Reti Neurali Artificiali
Tutorial
(Draft)

Crescenzio Gallo #1, Michelangelo De Bonis #2
IEEE MEMBERS

#Dipartimento di Scienze Biomediche, Università degli Studi di Foggia
[1]c.gallo@ieee.org, [2]m.debonis@ieee.org

The formulation of a problem is far more often essential than its solution, which may be merely a matter of mathematical or experimental skill.

Albert Einstein

***Sommario*—L'obiettivo di questo breve tutorial è di introduzione ad una potente classe di modelli matematici: la rete neurale artificiale. In realtà, questo è un termine molto generico che comprende molti diversi modelli di tipo matematico e varie tipologie di approcci. Scopo di questo tutorial non è di esaminarli tutti ma di far comprendere le funzionalità e le possibili implementazioni di questo potente strumento. Inizialmente si introdurranno le reti, per analogia, con il cervello umano. L'analogia non è molto dettagliata, ma serve ad introdurre il concetto di calcolo parallelo e distribuito. Succesivamnte si analizzerà un tipo di rete neurale nel dettaglio: la rete *feedforward* con algoritmo di calcolo dell'errore *backpropagation*. Si discuterà delle architetture dei modelli, i metodi dell'apprendimento e della rappresentazione dei dati. Nella sezione finale sarà presentato una serie di applicazioni ed estensioni al modello di base.**

I. INTRODUZIONE

Le reti neurali artificiali sono dei sistemi di elaborazione delle informazioni. I modelli che usano le reti neurali artificiali sono strutture che servono per fornire il legame tra i dati di ingresso-uscita. Esse offrono interessanti aspetti di applicazione: quando, per esempio, si vogliono rilevare con dei sensori delle grandezze intangibili, queste vanno scomposte in grandezze tangibili e cioè rilevabili con i sensori esistenti. I dati forniti da questi sensori vanno poi opportunamente correlati fra loro [1; 2; 3].

La correlazione esistente tra le grandezze rilevate è spesso ignota a priori, il che rende la rilevazione di una grandezza intangibile un problema generalmente difficile. L'impiego di sensori elementari in unione ad un sistema a rete neurale artificiale rende possibile la determinazione della correlazione cercata e quindi possibile il controllo del sistema di automazione.

Il sistema a rete neurale, a somiglianza della mente umana, è in grado di elaborare grandi quantità di dati d'ingresso, apparentemente tra loro non correlati, e di produrre in uscita una decisione utilizzabile. Questa sua capacità di apprendimento è la caratteristica fondamentale che rende possibile scoprire nella massa dei dati la correlazione cercata.

La prima fase per creare il modello del sistema consiste nella raccolta di tutti i dati necessari e questa è certamente la fase che richiede più tempo perché vanno esplorate tutte le possibilità. Ad esempio, nelle industrie alimentari, dove si vuole rilevare la grandezza intangibile costituita dal sapore, vanno raccolti innumerevoli campioni di caratteristiche diverse. Da un lato i campioni vanno analizzati con i sensori tradizionali di grandezze tangibili, come il pH, gli ingredienti chimici, il colore ed altri parametri, e vanno raccolti i dati relativi al processo di produzione come il tempo di cottura, la temperatura, ecc... Dall'altro lato i campioni vanno esaminati e valutati empiricamente sulla base del giudizio soggettivo di esperti umani. Infine va fatta una classifica in modo da definirne il limite di accettabilità.

La seconda fase consiste nel fare apprendere alla rete neurale, sulla base dei dati raccolti nella prima fase (*training set*), le relazioni esistenti tra i dati d'ingresso forniti dai sensori e le uscite desiderate fornite dagli esperti umani. Anche questa fase di addestramento del sistema può richiedere molto tempo. Se l'uscita richiesta è un sapore particolare, il sistema deve scoprire ed imparare la relazione che esiste tra tutti i dati d'ingresso relativi alle grandezze tangibili e l'uscita corrispondente a quel determinato sapore. In questa fase si scoprono anche quali ingressi non hanno alcuna influenza sul risultato d'uscita e si può cosi procedere alla loro eliminazione. Questa fase è detta di apprendimento e può essere di diversi tipi:

- **supervisionato** (*supervised learning*): quando il *training set* contiene sia i dati di ingresso che i relativi dati di uscita reali;
- **non supervisionato** (*unsupervised learning*): i valori del modello della rete neurale artificiale vengono modificati solo in funzione dei dati di ingresso.

La terza fase, l'ultima, ha lo scopo di provare e convalidare il sistema completo su un insieme di dati (*validation set*).

Ovviamente la prova sarà significativa se si sottoporranno all'esame del sistema campioni diversi da quelli usati nell'addestramento. È questo un criterio di verifica generale e fondamentale della fase di apprendimento sia degli esseri umani sia delle macchine: "si è davvero imparato se si sanno risolvere problemi diversi da quelli usati come esempi durante l'apprendimento". Questo ultimo passaggio viene anche chiamato di *generalisation set*.

Se le uscite del sistema non corrispondono ai risultati forniti dagli esperti umani, può darsi il caso che siano stati trascurati dei fattori d'influenza e quindi il sistema deve essere sottoposto ad un nuovo ciclo di addestramento.

Solo quando i risultati del sistema e degli esperti umani risultano compatibili, il modello può essere sviluppato in un prodotto con i relativi hardware e software.

Nonostante questi spettacolari successi l'uomo deve restare conscio dei limiti di questa tecnologia. La rappresentazione del sapere degli esperti in una base di conoscenza comporta sempre, a causa della riduzione ad aspetti esclusivamente formali, una più o meno grande semplificazione.

Nello specifico, 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. È necessario poi scrivere un apposito programma, più o meno complesso, per gestire od 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 perché ci sono troppe variabili, o perché 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.

II. L'ELABORAZIONE NEL CERVELLO

A. Il cervello: un sistema di elaborazione delle informazioni

Le reti neurali artificiali cercano di simulare all'interno di un sistema informatico il funzionamento di sistemi nervosi biologici del cervello.

Il cervello umano contiene circa 10 miliardi di cellule nervose dette neuroni. In media, ogni neurone è collegato ad altri neuroni attraverso circa 10000 sinapsi (i dati effettivi variano notevolmente a seconda della neuroanatomia locale). La rete cerebrale di neuroni forma un massiccio sistema parallelo di elaborazione delle informazioni. Ciò contrasta con l'architettura dei computer convenzionali, in cui un singolo processore esegue una singola serie di istruzioni.

Se consideriamo il tempo impiegato per ogni operazione elementare i neuroni operano tipicamente ad una velocità massima di circa 100Hz, mentre una CPU convenzionale[1] effettua diverse centinaia di milioni di istruzioni al secondo. Il cervello quantunque sia costituito da un hardware molto lento, rispetto ai processori artificiali, ha notevoli capacità:

- le sue prestazioni tendono a peggiorare "dolcemente" in caso di danni parziali. Al contrario, la maggior parte dei programmi e dei sistemi artificiali sono molto fragili: se si rimuovono alcune parti arbitrarie, molto probabilmente, il tutto cesserà di funzionare;
- può imparare (riorganizzarsi) attraverso l'esperienza;
- il recupero parziale dei danni è possibile se le unità sane possono imparare ad assumere le funzioni precedentemente svolte dalle aree danneggiate;
- esegue i calcoli in modo altamente parallelo ed estremamente efficiente. Ad esempio, una complessa percezione visiva si verifica in meno di 100ms, cioè, 10 fasi di lavorazione;
- ha consapevolezza della propria intelligenza (per inciso nessuno sa ancora come ciò avvenga).

Tipo elaboratore	*Elementi elaborazione*	*Dimensione Elementi*	*Velocità Elaborazione*	*Stile Calcolo*	*Fault Tolerant*	*Impara*	*Conscio Intelligenza*
Cervello	10^{14} sinapsi	10^{-6} metro	100 Hz	distribuito (parallelo)	sì	sì	sì
CPU (1 core)	10^{8} transistor	10^{-6} metro	10^{9} Hz	centralizzato (seriale)	no	un po'	no

Tabella I
Parellelismo tra cervello e CPU

Come branchia dell'Intelligenza Artificiale, le reti neurali artificiali sono, quindi, il tentativo di portare i computer un po' più vicino alle capacità del cervello imitandone alcuni aspetti di elaborazione delle informazioni, in un modo altamente semplificato.

B. Le reti neurali nel cervello

Il cervello non è una entità omogenea. In una classificazione macroscopica, si distinguono corteccia, mesencefalo, tronco cerebrale e del cervelletto. Ognuna di queste parti può essere gerarchicamente suddivisa in molte altre regioni e all'interno di ciascuna regione in aree, sia in base alla struttura anatomica delle reti neurali sia in base alla funzione svolta da questi ultimi.

[1]Tecnologia di processore con un singolo core.

Il modello generale di *proiezioni* (fasci di connessioni neurali) tra le aree è estremamente complesso e solo parzialmente conosciuto. La zona maggiormente studiata, ed anche la più grande, è il sistema visivo, dove i primi 10 o 11 stadi di trasformazione sono stati identificati. Si distinguono proiezioni *feedforward* che vanno dalla prima fase di trasformazione (stimoli sensoriali) all'ultima trasformazione (visualizzazione dell'immagine), e proiezioni di tipo *feedback*, cioè connessioni che vanno nella direzione opposta. Oltre a questi collegamenti ad ampio raggio, i

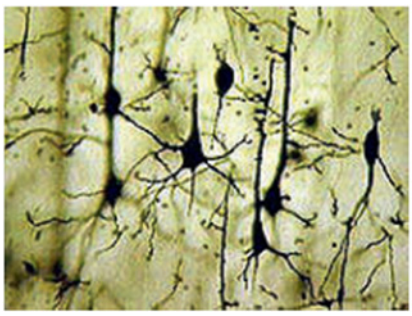

Figura 1. *Rete neurale biologica*

neuroni possono mettersi in contatto con molte migliaia di neuroni vicini. In questo modo si forma una serie di reti locali dense e complesse (Figura 1).

C. Neuroni e sinapsi

L'unità di base di calcolo nel sistema nervoso è la cellula nervosa, o neurone (Figura 2 e Figura 3). Un neurone è costituito dai seguenti elementi:

- *Dendriti* (ingressi)
- *Soma* (corpo della cellula)
- *Assone* (uscite)

Figura 2. *Singolo neurone biologico*

Un neurone riceve l'input da altri neuroni (molte migliaia in genere). Gli ingressi tendono a sommarsi, a dare un

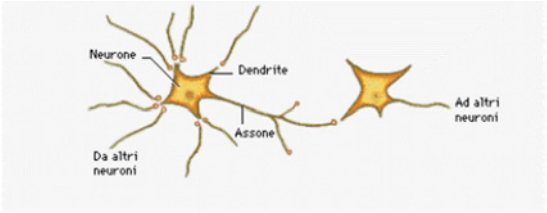

Figura 3. *Struttura di un neurone biologico*

contributo cumulativo al neurone destinatario. Una volta che l'ingresso supera un valore di soglia detto livello critico, si genera un impulso detto *spike*: un impulso elettrico che viaggia dal corpo, giù per l'assone, verso il neurone successivo o a tutti gli altri recettori a cui è connesso. Questo evento è anche chiamato *depolarizzazione*, ed è seguito da un periodo refrattario, durante il quale il neurone non è in grado di essere recettivo.

