Università degli Studi di Padova

FACOLTÀ DI INGEGNERIA
Corso di Laurea in Informazione

Tesi di laurea triennale

Reti neurali e loro applicazioni

Relatore:
LUCA
SCHENATO

Laureando:
STEFAN
KOKOROVIC

Anno Accademico 2011–2012

Indice

1 Introduzione **1**

2 Definizioni e metriche di prestazione **3**

2.1 Reti neurali biologiche 3

2.2 Reti neurali artificiali 4

2.3 Addestramento della rete e la funzione di errore 6

3 Modelli di reti neurali **9**

3.1 Multilayer perceptron 9

3.1.1 Architettura del modello MLP 9

3.1.2 Algoritmo di addestramento della rete 12

3.2 Radial basis function network 19

3.2.1 Architettura della rete RBF 19

3.2.2 Stima dei parametri 20

4 Tecniche di ottimizzazione **25**

4.1 Generalizzazione 25

4.2 Data preprocessing 27

5 Esempi **29**

Riferimenti bibliografici **35**

1 Introduzione

Le reti neurali offrono un insieme di strumenti molto potente che permette di risolvere problemi nell'ambito della classificazione, della regressione e del controllo non-lineare. Un semplice esempio è l'analisi di un'immagine per verificare la presenza di determinati oggetti oppure di testi, ed eventualmente riconoscerli.

Oltre ad avere un'elevata velocità di elaborazione le reti neurali hanno la capacità di imparare la soluzione da un determinato insieme di esempi. In molte applicazioni questo permette di raggirare il bisogno di sviluppare un modello dei processi fisici alla base del problema, che spesso può essere difficile, se non impossibile, da trovare. Tuttavia i principali svantaggi nascono: (a) dal bisogno di scegliere un adeguato insieme di esempi; (b) quando la rete deve rispondere ad ingressi sostanzialmente diversi da quelli dell'insieme degli esempi.

L'ispirazione per le reti neurali deriva dagli studi sui meccanismi di elaborazione dell'informazione nel sistema nervoso biologico, in particolare il cervello umano; infatti, gran parte della ricerca sulle reti ha proprio lo scopo di capire più a fondo questi meccanismi. Daremo nel seguito un quadro semplificato del funzionamento delle reti neurali biologiche in modo da trovare una corrispondenza pratica dei concetti riguardanti le reti neurali artificiali.

Dalle loro origini ad oggi, sono stati sviluppati numerosi modelli di reti neurali per risolvere problemi di diversa natura, tuttavia qui saranno presentati principalmente due modelli conosciuti come *multilayer perceptron* e rete *radial basis function*. Essi fanno parte di una classe di modelli di reti generica conosciuta come reti feedforward, ovvero reti in cui l'informazione si muove in un'unica direzione ed in cui non ci sono cicli. Attualmente questi due modelli stanno alla base della maggior parte di applicazioni pratiche.

Oltre a descrivere la struttura dei modelli in dettaglio, parleremo di una tecnica usata per addestrarli (concetto che sarà chiarito più avanti) e discuteremo una serie di situazioni che devono essere affrontate quando si applicano le reti neurali a problemi pratici.

Infine, daremo due esempi di applicazione delle reti neurali.

2 Definizioni e metriche di prestazione

2.1 Reti neurali biologiche

I neuroni sono delle celle elettricamente attive ed il cervello umano ne contiene circa 10^{11}. Esistono in forme diverse, anche se la maggior parte di essi ha la forma indicata nella figura 1. I dendriti rappresentano gli ingressi del neurone mentre l'assone ne rappresenta l'uscita. La comunicazione tra i neuroni avviene alle giunzioni, chiamate sinapsi. Ogni neurone è tipicamente connesso ad un migliaio di altri neuroni e, di conseguenza, il numero di sinapsi nel cervello supera 10^{14}.

