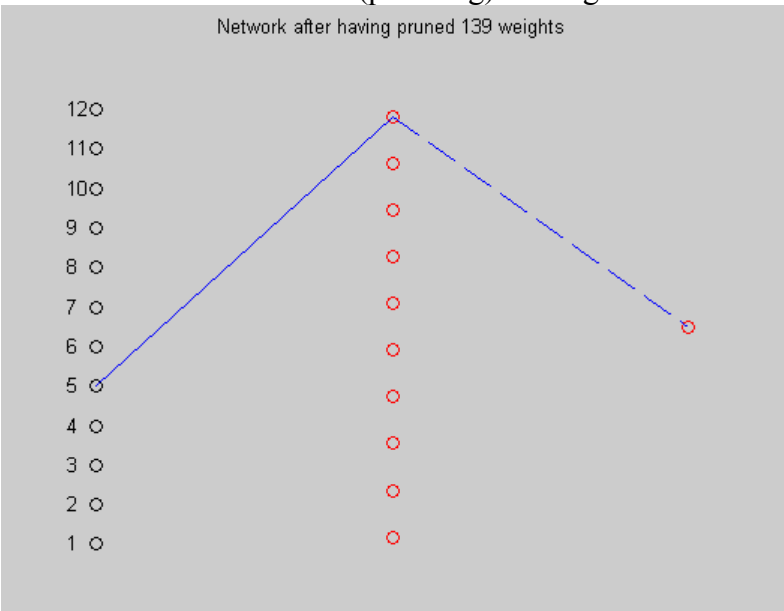
che è distribuito come una normale. Utilizzando le seguenti istruzioni si può tentare di eliminare dalla NN i rami superflui:

```
prparms=[50 0];
[thd,trv,fpev,tev,deff,pv]=nnprune('nnoe',NetDef,W1,W2,u1s,y1s,NN,trparms,prparms,u2s,y2s);
figure(1);
set(gca,'Ylim',[0 0.25]);
[mintev,index]=min(tev(pv));
index=pv(index)
[W1,W2]=netstruc(NetDef,thd,index);
trparms=settrain(trparms,'D',0);
[W1,W2,NSSEvec]=nnoe(NetDef,NN,W1,W2,trparms,y1s,u1s);
[w1,w2]=wrescale('nnoe',W1,W2,uscales,yscales,NN);
[yhat,NSSE]=nnvalid('nnoe',NetDef,NN,w1,w2,y1,u1);
figure(9);
ylen=length(y1);
plot(y1(3:ylen))
hold on
plot(yhat,'m--')
hold off
```

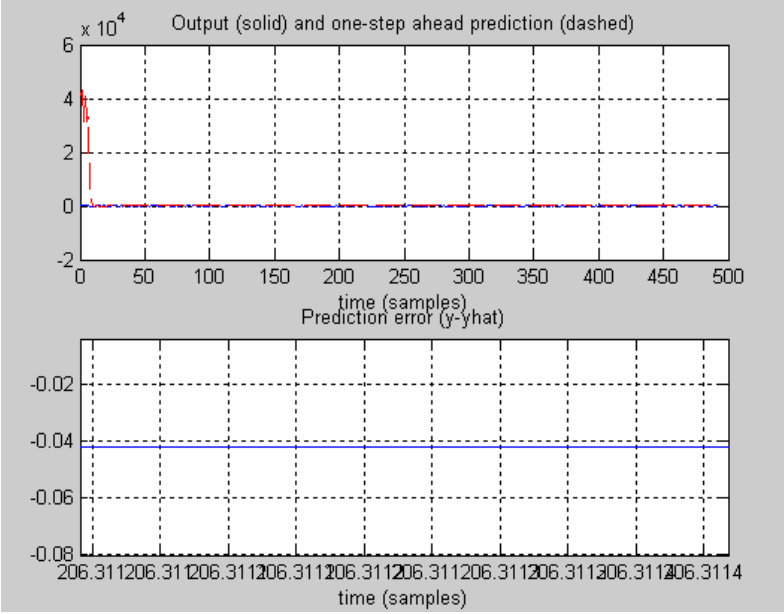
ciò che si ottiene è il seguente risultato, la rete di partenza è la seguente:



mentre la NN che viene fuori dallo sfoltimento (prunning) è la seguente:



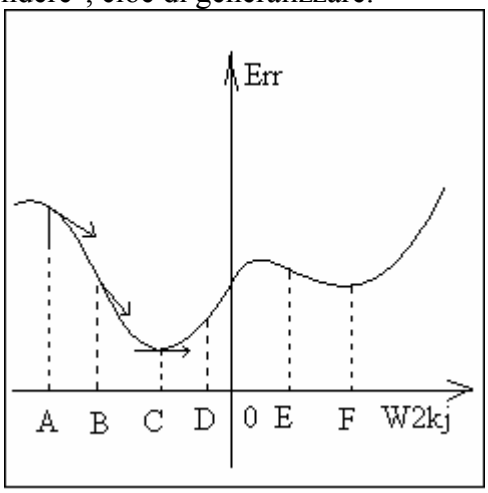
come si può notare tale rete è molto meno complessa rispetto alla precedente, inoltre validando quest'ultima rete si può notare che l'errore di predizione sicuramente peggiora ma è sicuramente accettabile:



è sempre presente un transitorio iniziale imputabile alla condizione iniziale da cui parte il sistema simulato.

Problemi relativi alle Reti Neurali

Purtroppo, non esiste una formula che permette di stabilire quanti neuroni dello strato nascosto siano necessari per emulare efficacemente una determinata funzione obiettivo, e a volte si tratta anche di grandi quantità. E' questo un grosso problema per le funzioni non lineari: aggiungendo o togliendo anche pochi neuroni dallo strato nascosto si possono ottenere grandi variazioni nella capacità di apprendimento della particolare funzione, quale la convergenza verso la funzione obiettivo o la perdita di efficienza, o problemi di risonanza, autooscillazioni ed altre anomalie che impediscono alla rete di "apprendere", cioè di generalizzare.



Un caso comune, é determinato dal fatto che, essendo assegnati inizialmente a caso il valore dei pesi (quando la rete è stata appena creata e non é stata ancora istruita), la funzione di errore per un determinato elemento potrebbe inizialmente cadere nel punto E come si può vedere in figura. L'algoritmo Error Back Propagation troverebbe in questo caso il suo minimo nel punto F, che non é il minimo assoluto della funzione per quanto riguarda quell'elemento, ed il metodo delle derivate impedisce all'algoritmo di superare la "sella" che separa i punti E o F dal punto C, perchè una volta raggiunto il minimo F la derivata é zero e non si hanno quindi più variazioni nel Peso sinaptico.
Per guidare la macchina nella scelta dei valori dei Pesi Sinaptici, durante la fase di addestramento si usano quindi esempi basati sulla funzione obiettivo. A ogni esempio la macchina raffina i propri Pesi Sinaptici in modo da far corrispondere ingressi e uscite in maniera appropriata. Quando la macchina raggiunge un posizionamento che attua nel modo migliore la funzione obiettivo, in pratica l'ha "appresa".
La rete può quindi essere commutata in modo di esecuzione: se si presentano ai suoi ingressi dei valori il cui campo di esistenza é compreso nel campo di valori su cui si é addestrata la macchina, la risposta dovrebbe essere corretta, anche nel caso che un caso specifico non era mai stato presente nel set di dati di addestramento, cioè un caso del tutto nuovo.
Si ricordi che la rete non sa nulla della funzione che sta cercando di apprendere, se non ciò che "vede" nel set di dati di addestramento. Se i dati sono carenti (perché gli esempi sono troppo pochi oppure perché contengono informazioni irrilevanti), la rete non riesce a compiere generalizzazioni adeguate. Può anche darsi, infatti, che gli esempi non contengano tutte le informazioni importanti che definiscono la funzione obiettivo.
L'insieme degli esempi ingresso-uscita distingue completamente una funzione obiettivo dall'altra; la natura degli esempi sottoposti alla rete non ha importanza, sia che si tratti di valori di parametri relativi ad un proceso fisico, o chimico, o una semplice funzione matematica...
Con questi procedimenti l'automazione viene portata a un livello più alto rispetto a quello solito: non solo si usa il computer per svolgere un compito ripetitivo, ma si rende automatico il procedimento stesso che crea il sistema capace di svolgere quel compito. Allora l'uomo non ha neanche la necessità di sapere quale formula é applicata per risolvere la funzione obiettivo.