Le terminazioni degli assoni sono quasi a contatto con i dendriti o con il corpo cellulare del neurone successivo. La trasmissione di un segnale elettrico da un neurone all'altro avviene tramite i neurotrasmettitori, una serie di informazioni di tipo chimico che vengono rilasciati dal neurone trasmettitore e si legano ai recettori nel neurone destinatario. Questo collegamento è chiamato *sinapsi*. La misura in cui il segnale da un neurone passa a quello successivo dipende da molti fattori, come ad esempio la quantità di neurotrasmettitori disponibili, il numero e la disposizione dei recettori, la quantità di neurotrasmettitore riassorbita e molto altro ancora...

D. Apprendimento sinaptico

Il cervello impara, naturalmente. Da quello che conosciamo delle strutture neuronali, un modo che il cervello ha per imparare è di modificare i punti di forza delle connessioni tra neuroni: aggiungere o eliminare le connessioni tra i neuroni stessi. Inoltre si impara sulla base dell'esperienza e, in genere, senza l'aiuto di un insegnante.

L'efficacia di una sinapsi può variare in funzione del risultato di una determinata esperienza, fornendo sia la memoria sia l'apprendimento attraverso il potenziamento a lungo termine (*LTP - Long Term Potentiaton*). Un modo in cui ciò avviene è attraverso il rilascio di altri neurotrasmettitori. Comunque molti altri cambiamenti possono anche essere coinvolti.

Il potenziamento a lungo termine è un aumento duraturo dell'efficacia sinaptica ($>$ 1 ora) come il risultato di una alta frequenza di stimolazione di vari ingressi attraverso quello che è comunemente denominato il *Postulato di Hebbs*:

> *"Quando un assone della cellula A eccita la cellula B ripetutamente o prende parte in modo persistente a questo processo di stimolazione, un processo di crescita, o cambiamento metabolico,*

avviene in una o entrambe le cellule in modo che l'efficienza di A come cellula di trasmissione di B è aumentata."

Alcuni ricercatori scoprirono che nel cervello il processo denominato *LTP* denota le seguenti conseguenze:

- le sinapsi diventano più o meno importanti nel tempo (fenomeno della plasticità)
- LTP è basato sull'esperienza
- LTP è basato solo su fenomeni locali di scambio delle informazioni (Postulato di Hebb)

III. Modelli di Neuroni Artificiali

Come precedentemente detto si è tentato di costruire modelli di elaborazione dati per i computer basati sui neuroni, al fine di eseguire simulazioni molto dettagliate dei "circuiti" del cervello. Da un punto di vista tecnico, da ingegneri o specialisti dell'informatica, si ha comunque più l'interesse alle proprietà generali delle reti neurali artificiali, indipendentemente da come esse siano realmente "realizzate" nel cervello. Questo significa che si può usare molto più semplicemente, una forma astratta dei "neuroni", che (si spera) catturi l'essenza del calcolo neurale, anche se lascia fuori gran parte dei dettagli su come effettivamente lavorino i neuroni biologici.

Storicamente come primo approccio all'implementazione del modello a neuroni si è scelto di realizzare hardware *ad-hoc* con circuiti elettronici di tipo neuronico, spesso integrati su chip VLSI. Si ricordi però che i computer possono essere molto più veloci dei cervelli umani, si possono quindi gestire grandi reti (con un modello di tipo neuronale), abbastanza semplicemente con software di simulazione e in tempi ragionevoli. Questo ultimo aspetto software offre alcuni vantaggi evidenti rispetto a dover utilizzare particolari "computer con hardware neuronali".

A. Vantaggi e svantaggi di un modello a rete neurale

Le reti neurali artificiali hanno i seguenti punti di forza:

:-) le reti neurali lavorano in parallelo;
:-) l'elaborazione dei sistemi nervosi è distribuita su molti elementi, quindi ci sono molti elementi, neuroni, che lavorano alla stessa operazione;
:-) si accede al dato non attraverso i mezzi "tradizionali", cioè attraverso l'indirizzo di memoria a cui far riferimento, ma attraverso le informazioni, anche parziali, riferite al "contenuto";
:-) si impara attraverso un apprendimento *on line*, cioè in funzione dell'esperienza, con o senza l'aiuto di un istruttore esterno.

I difetti principali che si possono imputare alle reti neurali artificiali, e sono le motivazioni principali per cui i *puristi* della matematica li trovano poco attraenti, sono i seguenti:

:-(effetto *black-box*: i modelli prodotti dalle reti neurali, anche se molto efficienti, non sono spiegabili in linguaggio simbolico umano per cui i risultati vanno accettati "così come sono";
:-(la realizzazione di una rete neurale dipende molto dall'esperienza del creatore e dalla sua sensibilità e capacità di adottare e capire la rete in funzione del set di dati.

B. Un semplice neurone artificiale

L'elemento di base di calcolo (modello neuronale) è spesso chiamato nodo, unità, o neurone. Esso riceve un input da alcune altre unità, o anche da una fonte esterna. Ad ogni ingresso è associato un peso w, che può essere modificato nella fase di *apprendimento sinaptico* (Figura 4). L'unità calcola una funzione f della somma ponderata dei suoi ingressi:

$$y_i = f\Big(\sum_j w_{ij} y_j - \theta_i\Big) \tag{1}$$

l'uscita, a sua volta, può servire da input per le altre unità.

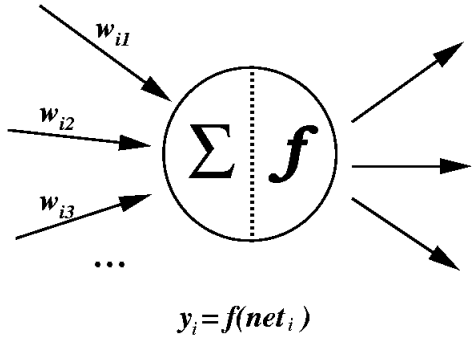

Figura 4. *Unità elementare: neurone artificiale*

- θ_i è detta soglia di eccitazione del neurone. Questo valore di soglia può essere eliminato aggiungendo un ingresso fittizio $y_k = 1$ (di valore sempre unitario) chiamato *BIAS* e con il valore relativo alla connessione di peso pari a $w_{ik} = -\theta_i$ quindi la formula 1 diventa:

$$y_i = f\Big(\sum_j w_{ij} y_j\Big) \tag{2}$$

- La somma ponderata $\sum_j w_{ij} y_j$ è chiamata input di rete per l'unità i, spesso scritto net_i.
- Si noti che w_{ij} si riferisce al peso da unità j per unità i (non viceversa).
- La funzione f è la funzione di attivazione per l'unità. Nel caso più semplice, f è la funzione di identità e l'unità di output è solo il suo ingresso di rete. Questo neurone così strutturato prende il nome di unità lineare.

IV. Regressione Lineare

A. Creazione di un modello di dati

Si consideri i dati rilevati del rapporto tra le coppie (x_i, y_i) ovvero delle informazioni sul peso di un'automobile (asse delle ascisse) e il relativo consumo di carburante (asse delle ordinate) per ognuno dei 74 campioni di riferimento in un anno solare. Queste coppie sono rappresentate graficamente,

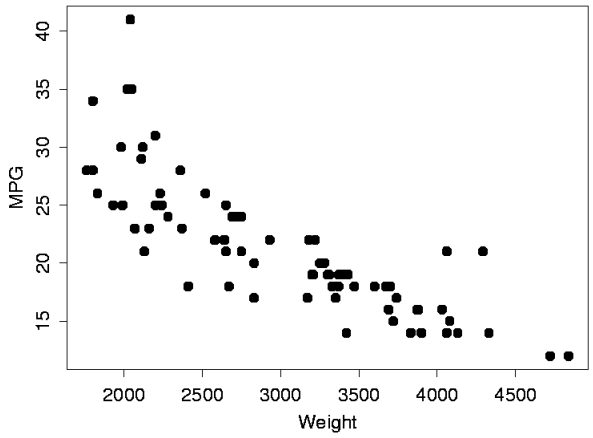

Figura 5. *Diagramma a dispersione per il set di dati (esempio autovetture)*

Figura 5, su un piano cartesiano mediante punti detto *diagramma a dispersione*. Chiaramente il peso e il consumo di carburante sono collegati, in modo che, in generale, le auto più pesanti usano più carburante.

Ora si suppone che sia dato il peso di una 75-esima vettura e si voglia prevedere la quantità di combustibile che verrà utilizzato sulla base dei dati di cui si dispone per le precedenti 74 automobili. A queste domande si può rispondere utilizzando un modello - un semplice modello matematico - dei dati. Il modello più semplice in questo caso è della forma:

$$y = w_1 x + w_0 \tag{3}$$

Si tratta di un modello lineare: su un piano cartesiano XY, l'equazione 3 descrive una linea retta con pendenza w_1 e con l'intercetta w_0, come la figura 6 mostra[2].

Come scegliamo i due parametri w_0 e w_1 per la formulazione del modello? Chiaramente, qualsiasi linea retta tracciata in qualche modo attraverso i dati potrebbe essere usata come un indicatore, ma per alcune linee ci sarà un livello di previsione migliore che per le altre. La linea in figura 6 non è certamente un buon modello: per la maggior parte delle automobili predirà un consumo di carburante troppo alto per un determinato peso.

[2]Si noti che si è riscalato le unità di misura negli assi. Questa riscrittura, comunque, non cambia il problema.

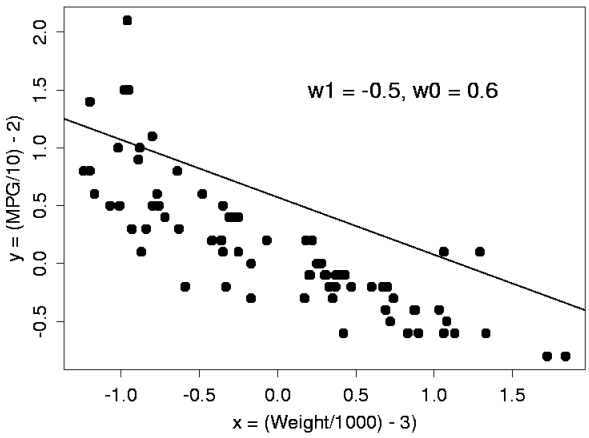

Figura 6. *Un esempio, migliorabile, di regressione lineare*

B. La Funzione Errore

Al fine di precisare ciò che si intende per "buon indice di previsione", si definisce *errore*, o indice di perdita, la funzione E sui parametri del modello. Una scelta per la definizione dell'errore E è data dalla somma dei quadrati delle differenze:

$$E = \frac{1}{2} \sum_i \Big(t_i - y_i \Big)^2 \tag{4}$$

In altre parole, è la somma per tutti i punti i nel nostro set di dati del quadrato della differenza tra il valore effettivo t_i (consumo di carburante) e il valore del modello di previsione y_i, calcolato con valore di ingresso x_i (peso della vettura) secondo la formula 3. Per un modello lineare, l'errore dell'indice così definito è una funzione quadratica dei parametri del modello. La Figura 7 mostra il valore E per un intervallo di valori di w_0 e w_1. La Figura 8 mostra le stesse relazioni in un ambiente bidimensionale.

C. Minimizzare l'Errore

La funzione di perdita E fornisce una misura oggettiva dell'errore nella previsione per una precisa scelta dei parametri del modello. Si può quindi riformulare l'obiettivo di individuare il migliore modello (lineare) come trovare i valori dei parametri del modello che minimizzano E. Per i modelli lineari, la regressione lineare rappresenta un metodo diretto per calcolare questi parametri nel modello ottimale. Tuttavia, questo approccio analitico non può essere generalizzato nel caso di modelli non lineari. Anche se la soluzione non può essere calcolata in modo esplicito in questo caso, il problema può essere risolto con una tecnica numerico iterativa chiamata *gradient descent* (discesa del gradiente o anche nota come regola di `Widrow-Hoff`). Questa tecnica funziona secondo il seguente algoritmo:

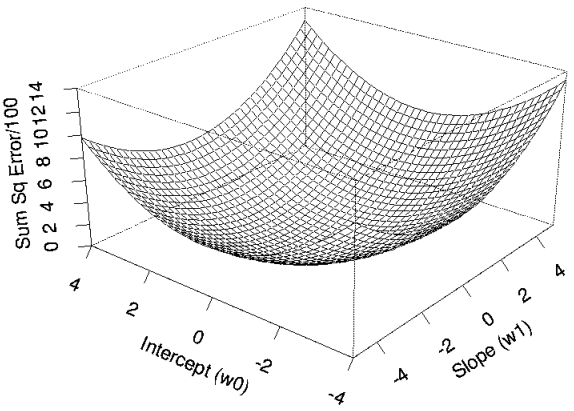

Figura 7. *Funzione di perdita E rispetto a w_0 e w_1 (tridimensionale)*

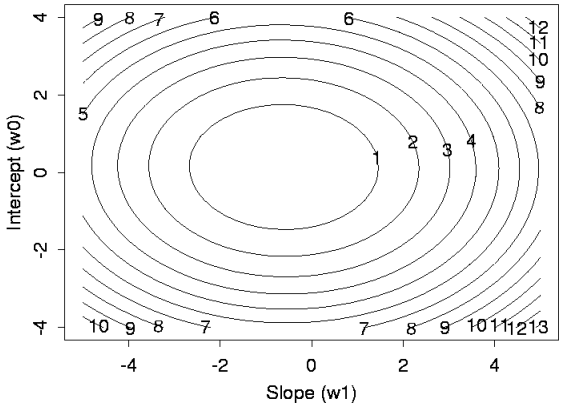

Figura 8. *Funzione di perdita E rispetto a w_0 e w_1 (bidimensionale)*

1) si scelgano, in modo casuale, alcuni valori iniziali per i parametri del modello.
2) si calcoli il gradiente G[3] della funzione di errore rispetto a ciascun parametro del modello.
3) si cambino i parametri del modello in modo che si muovano in direzione della diminuzione dell'errore, cioè, in direzione di $-G$.
4) si ripetino i passaggi 2 e 3 fino a quando il valore di G si avvicina a zero.

[3]Che cosa è il gradiente? In matematica il *gradiente* di un campo scalare è una funzione a valori reali di più variabili reali, quindi definita in una regione di uno spazio a due, tre o più dimensioni. Il gradiente di una funzione è definito come il vettore che ha per componenti cartesiane le derivate parziali della funzione. Il gradiente rappresenta quindi la direzione di massimo incremento di una funzione di n variabili $f(x_1, x_2, \ldots, x_n)$. Il gradiente è quindi una grandezza vettoriale che indica come una grandezza fisica vari in funzione dei suoi diversi parametri.

Come funziona questo algoritmo? Il gradiente di E fornisce la direzione in cui la funzione dell'errore con i valori attuali di w ha la pendenza più ripida. Quindi per diminuire E, si devono effettuare dei piccoli passi nella direzione opposta, $-G$ (Figura 9). Ripetendo questa operazione più

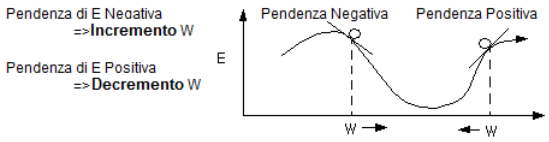

Figura 9. *Incremento o decremento dei valori di w_0 e w_1*

volte, in modo iterativo, ci si muove in "discesa" verso il minimo di E, cioè fino a raggiungere un minimo in cui $G = 0$, in modo tale che nessun ulteriore progresso sia possibile (Figura 10). La Figura 11 mostra il miglior modello

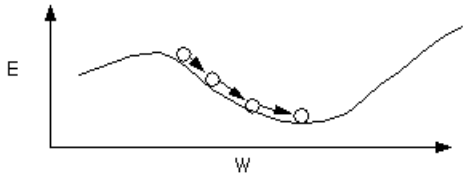

Figura 10. *Ricerca del minimo per ridurre l'errore E*

lineare per il set di dati del rapporto peso/consumo delle autovetture trovato con questa procedura.

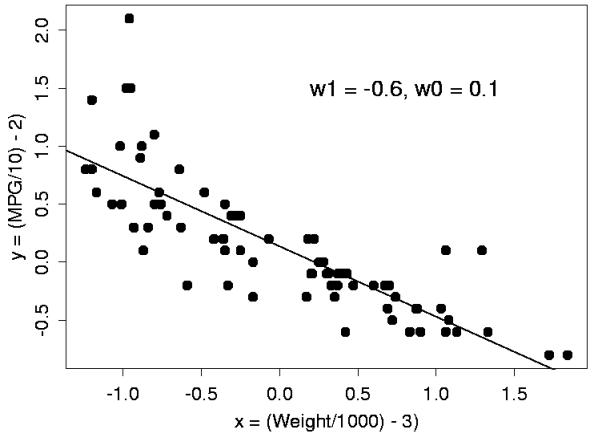

Figura 11. *Miglior modello lineare*

D. Implementazione con rete neurale

Il modello lineare con l'equazione 3 si può implementare con la semplice rete neurale di Figura 12. Si tratta dell'unità

di tipo bias, un ingresso e una unità di uscita lineare. L'unità di input rende l'ingresso x (il peso di un auto) a disposizione della rete, mentre l'unità di bias è sempre un valore costante uguale a 1 (confronta equazione 1). L'unità di output calcola la somma nel seguente modo:

$$y_2 = y_1 w_{21} + 1.0 w_{20} \tag{5}$$

È facile vedere che questa equazione è equivalente all'equazione 2, con w_{21} valore della pendenza della linea retta, e w_{20} la sua intercetta con l'asse y.

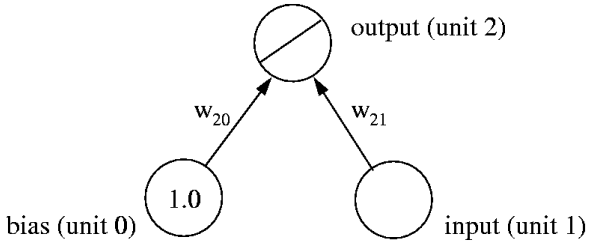

Figura 12. *Rete neurale lineare*

V. Reti Neurali Lineari

A. Regressione multipla

L'esempio precedente mostra come si può scoprire una funzione lineare ottimale per predire una variabile (consumo di carburante) in funzione di un'altra variabile (peso delle autovetture). Si supponga ora che ci sia dato anche una o più variabili aggiuntive che potrebbero essere utili come predittori. Il nostro modello di rete neurale può essere facilmente esteso a questo caso con l'aggiunta di più unità di ingresso (Figura 13).

Allo stesso modo, se può essere utile prevedere più di una variabile dal set di dati che si stanno considerando si aggiunge più di una unità di uscita (Figura 14). La funzione di perdita E per una rete con più uscite si ottiene semplicemente sommando tutti i valori di perdita per ogni unità d'uscita. La rete ora ha una tipica struttura a strati: uno strato per le unità di ingresso (e il bias), collegato da un insieme di pesi ad uno strato di unità di uscita.

B. Calcolo del gradiente

Al fine di formare delle reti neurali come quelle indicate nella sezione precedente, si deve poter calcolare il gradiente G della funzione di perdita E in relazione ad ogni peso w_{ij} della rete. Esso ci dice come una piccola variazione influirà sul peso complessivo dell'errore E. Si suddivide la funzione di perdita in termini distinti per ogni punto p nei dati di addestramento:

$$E = \sum_p E^p, \qquad E^p = \frac{1}{2} \sum_o \left(t_o^p - y_o^p \right)^2 \tag{6}$$

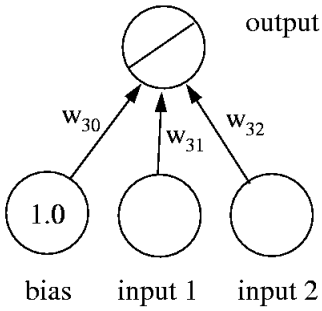

Figura 13. *Rete neurale artificiale lineare - Multipli Ingressi Singola Uscita*

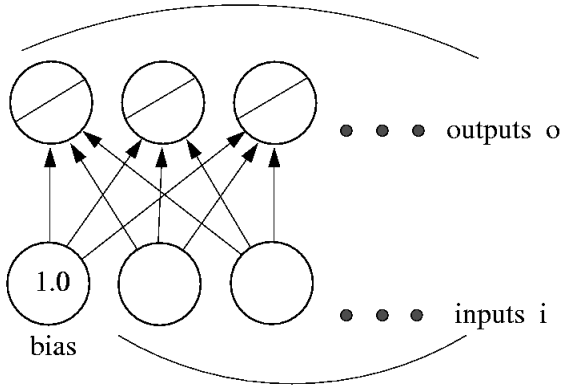

Figura 14. *Rete neurale artificiale lineare - Multipli Ingressi Multiple Uscite*

dove O è lo spazio delle unità di uscita della rete[4]. Dal momento che la derivata e la somma sono funzioni lineari si possono intercambiare di posizione. Si può, quindi, scrivere il gradiente diviso in componenti separate per ciascun punto di formazione:

$$G = \frac{\partial E}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_p E^p = \sum_p \frac{\partial E^p}{\partial w_{ij}} \tag{7}$$

Per una semplicità di notazione si descrive il calcolo del gradiente di un unico punto, omettendo, quindi, l'apice p. Utilizzando la regola della scomposizione *chain rule* si può riscrivere il gradiente:

$$\frac{\partial E}{\partial w_{oi}} = \frac{\partial E}{\partial y_o} \frac{\partial y_o}{\partial w_{oj}} \tag{8}$$

Il primo fattore può essere ottenuto differenziando l'equazione 6:

$$\frac{\partial E}{\partial y_o} = -(t_o - y_o) \tag{9}$$

[4]La notazione apice p indica il punto di calcolo della funzione di perdita e non un elevamento a potenza.

utilizzando $y_o = \sum_j w_{oj} y_j$, il secondo fattore della 8 diventa

$$\frac{\partial y_o}{\partial w_{oi}} = \frac{\partial}{\partial w_{oi}} \sum_j w_{oj} y_j = y_i \tag{10}$$

Ricomponendo l'equazione 8 in funzione dei termini descritti dalle equazioni 9 e 10 si otterrà:

$$\frac{\partial E}{\partial w_{oi}} = -(t_o - y_o) y_i \tag{11}$$

Per trovare il gradiente G per l'intero set di dati, si somma per ogni peso il contributo dato dalla equazione 11 su tutti i punti dei dati.

Si può sottrarre una piccola parte μ (chiamata *tasso di apprendimento*) dai pesi del gradiente G per eseguire l'algoritmo di discesa del gradiente.