Dopo aver descritto la problematica dal punto di vista della struttura logica di calcolo (il "contenitore") andiamo a considerare alcuni aspetti relativi ai dati da trattare (il "contenuto").
Gli esempi offrono abbastanza informazioni perché la macchina risponda in modo adeguato ai nuovi ingressi? Non è detto. Che la rete si comporti bene di fronte agli esempi di addestramento non sempre significa che si comporterá altrettanto bene di fronte a un caso mai incontrato prima.
Se gli esempi ingresso-uscita a disposizione non contengono le informazioni cruciali, non è detto che la macchina acquisisca le competenze volute.
Per fortuna, spesso le informazioni necessarie possono essere aggiunte sotto forma di uno stratagemma chiamato "esempio forzato".
Gli "esempi forzati" possono accelerare l'apprendimento. Come nel gioco delle venti domande, in cui le risposte a certi quesiti elementari possono restringere di molto la ricerca, pochi esempi "strategici" possono decidere tra l'apprendimento e il mancato apprendimento di una funzione.
Per essere sicuro che il miglioramento di efficienza raggiunto dalla rete neurale implementata fosse dovuto alle informazioni contenute negli esempi forzati, si è tentato d’ingannare la macchina con due alternative. La prima erano esempi privi di informazioni: cioè alla macchina sono state fornite informazioni casuali. La macchina non ne trasse alcun vantaggio, poiché le sue prestazioni furono circa le stesse che in assenza di questi esempi. In seguito sono stati forniti alla macchina degli esempi contenenti informazioni deliberatamente errate. In questo caso le prestazioni peggiorarono subito, come ci si aspettava. Gli esempi forzati si erano rivelati effettivamente utili.
Non è necessario che gli esempi siano reali. Si possono usare esempi virtuali, purché non si chieda alla macchina di prendere la decisione giusta rispetto a un processo "reale" quanto di agire in modo coerente con il set di dati di addestramento.
Per esempio nella visione artificiale, il cui scopo è quello di riconoscere gli oggetti, si possono fornire molti esempi basati sull'invarianza, cioè sul fatto che un oggetto rimane se stesso anche quando cambia posizione all'interno del campo visivo oppure quando cambia dimensioni.
Supponiamo quindi di voler addestrare un sistema di visione artificiale a riconoscere immagini di alberi. Può darsi che non si riesca a specificare in termini matematici precisi un metodo consistente per identificare un albero: in tal caso non si riesce a strutturare il problema e ad assegnare alla macchina regole rigide. Se ci si limita a mostrare alla macchina fotografie di alberi e di altri oggetti, le si forniscono informazioni, ma non le si dice tutto quello che si sa. Per esempio si sa che un albero resta tale anche se subisce una traslazione o un cambiamento di scala. Gli esseri umani lo intuiscono benissimo, ma la macchina no, a meno che non le si comunichino le informazioni in maniera esplicita. In assenza di esempi congruenti la macchina potrebbe impiegare un tempo lunghissimo, o addirittura infinito, per conseguire questa "comprensione".
L'algoritmo di apprendimento adattativo cerca di mettere a punto i Pesi Sinaptici W della rete neurale in modo che essi concordino simultaneamente con tutto l'insieme dei dati di addestramento. Ma solitamente una soluzione perfetta è impossibile, quindi si deve ricercare un compromesso. A questo scopo la macchina deve valutare a ogni passo il grado di accordo raggiunto. Durante la fase di apprendimento, determinati esempi possono essere appresi meglio di altri (per esempio a causa di un particolare numero di neuroni dello strato nascosto, che lo porta a "risuonare" con un particolare insieme di esempi che a loro volta costituiscono un "ottimo locale"), ma se il programma riesce a individuare l'esempio che è stato appreso nel modo peggiore, all'iterazione successiva può dedicargli più attenzione.
La tecnologia delle reti neurali deve quindi affrontare ancora molti problemi. Le difficoltà più gravi derivano dalla tendenza che hanno le Reti a "sovrapprendere" nella fase di addestramento, tendenza che può compromettere il buon funzionamento del programma. Si ha sovrapprendimento quando la macchina impara "a memoria" gli esempi di addestramento a scapito della generalizzazione.
Nel cercare un posizionamento ottimo dei pesi sinaptici (il cosiddetto ottimo globale) talvolta l'algoritmo di apprendimento finisce in una configurazione meno buona (un ottimo locale), che è migliore di altre soluzioni ma non coincide con ciò che di meglio si potrebbe in teoria conseguire.

In generale non vi è un metodo efficace per evitare gli ottimi locali. Alcuni problemi di apprendimento sono risultati NP-completi, termine tecnico che indica una classe di problemi di calcolo per i quali si ritiene che il conseguimento dell'ottimo globale richieda un tempo di calcolo eccessivo. In pratica tuttavia questo problema non ha conseguenze troppo gravi: se la macchina raggiunge un buon punto di ottimo locale, di solito la prestazione risulta soddisfacente.
Nonostante sia ancora afflitto da questi problemi, l'apprendimento artificiale si è dimostrato valido nella risoluzione di un'ampia gamma di problemi pratici.

Bibliografia

Bittanti Sergio, *"Identificazione dei modelli e controllo adattativo"*, Pitagora editrice Bologna

Lazzerini Beatrice, *"Introduzione alle Reti Neurali"*

Poma Salvatore, *"Reti Neurali"*

Magnus Nørgaard, *"N*eural Network Based System Identification Toolbox for use with Matlab*"*

Indice