C. L'algoritmo di discesa del gradiente

1) Si inizializzano tutti i pesi con dei piccoli valori casuali.
2) Si ripetono le seguenti operazioni per ogni punto p del set dei dati
   a) Per ogni peso w_{ij} inizializza $\Delta w_{ij} = 0$
   b) Per ogni punto di dati $(x,t)^p$
      i) imposta l'unità di ingresso a x
      ii) si calcola il valore dell'unità di uscita
      iii) per ogni peso w_{ij} si imposta $\Delta w_{ij} = \Delta w_{ij} + (t_i - y_i) y_j$
   c) per ogni peso w_{ij} si imposta $w_{ij} = w_{ij} + \mu \Delta w_{ij}$

L'algoritmo termina una volta che si è sufficientemente vicini al minimizzare la funzione di perdita, di errore, cioè dove $G = 0$. Si dice allora che l'algoritmo *converge*. In sintesi:

	Caso generale	*Rete lineare*
Formazione dei dati	(x,t)	(x,t)
Parametro del modello	w	w
Modello	$y = g(w,x)$	$y_o = \sum_j w_{oj} y_j$
Funzione Errore	$E(y,t)$	$E = \sum_p E^p$, $E^p = \frac{1}{2} \sum_o (t_o^p - y_o^p)^2$
Gradiente rispetto w_{ij}	$\frac{\partial E}{\partial w_{ij}}$	$-(t_i - y_i) y_j$
Regola aggiornamento dei Pesi	$\Delta w_{ij} = -\mu \frac{\partial E}{\partial w_{ij}}$	$\Delta w_{oi} = \mu (t_o - y_o) y_i$

Tabella II
Parallelo tra rete neurale lineare e caso generale

D. Tasso di apprendimento

Una considerazione importante da fare sul tasso di apprendimento μ. Esso determina quanto cambino i pesi w ad ogni passo dell'algoritmo. Se il valore di μ è troppo piccolo, l'algoritmo impiegherà molto tempo per convergere (Figura 15). Al contrario, se il valore di μ è troppo grande, si può finire per creare un effetto di rimbalzo e la funzione di perdita va fuori controllo: l'algoritmo, allora, *diverge* (Figura 16). Questo di solito conclude l'applicazione del calcolo dell'errore con un overflow dell'elaboratore.

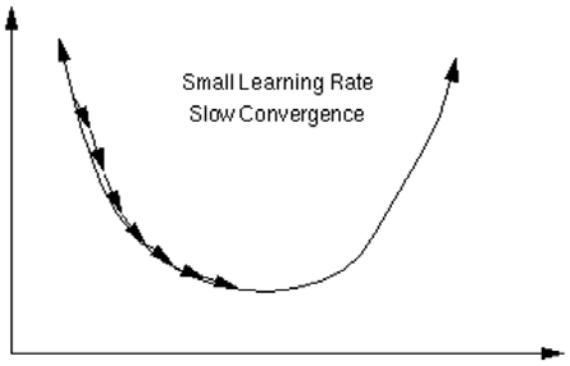

Figura 15. *Convergenza nella minimizzazione dell'Errore*

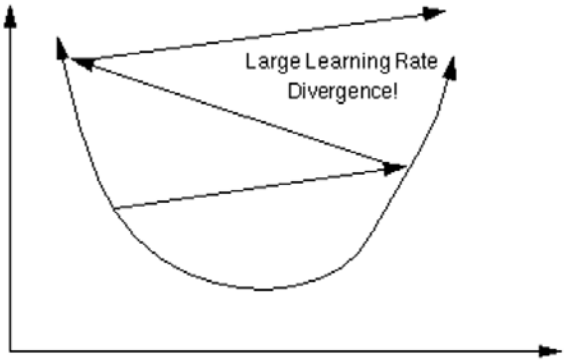

Figura 16. *Divergenza dalla minimizzazione dell'Errore*

E. *Apprendimento batch* versus *apprendimento on-line*

Negli algoritmi citati nelle sezioni precedenti si sono accumulati i contributi di pendenza per tutti i punti dei dati nel set di training prima di aggiornare i pesi. Questo metodo è spesso indicato come *apprendimento batch*. Un approccio alternativo è l'*apprendimento on-line*, in cui si aggiornano i pesi subito dopo aver preso in considerazione ogni punto del set dei dati. Poiché il gradiente di un singolo punto può essere considerato una buona approssimazione (anche se con un certo grado di rumore) del gradiente globale G (Figura 17), questo metodo è chiamato anche discesa del gradiente *stocastico*. L'apprendimento on-line offre un certo numero di vantaggi:

- spesso è molto più veloce, soprattutto quando il training set è ridondante (contiene molti punti di dati simili);
- può essere utilizzato quando non vi è un training set prefissato;

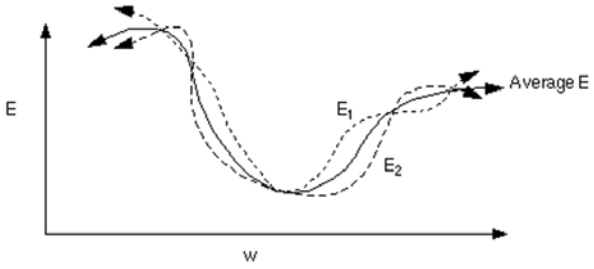

Figura 17. *Valore medio dell'Errore*

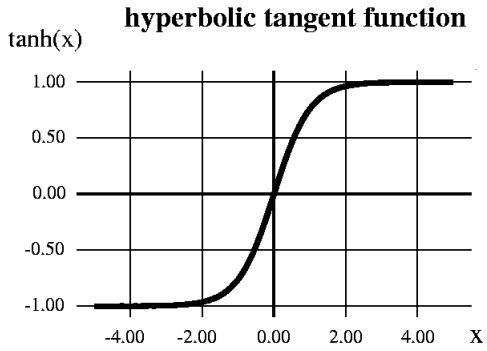

Figura 18. *Funzione di attivazione a tangente iperbolica*

- è migliore nel tracking di ambienti non stazionari (cioè dove il modello migliore cambia gradualmente nel tempo);
- il rumore del gradiente può aiutare a uscire dai minimi locali (che sono un problema per l'algoritmo della discesa del gradiente nei modelli non lineari).

Questi vantaggi sono, comunque, pagati a caro prezzo: potenti tecniche di ottimizzazione (come ad esempio: support vector machines, metodi bayesiani, ecc ...), di cui non si parlerà in questo tutorial, sono metodi di apprendimento batch che non possono essere utilizzati in modo on-line[5]. Un compromesso tra i due metodi di apprendimento è l'uso di algoritmi di apprendimento a *mini-batch*: i pesi sono aggiornati dopo ogni n punti di dati, dove n è maggiore di 1, ma più piccolo della dimensione massima del set dei dati. Al fine di mantenere le cose semplici, in questo tutorial, ci si concentra molto sull'apprendimento di tipo on-line, dove la tecnica della discesa del gradiente è tra le migliori tecniche disponibili.

VI. Reti multi-layer

A. Un problema non lineare

Si consideri ancora il miglior fit lineare che si è trovato per il set dei dati delle autovetture. I dati, come ovvio, sono punti che non sono distribuiti uniformemente su tutta la retta di interpolazione (o di regressione): per le autovetture con pesi bassi, si nota che il modello prevede che vi sia più consumo rispetto a quello del dato effettivo. In realtà, più che una retta, una semplice curva potrebbe adattarsi meglio a questi dati. Si può costruire una rete neurale in grado di effettuare delle interpolazioni non di tipo lineare semplicemente introducendo un nuovo livello (e quindi dei nuovi nodi) con una adeguata funzione di attivazione sui nuovi neuroni. Una funzione di attivazione utile per questo scopo è la forma di S della *tangente iperbolica* $tanh$ (Figura 18).

[5]Naturalmente questo significa anche che se si volesse effettuare l'apprendimento batch in modo efficiente bisognerebbe avere accesso a più metodi ed imparare molte tecniche differenti il che rende la questione piuttosto complicata.

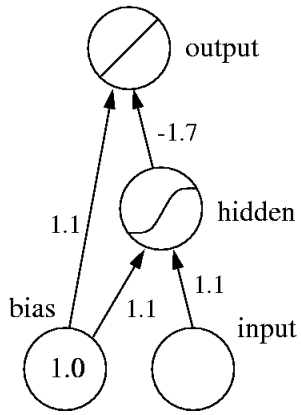

Figura 19. *Esempio con un livello nascosto*

La Figura 19 mostra la nuova rete: vi è rappresentata la rete neurale con l'aggiunta di un nodo e relativa funzione di attivazione a tangente iperbolica. Questo nodo è tra il livello di ingresso e quello di uscita. Poiché tale nodo è "nascosto" all'interno della rete, è comunemente chiamato unità nascosta [4]. Si noti che l'unità nascosta riceve anche un peso dall'unità bias. In generale, tutte le unità di rete neurale devono avere la connessione al peso bias. Per semplicità, l'unità di bias ed i suoi relativi pesi sono di solito omessi dai diagrammi delle reti neurali, a meno che, però, non sia diversamente specificato, si deve sempre pensare che essi ci siano.

Quando questa rete è addestrata con la tecnica della discesa del gradiente sul set dei dati delle automobili, si noti come la funzione di attivazione a tangente iperbolica faccia modificare l'andamento previsionale della rete neurale (Figura 20). Ciascuno dei quattro pesi nella rete svolge un ruolo particolare in questo processo: i due pesi dell'unità bias ruotano la funzione tangente iperbolica nel piano cartesiano secondo le direzioni x e y, rispettivamente, mentre gli altri due pesi effettuino una operazione di ridimensionamento, di

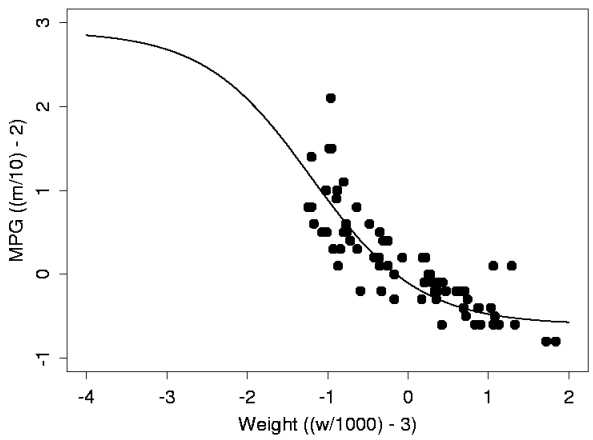

Figura 20. *Adattamento non lineare*

scalatura, lungo queste due direzioni. La Figura 19 riporta i valori di peso che ha prodotto la soluzione di Figura 20.

B. Livelli nascosti

Si può sostenere che nell'esempio precedente si è truffato, poiché si è selezionato una funzione di attivazione dell'unità nascosta che possa adattarsi bene al set di dati. Cosa si dovrebbe fare se il set dei dati[6] si presenta come nel diagramma a dispersione della Figura 21?

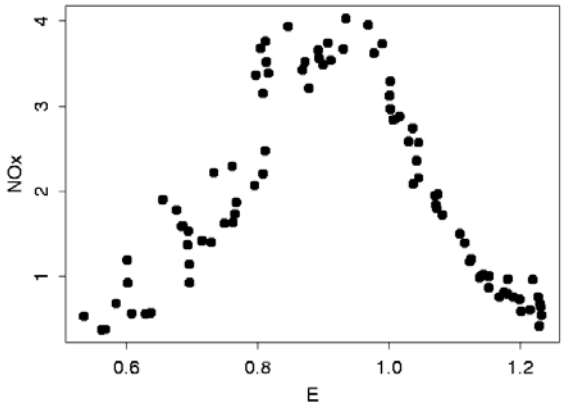

Figura 21. *Diagramma a dispersione del set dei dati (esempio etanolo)*

Ovviamente la funzione tangente iperbolica, non può andare bene su questo set dei dati. Si potrebbe realizzare una speciale funzione di attivazione per ogni insieme di dati che si incontra, ma ciò sarebbe la sconfitta dell'obbiettivo proposto: realizzare un *meccanismo* che in automatico *impari* a modellare i dati. Si vorrebbe avere una funzione generale che permetta di prevedere ed adattare una *qualsiasi* determinata serie di dati.

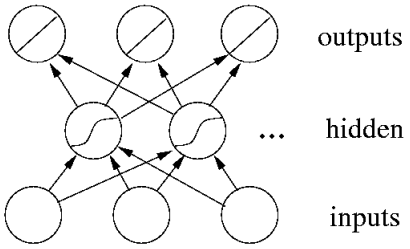

Figura 22. *Rete neurale multi layer*

Fortunatamente esiste una soluzione molto semplice: aggiungere una unità nascosta in più. Infatti, una rete con solo due unità nascoste che utilizzano la funzione tangente iperbolica (Figura 22) potrebbe andare bene per il set dei dati presentato nella Figura 21. L'adattamento della curva al set dei dati potrebbe essere ulteriormente migliorato con l'aggiunta di un'ulteriore unità al livello nascosto. Si noti, tuttavia, che, avendo un livello nascosto troppo grande o più di un livello nascosto, le prestazioni totali della rete possono degradare(si veda la Sezione VIII). Come principio generale, non si dovrebbe eccedere nell'uso di unità nascoste per risolvere il problema dato. Un metodo per evitare questo rischio è di avviare la formazione di una rete neurale molto piccola. Si applichi l'algoritmo di discesa del gradiente: se non si riesce a trovare una soluzione soddisfacente, si fa crescere la rete con l'aggiunta di una ulteriore unità nascosta, fino a quando non si determina la struttura idonea della rete neurale per soddisfare i requisiti del set dei dati.

I risultati teorici indicano che date molte unità nascoste, una rete come quella in Figura 22 potrebbe approssimare qualsiasi funzione ragionevole in qualsiasi grado di accuratezza richiesto. In altre parole, ogni funzione può essere espressa come una combinazione lineare di funzioni $tanh$: $tanh$ è una *funzione di base universale*. Molte funzioni formano una base universale: le due *classi* di funzioni di attivazione comunemente utilizzate nelle reti neurali sono la *sigmoide*, alla quale appartiene la tangente iperbolica, e la funzioni base *radiale*.

VII. Error Backpropagation

Si è già notato che per formare reti lineari è possibile ricorrere l'algoritmo della discesa del gradiente. Nel tentativo di fare lo stesso per le reti *multi-layer*, però, ci si imbatte in una ulteriore difficoltà: non si hanno a disposizione tutti i valori di "riferimento" (uscite desiderate) per le unità nascoste. Questo sembra essere un problema

[6]In questo ulteriore esempio si esamina la concentrazione di NO, ossido di azoto, e di NO_2, diossido di azoto o ipoazotide, in funzione della quantità di etanolo.

insormontabile: in che modo si potrebbe dire alle unità nascoste come aggiornare i pesi delle loro connessioni? La questione irrisolta è stata, in effetti, il motivo per cui le reti neurali sono cadute in disgrazia nell'ambito scientifico dopo un periodo iniziale di alta popolarità che va dalla fine degli anni '40 alla fine anni '60[7]. Ci vollero circa 20 anni prima che l'algoritmo di errore *backpropagation*[8] rendesse giustizia alle reti neurali: ottenendo un popolare metodo per addestrare le unità nascoste portando, quindi, ad una nuova ondata di ricerca nel campo delle reti neurali e delle sue applicazioni.

In linea di principio il backpropagation fornisce un metodo per formare le reti neurali con qualsiasi numero di unità nascoste disposte in un qualsiasi numero di strati. La rete può non essere organizzata in strati; qualsiasi modello di connettività che permette un ordinamento parziale dei nodi tra ingresso e uscita è permesso. In altre parole, ci deve essere un modo per ordinare le unità in modo tale che tutte le connessioni vadano da unità "precedenti", vicino all'ingresso, alle unità "successive", cioè, più vicine all'uscita. Ciò equivale anche ad affermare che il loro modello di collegamento non deve contenere cicli. Le reti che rispettano questo vincolo sono chiamate *reti feedforward* ed il loro modello di collegamento forma un grafo orientato *aciclico*.

A. Algoritmo

Si vuole costruire una rete feedforward multilayer con l'algoritmo di discesa del gradiente per approssimare una funzione sconosciuta, sulla base di alcuni dati di apprendimento, formato da coppie (x, t). Il vettore x rappresenta un insieme di dati di ingresso alla rete, e il vettore t il corrispondente "obiettivo" (output desiderato). Come si è già visto in precedenza, la pendenza complessiva rispetto all'intero training set è solo la somma delle pendenze per ogni punto campionato, quindi si descriverà come calcolare il gradiente solo per un unico punto campionato del set di dati. La notazione w_{ij}, come già espresso precedentemente, indica il peso dall'unità j all'unità i.

1) *Definizioni*:
   - l'*errore* del segnale per l'unità j: $\delta_j = -\frac{\partial E}{\partial net_j}$
   - il *gradiente*(negativo) per il peso w_{ij}: $\Delta w_{ij} = -\frac{\partial E}{\partial w_{ij}}$
   - l'insieme dei nodi precedenti l'unità i: $A_i = \{j : \exists w_{ij}\}$
   - l'insieme dei nodi successivi l'unità j: $P_j = \{i : \exists w_{ij}\}$
2) *Il gradiente.* Come si è fatto per le reti lineari, si scompone il gradiente in due fattori che secondo la *chain rule*:

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

   Il primo fattore è l'errore di unità i. Il secondo è

$$\frac{\partial net_i}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\sum_{k\in A_i} w_{ik}y_k = y_j$$

   pertanto:

$$\Delta w_{ij} = \delta_i y_j$$

   Per calcolare questo gradiente, si ha quindi bisogno di conoscere il valore "obiettivo" (attività) e l'errore per tutti i nodi della rete.
3) *Inoltro dell'attivazione.* L'attività delle unità di ingresso è determinata dalla rete esterna di ingresso della x. Per tutte le altre unità, l'attività si propaga in avanti:

$$\Delta y_i = f_i\Big(\sum_{j\in A_i} w_{ij}y_j\Big)$$

   Si noti che ora l'attività dell'unità i può essere calcolata, e che quindi l'attività di tutti i suoi nodi precedenti (che formano l'insieme A_i), deve essere nota. Poiché le reti feedforward non contengono cicli, vi è un ordinamento dei nodi tra ingresso e uscita che rispetta questa condizione.
4) *Calcolo dell'errore in uscita*. Supponendo che si stia usando la somma dei quadrati delle differenze:

$$E = \frac{1}{2}\sum_o (t_o - y_o)^2$$

   per l'unità di output o risulta essere semplicemente

$$\delta_o = t_o - y_o$$

5) *Error backpropagation*. Per le unità nascoste, si deve propagare *indietro* l'errore dei nodi di uscita (da cui il nome dell'algoritmo). Anche in questo caso utilizzando il metodo della *chain rule*, si può espandere l'errore di una unità nascosta in termini dei suoi nodi successivi:

$$\delta_i = -\sum_{i\in P_j}\frac{\partial E}{\partial net_i}\frac{\partial net_i}{\partial y_j}\frac{\partial y_j}{\partial net_j}$$

   Dei tre fattori all'interno della somma, il primo è proprio l'errore del nodo i. Il secondo è

[7]Contribuì in modo decisivo alla fine dello sviluppo delle reti neurali soprattutto l'articolo di Marvin Minsky e Seymour Papert, nel 1969, che tracciarono tutti i limiti e i difetti delle reti neurali.

[8]L'algoritmo della "*retropropagazione dell'errore*", l'error backpropagation appunto, venne proposto nel 1985 da Rumelhart, William e Hilton

$$\frac{\partial net_i}{\partial y_j} = \frac{\partial}{\partial y_j} \sum_{k \in A_i} w_{ik} y_k = w_{ij}$$

mentre il terzo è la derivata della funzione di attivazione per il nodo j:

$$\frac{\partial y_j}{\partial net_j} = \frac{\partial f_j(net_j)}{\partial net_j} = f_j'(net_j)$$

Per le unità nascoste h che utilizzano la funzione di attivazione *tangente iperbolica*, si è in grado di utilizzare le particolari identità $tanh(u)' = 1 - tanh(u)^2$, e quindi:

$$f_h'(net_h) = 1 - y_h^2$$

Mettendo insieme tutte le semplificazioni attuate:

$$\delta_j = f_j'(net_j) = \sum_{i \in P_j} \delta_i w_{ij}$$

Si noti che, al fine di calcolare l'errore per l'unità j, si deve prima conoscere l'errore di tutti i suoi nodi successivi (che formano l'insieme P_j). Ancora una volta, fino a quando non ci sono cicli nella rete, vi è un ordinamento dei nodi che va al contrario dall'uscita all'ingresso e che rispetta questa condizione. Per esempio, si può utilizzare semplicemente il contrario dell'ordine in cui l'attività si è propagata in avanti.

B. Un esempio applicativo

Si mostrerà ora un esempio di rete feedforward multi-layer addestrata con l'algoritmo del backpropagation. La Figura

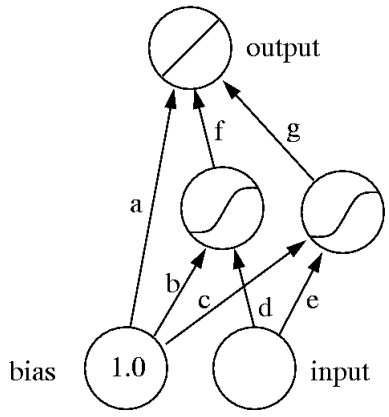

Figura 23. *Rete neurale multi-layer con due nodi nascosti*

21 mostra i dati da modellare mentre la Figura 23 mostra una rete con due unità nascoste, ciascuna con una funzione di attivazione non lineare a tangente iperbolica. L'unità di output è una combinazione lineare delle due funzioni

$$y_o = f\, tanh_1(x) + g\, tanh_2(x) + a \tag{12}$$

Dove

$$tanh_1(x) = tanh(dx + b) \tag{13}$$

e

$$tanh_2(x) = tanh(ex + c) \tag{14}$$

si fissano i pesi con valori iniziali random nell'intervallo $[-1, 1]$. Ogni unità nascosta ha quindi la funzione di attivazione $tanh$ casuale. La Figura 24 mostra le due funzioni di attivazione e l'uscita della rete, che è la loro somma (più una costante negativa)[9]. Ora si addestra la rete (con

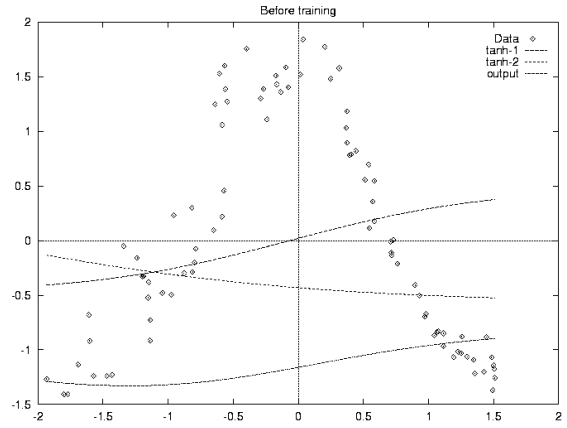

Figura 24. *Esempio con la rete non addestrata*

tasso di apprendimento $\mu = 0.3$), e si aggiornano i pesi dopo ogni punto campionato (apprendimento online). Dopo aver eseguito l'algoritmo di apprendimento sull'intero set di dati per 10 volte (chiamato 10 epoche di formazione), le funzioni calcolate prendono l'aspetto illustrato dalla Figura 25 con la curva centrale che identifica l'uscita. Dopo 12 epoche, si ha la rappresentazione della Figura 26 dove la curva che individua l'output è la curva con la tipica connotazione gaussiana: e dopo 20 epoche si noti come la curva approssimi i valori del set dei dati nella Figura 27. Come le funzioni di attivazione durante il processo di apprendimento sono allungate, scalate e spostate dai pesi che cambiano nella rete, si spera che l'errore del modello si vada minimizzando. Nella Figura 28 si traccia la somma totale dell'errore quadratico su tutti gli 88 campioni dei dati in funzione delle epoche di apprendimento. Quattro percorsi

[9]Se si ha difficoltà ad individuare sul piano cartesiano le funzioni descritte: le prime due curve sono le funzioni $tanh$, mentre, quella più in basso è l'uscita di rete.

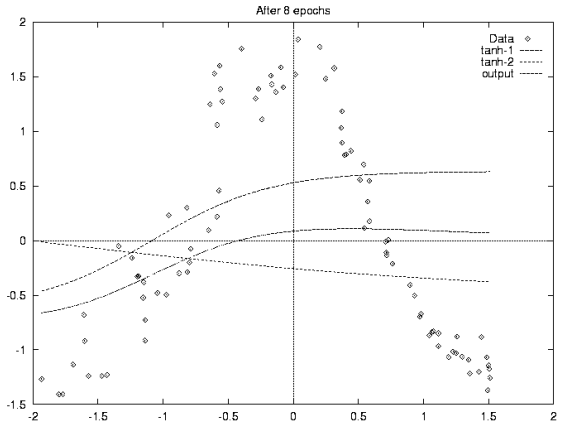

Figura 25. *Esempio con la rete addestrata dopo 8 cicli*

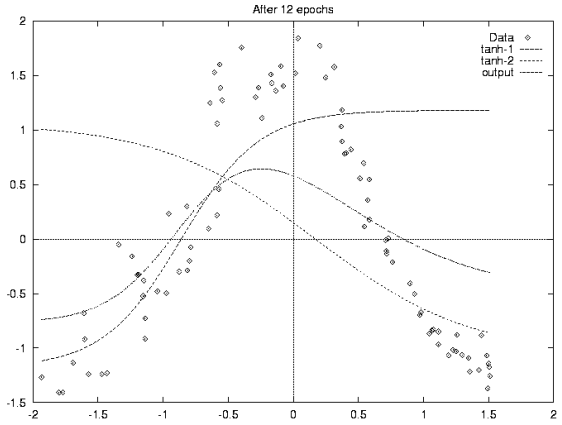

Figura 26. *Esempio con la rete addestrata dopo 12 cicli*

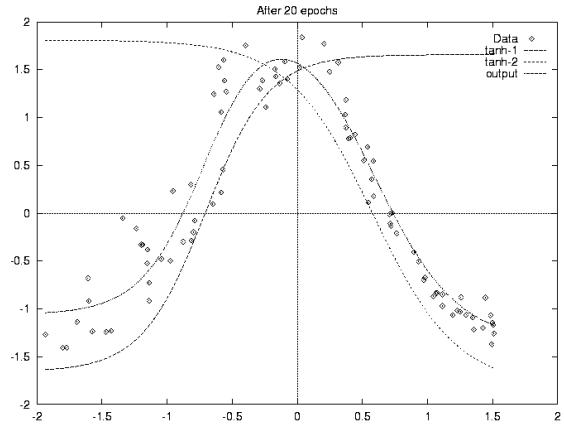

Figura 27. *Esempio con la rete addestrata dopo 20 cicli*

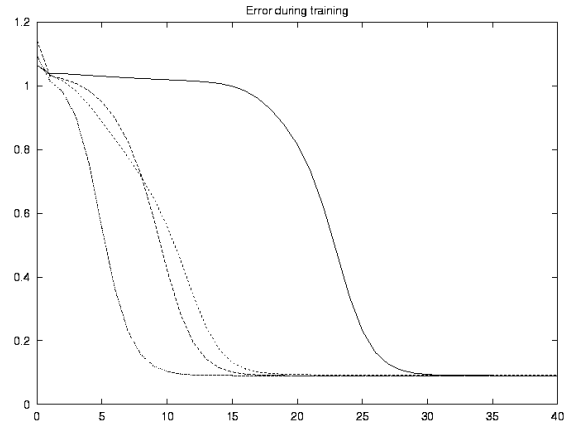

Figura 28. *Visualizzazione dell'Errore durante l'addestramento*

vi sono riportati, con peso di inizializzazione diverso ogni volta.

VIII. Overfitting

Nell'esempio precedente si è usato una rete con due unità nascoste. Solo guardando i dati, è stato possibile immaginare che due funzioni $tanh$ avrebbero fatto un buon lavoro nell'approssimare i dati. In generale, tuttavia, non si può sapere quante unità nascoste, o equivalentemente, quanti pesi si avrà bisogno di produrre per una ragionevole approssimazione dei dati. Inoltre, di solito, si cerca un modello dei dati che fornirà, in media, le migliori previsioni possibili per l'insieme dei dati. Questo obiettivo può entrare in conflitto con il più semplice compito di modellare bene un insieme di dati di apprendimento. In questa sezione si vedranno alcune tecniche che evitano la formazione di un modello troppo preciso (overfitting).

A. *Andamento di tipo* Bias *o* Variance

Si considerino i grafici rappresentati nella Figura 29. I dati (rappresentati dai punti) sono stati tutti generati da una funzione regolare, $h(x)$, con l'aggiunta di un rumore ϵ. Ovviamente si vuole costruire un modello in grado di approssimare la funzione $h(x)$, dato un insieme specifico di dati $y(x)$ generato come:

$$y(x) = h(x) + \epsilon \tag{15}$$

Nel sistema cartesiano di sinistra si cerca di approssimare i punti utilizzando una funzione $g(x)$, con un numero esiguo di parametri: una linea retta. Il modello ha il pregio di

essere molto semplice, ci sono solo due parametri liberi. Tuttavia, non fa un buon lavoro nel raccordare i dati e, quindi, non prevede correttamente eventuali nuovi punti di dati. Si dice che il modello semplice ha un elevato *bias*. Il sistema cartesiano di destra mostra un modello che è

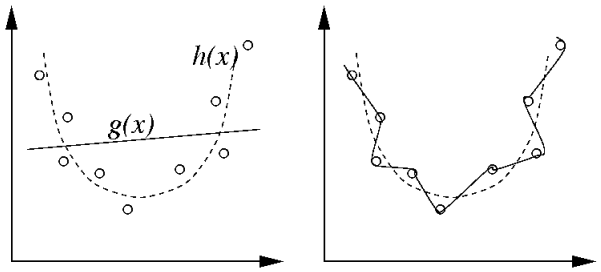

Figura 29. *Adattamento di una curva complessa*

stato costruito usando molti parametri liberi. Esso fa un ottimo lavoro di approssimazione dei punti dati e, quindi, l'errore nei punti dati è vicino allo zero. Tuttavia non ha comunque una buona valenza predittiva di $h(x)$ per i nuovi valori di ingresso x. Il modello ha una elevata *varianza* e non rispecchia la struttura che ci si aspetti in ogni set di dati generati dall'equazione 15.

Chiaramente ciò che si vuole è una via di mezzo: un modello che sia abbastanza potente per rappresentare la struttura dei dati ($h(x)$), ma non così potente da seguire in modo fedele il modello anche sul rumore associato ai campioni dei dati.

L'andamento di tipo a varianza o bias è probabile che diventi un problema se si dispone di un set di dati con pochi punti. Nel caso opposto, in cui si abbia un numero infinito di punti di dati (come in una linea di apprendimento continuo), non c' è pericolo di overfitting dei dati, poiché il rumore associato ad un qualsiasi dato punto svolge un ruolo insignificante nella forma complessiva del modello. Le tecniche seguenti si applicano a situazioni in cui si abbia un set di dati finito e, in genere, si intenda effettuare un addestramento della rete neurale in modalità batch.

B. Prevenire l'overfitting

1) Fasi di arresto: Una delle tecniche più semplici e più diffuse per evitare overfitting è quella di dividere l'intero set dei dati in due *subset*: un insieme per l'addestramento (*training set*) della rete e l'altro per la relativa validazione (*validation set*). Si procede all'addestramento della rete neurale utilizzando solo i dati del *training set*. Ogni tanto, però, ci si ferma nell'addestramento della rete e si testano le sue prestazioni della rete sul set di validazione indipendente. Nessun aggiornamento dei pesi nella rete è effettuato nel corso di questa fase. Poiché i dati di convalida sono indipendenti dai dati di apprendimento, le prestazioni rilevate sono una buona misura della capacità di generalizzazione della rete stessa. Fino a quando la rete impara il legame nella struttura dei dati ($h(x)$ nell'esempio precedente), le prestazioni sul set di validazione miglioreranno con l'addestramento. Una volta però che la rete smette di apprendere le informazioni circa il legame dei campioni e riconosce gli errori introdotti su ogni campione (ϵ dell'equazione 15) le prestazioni sul set di validazione smetteranno di crescere[10] come mostrato in Figura 30. In Figura 30 sono raffigurate l'andamento dell'errore sul set di addestramento e su quello di validazione. Per evitare il fenomeno dell'overfitting, basta interrompere l'addestramento al tempo t, valore in cui le prestazioni della rete risultano ottimali. Un dettaglio degno di nota quando si

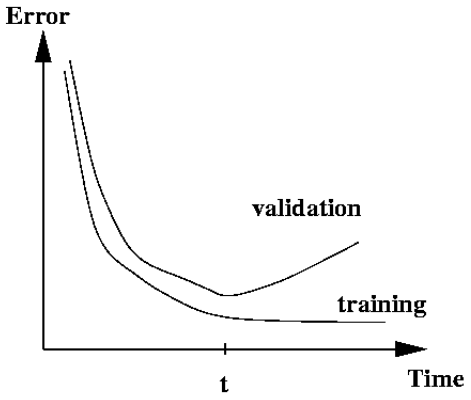

Figura 30. *Errore nella fase di addestramento e di validazione*

utilizza la tecnica dell'arresto precoce: se si vuole testare la rete addestrata su un insieme di dati indipendenti per misurare la sua capacità di generalizzare si ha bisogno di un terzo, indipendente, set di prova (*generalisation set*). Questo perché si utilizza il validation set per decidere quando smettere di addestrare la rete neurale e, quindi, la rete così formata non è più del tutto indipendente dal set di validazione. I requisiti richiesti per un addestramento indipendente, una convalida e un'ultima fase di generalizzazione, indicano che la tecnica dell'arresto precoce può essere utilizzata solo in una situazione ricca di dati.

2) Decadimento dei pesi: La funzione su misura della Figura 29 mostra un alto indice di curvature, mentre la funzione lineare è altamente liscia. Con il termine *regolarizzazione* ci si riferisce ad un insieme di tecniche che contribuisce a garantire che la funzione calcolata dalla rete non sia più curva del necessario. Questo risultato è ottenuto con l'aggiunta di una penalità per la funzione errore:

$$\widetilde{E} = E + \nu\Omega \tag{16}$$

Una forma possibile di regolarizzazione nasce dalla constatazione che una mappatura in overfitting con regioni di

[10]La funzione errore non diminuisce più con il trascorrere del tempo se non addirittura peggiorare

ampia curvatura necessita di grossi pesi nelle connessioni. Si potrebbe quindi penalizzare la scelta di grossi pesi

$$\Omega = \frac{1}{2}\sum_i w_i^2 \tag{17}$$

grazie a questa funzione errore modificata i pesi sono aggiornati

$$\Delta w_{ij} = -\mu\frac{\partial \widetilde{E}}{\partial w_{ij}} = -\mu\frac{\partial E}{\partial w_{ij}} - \mu\nu w_{ij} \tag{18}$$

dove il secondo termine fa sì che il peso si riduca in funzione della propria dimensione. In assenza di ingressi, però, tutti i pesi tendono a diminuire in modo esponenziale, da cui il termine di decadimento dei pesi.

3) Training con il rumore: Un ultimo metodo che spesso può aiutare a ridurre l'importanza delle caratteristiche di rumore specifico associato ad un particolare campione di dati è quello di aggiungere una piccola quantità supplementare di rumore (un piccolo valore casuale con valore medio pari a zero) per ogni ingresso. Ogni volta che un modello di input specifico x è presentato, si aggiunge un valore casuale diverso, si utilizza quindi $x + \epsilon$. In un primo momento questo può sembrare un artificio piuttosto strano: danneggiare deliberatamente i propri dati. Tuttavia si può notare che ora è difficile per la rete approssimare i dati specifici in modo puntuale e specifico. In pratica l'addestramento con rumore aggiunto riduce l'overfitting e quindi migliora la generalizzazione della rete in molte applicazioni.

Se si ha un insieme di addestramento finito, un altro modo di introdurre rumore nel processo di formazione è quello di utilizzare l'addestramento online, cioè, aggiornare i pesi dopo ogni punto del set dei dati e riordinare in modo casuale i campioni alla fine di ogni epoca di addestramento. In questo modo, ogni aggiornamento del peso si basa su una stima disturbata (stocastica) del gradiente reale.

IX. Momentum

A. Minimo Locale

Nell'algoritmo della discesa del gradiente si inizia da un punto qualsiasi della funzione di errore definito sui pesi, e si tenta di passare al minimo globale della funzione. Nella funzione esemplificata dalla Figura 31 la situazione è semplice. Ogni passo nella direzione verso il basso porterà più vicini al minimo globale. Per i problemi reali, tuttavia, le superfici di errore sono tipicamente complesse e possono assomigliare più alla situazione mostrata in Figura 32. Qui ci sono numerosi minimi locali, e la palla è mostrata intrappolata in un minimo locale. Il progresso è possibile solo se da questa posizione si va più in alto prima di ridiscendere verso il minimo globale. Si è già discusso di un modo per sfuggire ad un minimo locale: l'uso dell'addestramento online (confronta la sezione V-E). Il rumore stocastico sulla superficie della funzione di Figura 17 potrebbe far rimbalzare la rete fuori dai minimi locali, purché essi non siano troppo acuti.

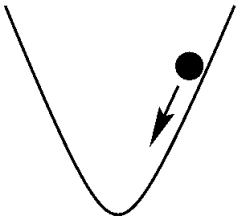

Figura 31. *Minimo globale*

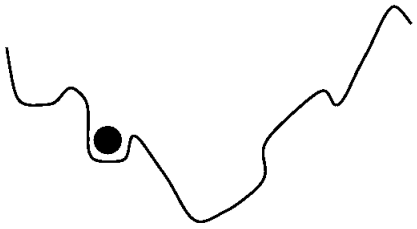

Figura 32. *Minimo globale e minimi locali*

B. Momentum

Un'altra tecnica che può aiutare la rete a non restare imbrigliata nei minimi locali è l'uso di un valore chiamato *momentum*. Questo è probabilmente la più popolare estensione dell'algoritmo backpropagation: è difficile trovare casi in cui non venga utilizzata questa tecnica. Con il momentum m, l'aggiornamento del peso in un dato tempo t diventa

$$\Delta w_{ij}(t) = \mu_i\delta_i y_i + m\Delta w_{ij}(t-1) \tag{19}$$

dove $0 < m < 1$ è un nuovo parametro globale che deve essere determinato per tentativi ed errori. Il momentum aggiunge semplicemente una frazione in funzione di m del peso precedente aggiornato. Quando il gradiente tiene la stessa direzione aumenterà le dimensioni dei passi compiuti verso il raggiungimento del minimo. Anzi, sarà spesso necessario ridurre il tasso globale di apprendimento μ quando si utilizza un momentum (vicino a 1). Se si combina un alto tasso di apprendimento con un alto valore del momentum, si tende al minimo, e forse oltre, con passi da gigante!

Quando la pendenza cambia spesso direzione, il momentum appiana le variazioni. In tali casi la superficie della funzione errore è sostanzialmente differente lungo direzioni diverse, portando alla formazione di valli lunghe e strette. Per la maggior parte dei punti in superficie, il gradiente non punta verso il minimo, e le successive fasi di discesa del gradiente possono oscillare da un lato all'altro, procedendo, quindi, molto lentamente alla ricerca del minimo (Figura 33). La Figura 34 mostra come l'aggiunta del momentum aiuti ad accelerare la convergenza al minimo, mediante

sistemi di smorzamento di queste oscillazioni. Per illustrare

Figura 33. *Ricerca del minimo globale senza momentum*

Figura 34. *Ricerca del minimo globale con momentum*

questo effetto con un esempio pratico si sono addestrate 20 reti su di un semplice problema di codifica con e senza l'utilizzo del momentum. I tempi medi di addestramento, espressi in epoche, sono quelli riportati nella Tabella III.

Momentum	Tempo di addestramento
0	217
0.9	95

Tabella III
Tempo di addestramento di una rete con e senza momentum

X. Classificazione

A. Discriminanti

Le reti neurali possono essere utilizzate anche per classificare i dati [5; 6]. A differenza dei problemi di regressione, dove l'obiettivo è quello di produrre una uscita di un particolare valore per un dato in ingresso, i problemi di classificazione impongono di etichettare ogni dato punto come appartenente ad una delle n classi. Le reti neurali possono essere addestrate per fornire una funzione discriminante che separi le classi. Ad esempio, una rete con una singola uscita lineare può risolvere un problema di due classi: imparare una funzione discriminante che risulti maggiore di zero per una classe e minore di zero per l'altra. La Figura 35 mostra due esempi di un problema di classificazione a due classi: i punti pieni appartenengono ad una classe mentre i punti vuoti appartengono all'altra classe. In ogni caso viene disegnata una linea dove la funzione discriminante che separa le due

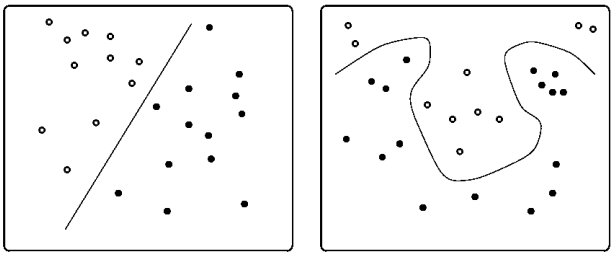

Figura 35. *Esempi di classificazione*

classi è pari a zero. Nel riquadro a sinistra della Figura 35 una linea retta può fungere da discriminante: si è in grado di posizionare la linea in modo tale che tutti i punti pieni si trovino su di un lato, e tutti quelli vuoti giacciano sull'altro. Le classi sono dette linearmente separabili . Tali problemi possono essere appresi da reti neurali senza l'utilizzo di unità nascoste. Nel riquadro destro della Figura 35, una funzione altamente non lineare è necessaria per garantire la separazione delle classi. Questo problema può essere risolto solo da una rete neurale con unità nascoste.

B. Binaria

Per usare una rete neurale per la classificazione, si ha bisogno di costruire un equivalente problema di approssimazione della funzione tramite l'assegnazione di un valore obiettivo per ogni classe. Per un problema *binario* (due classi) si può usare una rete neurale con una singola uscita y, e dei valori obiettivo binario: 1 per una classe, e 0 per l'altra. Si può quindi interpretare l'uscita della rete come una stima della probabilità che un dato campione appartenga alla classe '1'. Per classificare un nuovo campione dopo l'addestramento, si può impiegare il discriminante come il massimo della verosimiglianza $y > 0.5$. Una rete con uscita lineare utilizzata in questo modo, tuttavia, spende un sacco dei suoi sforzi per ottenere i valori obiettivo esattamente giusti per i suoi punti di addestramento, mentre tutta l'attenzione dovrebbe essere rivolta al corretto posizionamento del discriminante. La soluzione è utilizzare una funzione di attivazione in uscita che saturi i due valori obiettivo: tale funzione sarà più vicina al valore obiettivo per ogni ingresso della rete che è sufficientemente ampio ed ha il segno corretto. In particolare, si utilizza la funzione logistica (Figura 36):

$$f(u) = \frac{1}{1+e^{-u}} \qquad f'(u) = f(u)(1-f(u)) \qquad (20)$$

Data l'interpretazione probabilistica, l'uscita della rete, per esempio, di 0.01 per un campione del set dei dati che sia attualmente nella classe '1' è un errore di classificazione molto più grave dell'errore di un campione con uscita 0.1. Purtroppo la funzione di perdita che somma il quadrato delle differenze rende nulla la distinzione tra questi due casi. Una funzione di perdita che sia appropriata per affrontare

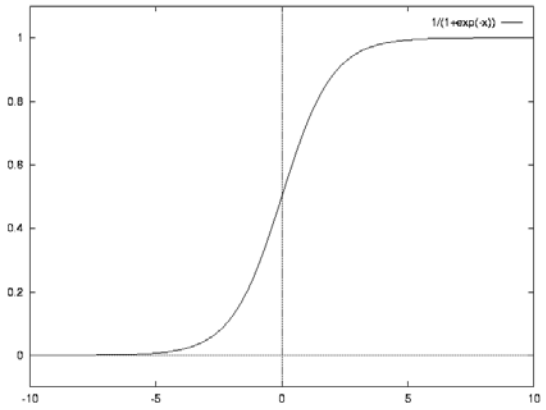

Figura 36. *Funzione di attivazione logistica*

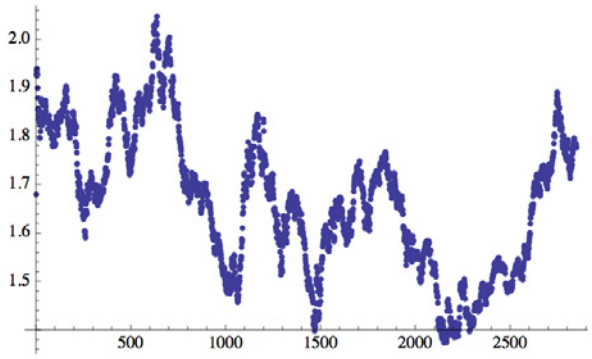

Figura 37. *Tassi di cambio del marco rispetto al dollaro.*

le probabilità è l'errore *cross-entropy*. Per il caso delle due classi è data da

$$E = -t \ln y - (1-t)\ln(1-y) \tag{21}$$

Quando la funzione di attivazione sull'unità di output è di tipo logistica e l'errore è di cross-entropy sono utilizzati insieme all'apprendimento backpropagation, il segnale di errore per l'unità di uscita diventa semplicemente la differenza tra l'obiettivo e l'uscita stessa:

$$\frac{\partial E}{\partial net} = \ldots = y - t \tag{22}$$

In altre parole, applicare l'errore cross-entropy nel presente caso equivale ad omettere il fattore $f'(net)$ per cui l'errore, altrimenti, andrebbe moltiplicato. Ciò non è casuale, ma indica una profonda connessione matematica: la funzione errore cross-entropy e la funzione di attivazione logistica sulle unità di uscita sono la "combinazione giusta" da utilizzare per la probabilità binaria, proprio come le uscite lineari e l'errore della somma dei quadrati sono la combinazione giusta per valori scalari.

XI. Un esempio pratico di rete neurale

Come si è potuto osservare nell'analisi di queste brevi note, le finalità principali delle reti neurali sono sostanzialmente due: previsione e classificazione. Scopo di questa sezione è di analizzare un'applicazione di reti neurali orientata alla previsione dei tassi di cambio. L'enfasi è posta sulle problematiche metodologiche e sulla valutazione dei risultati [7; 8; 9].

In questo esempio, i dati riguardano i tassi di cambio giornalieri della sterlina e del marco confrontati con il dollaro statunitense nel decennio 1987–1997; l'analisi è stata effettuata con l'ambiente di calcolo *Mathematica*.

Sono disponibili circa 2900 giorni di tassi di cambio, con quattro tassi per giorno rappresentanti i valori di apertura, massimo, minimo e di chiusura. Il grafico del valore massimo del marco rispetto al giorno è rappresentato in figura 37.

Si supponga di voler prevedere il valore di apertura utilizzando i tassi di cambio del giorno precedente. Perciò, il valore di apertura è definito come il risultato y del processo e gli altri tre tassi sono gli input u.

Il modello predittivo può essere descritto dalla seguente equazione:

$$\hat{y} = g(\theta, x(t)) \tag{23}$$

Qui $\hat{y}(t)$ è la previsione dell'output $y(t)$, la funzione g è il modello di rete neurale, θ rappresenta i parametri del modello, e $x(t)$ è il regressore del modello, dato dalla seguente equazione:

$$x(t) = [y(t-1)u_1(t-1)u_2(t-1)u_3(t-1)]^T \tag{24}$$

Per un migliore test del predittore il data set è diviso in dati di training (ue e ye) e di validazione (uv e yv). Il secondo data set è utilizzato per validare e confrontare i diversi predittori.

NeuralARX è il modello neurale utilizzato per questa applicazione. Tale rete viene inizializzata e stimata con il comando NeuralARXFit di *Mathematica*. Inizialmente viene stimato un modello di predizione lineare. Per trovare tale predizione, viene scelto FeedForwardNet senza neuroni nascosti. Per ottenere il regressore nella forma 23 occorre scegliere gli indici del regressore come segue: $n_a = 1$, $n_b = 1,1,1$ e $n_k = 1,1,1$.

Stimiamo un modello lineare per la previsione dei tassi. Il modello lineare dell'equazione 23 può essere espresso come segue:

$$\hat{y} = a_1 y(t-1) + b_1 u_1(t-1) + b_2 u_2(t-1) + b_3 u_3(t-1) + b_4 \tag{25}$$

Se si osservano i parametri del modello "addestrato", si vede che b_3 è vicino a t_1 ed il resto dei parametri sono vicini a 0. Ciò significa che il tasso di cambio di apertura è per lo più correlato al tasso di chiusura del giorno precedente. Ciò

sembra una caratteristica abbastanza naturale, ed è quindi possibile avere una qualche fiducia nel modello.

Prima dell'addestramento di un modello non lineare, occorre valutare la previsione single-step sui dati di validazione. Poiché il mercato cambia nel tempo, la previsione viene valutata solo sui 100 giorni successivi ai dati di stima. Il grafico di Figura 38 mostra il tasso previsto insieme al tasso reale. Le due curve sono così vicine che risulta abbastanza difficile discernerle. Viene anche fornito l'errore quadratico medio (RMSE), utilizzabile come misura della qualità della previsione.

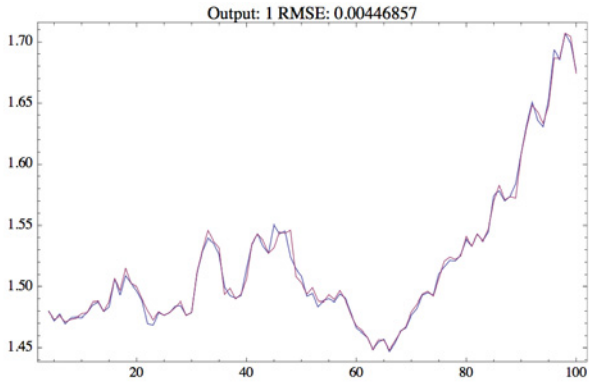

Figura 38. *Valutazione della previsione one-step sui dati di validazione.*

È ora possibile provare modelli non lineari per vedere se sono più efficienti di quello lineare. Utilizziamo una rete *Feed-Forward* con due neuroni nascosti. Il regressore scelto è lo stesso del modello lineare. Un modello lineare è incluso in parallelo con la rete. Poiché i dati di validazione sono inclusi nel successivo addestramento, la stima è ottenuta con la "stopped search".

Addestrando una rete FF con due neuroni sui dati dei tassi di cambio si può apprezzare il miglioramento della rete neurale durante l'addestramento. Esso è molto piccolo poiché l'inizializzazione della rete fa uso dei minimi quadrati per adattare i parametri lineari.

Calcolando la previsione one-step sullo stesso intervallo di dati utilizzato per il modello lineare si vede (figura 40) che l'errore è leggermente inferiore per il modello non lineare rispetto a quello lineare: perciò, la previsione è leggermente migliore. È tipico che il miglioramento di dati economici sia piccolo; altrimenti, sarebbe troppo facile fare denaro.

Consideriamo ora una rete *Radial Basis Function* (RBF) con due neuroni, mantenendo gli stessi argomenti utilizzati per la rete FF. Inizializzando ed addestrando una rete RBF sui dati disponibili otteniamo il grafico di figura 41. Le prestazioni della rete RBF sui dati di validazione peggiorano durante l'addestramento, restituendo la rete RBF inizializzata.

La valutazione della previsione one-step con la rete RBF è rappresenta nel grafico di figura 42. La rete RBF è leggermente migliore della rete lineare, ma non altrettanto buona quanto la rete FF. Se si ripete la valutazione dell'esempio, il risultato potrebbe cambiare leggermente a seguito della casualità nell'inizializzazione.

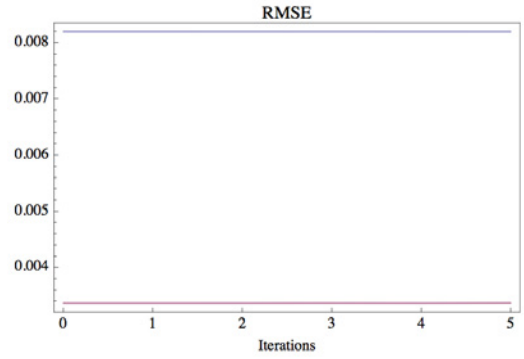

Figura 39. *Training di una rete FF con due neuroni nascosti.*

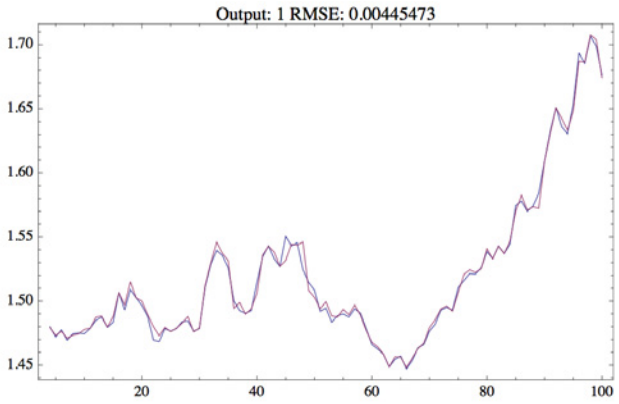

Figura 40. *Previsione one-step.*

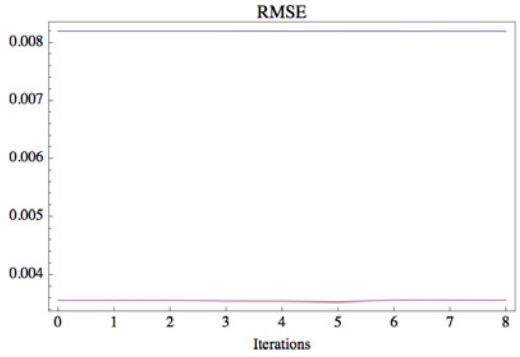

Figura 41. *Inizializzazione e addestramento della rete RBF.*

Si potrebbe ripetere l'esempio modificando diverse opzioni come:

- cambiare il numero di neuroni;

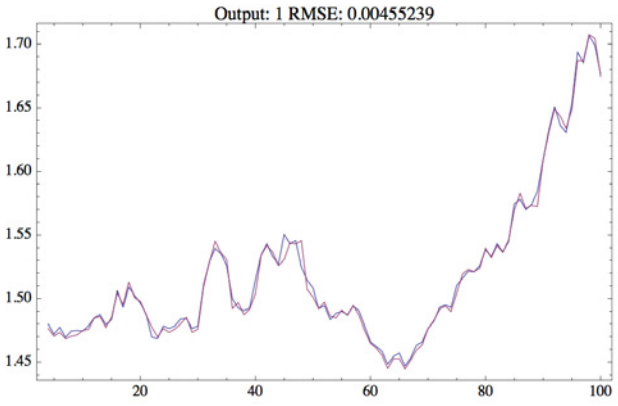

Figura 42. *IValutazione della previsione one-step con la rete RBF.*

- cambiare il regressore per contenere un maggior numero di risultati precedenti;
- escludere alcuni dei segnali di input dal regressore.

È anche possibile cambiare l'intervallo di dati utilizzando quello adoperato nell'addestramento e nella validazione, come pure provare a prevedere la sterlina invece del marco. L'esempio illustra che è possibile prevedere il tasso di cambio di apertura facendo uso dei tassi del giorno precedente. la relazione è naturalmente non lineare, dato che i modelli non lineari basati su reti FF e RBF operano leggermente meglio del modello lineare.

Riferimenti bibliografici

[1] S. Haykin, *Neural Networks: a comprehensive foundation*. Prentice Hall, 2001.

[2] C. Gallo, C. D. Letizia, and G. D. Stasio, "Artificial neural networks in financial modelling," *Quaderno Dipartimento di Scienze Economiche, Matematiche e Statistiche. Univeristà defigli Studi di Foggia*, no. 2, 2006.

[3] C. Gallo, "Reti neurali artificiali: Teoria ed applicazioni," *Quaderno Dipartimento di Scienze Economiche, Matematiche e Statistiche. Univeristà defigli Studi di Foggia*, no. 28, 2007.

[4] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[5] M. Minsky and S. Papert, *Perceptron: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, 1969.

[6] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, pp. 386–408, 1958.

[7] G. P. Zhang and M. Y. H. Patuwo, "Forecasting with artificial neural networks: The state of the art," *Int. Archives of Photogrammetry and Remote Sensing*, vol. 1, no. 14, pp. 35–62, 1998.

[8] E. Azoff, *Neural Network Time Series Forecasting of Financial Markets*. John Wiley & Sons, 1994.

[9] L. Galati and A. Tumietto, *La previsione nei mercati finanziari*. Bancaria Editrice, 1999.

Indice

ELENCO DELLE FIGURE

ELENCO DELLE TABELLE