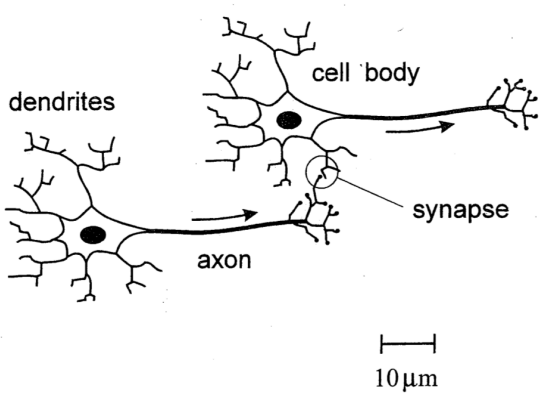

Figura 1: Illustrazione schematica di due neuroni biologici.

Ogni neurone si può trovare principalmente in 2 stati: attivo oppure a riposo. Quando il neurone si attiva esso produce un potenziale di azione (impulso elettrico) che viene trasportato lungo l'assone. Una volta che il segnale raggiunge la sinapsi esso provoca il rilascio di sostanze chimiche (neurotrasmettitori) che attraversano la giunzione ed entrano nel corpo di altri neuroni. In base al tipo di sinapsi, che possono essere eccitatori o inibitori, queste sostanze aumentano o diminuiscono rispettivamente la probabilità che il successivo neurone si attivi. Ad ogni sinapsi è associato un peso che ne determina il tipo e l'ampiezza dell'effetto eccitatore o inibitore. Quindi, in poche parole, ogni neurone effettua una somma pesata degli ingressi proveni-

enti dagli altri neuroni e, se questa somma supera una certa soglia, il neurone si attiva.

Ogni neurone, operando ad un ordine temporale del millisecondo, rappresenta un sistema di elaborazione relativamente lento; tuttavia, l'intera rete ha un numero molto elevato di neuroni e sinapsi che possono operare in modo parallelo e simultaneo, rendendo l'effettiva potenza di elaborazione molto elevata. Inoltre la rete neurale biologica ha un'alta tolleranza ad informazioni poco precise (o sbagliate), ha la facoltà di apprendimento e generalizzazione.

2.2 Reti neurali artificiali

Da qui in poi ci riferiremo alle reti neurali artificiali usando semplicemente l'espressione *reti neurali*. Nel caso ci riferissimo alle reti biologiche, questo sarà specificato.

Come abbiamo detto nell'introduzione, ci concentreremo su una classe particolare di modelli di reti neurali: le reti a catena aperta (feedforward). Queste reti possono essere viste come funzioni matematiche non lineari che trasformano un insieme di variabili indipendenti $\mathbf{x} = (x_1, ..., x_d)$, chiamate ingressi della rete, in un insieme di variabili dipendenti $\mathbf{y} = (y_1, ..., y_c)$, chiamate uscite della rete. La precisa forma di queste funzioni dipende dalla struttura interna della rete e da un insieme di valori $\mathbf{w} = (w_1, ..., w_d)$, chiamati pesi. Possiamo quindi scrivere la funzione della rete nella forma $\mathbf{y} = \mathbf{y}(\mathbf{x}; \mathbf{w})$ che denota il fatto che $\mathbf{y}$ sia una funzione di $\mathbf{x}$ parametrizzata da $\mathbf{w}$.

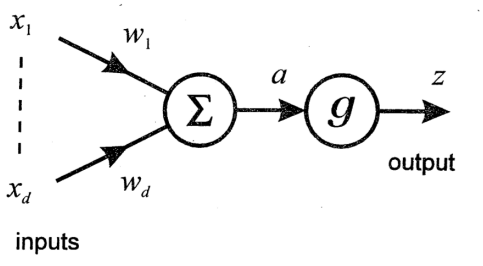

Figura 2: Modello di McCulloch-Pitts

Modello di McCulloch-Pitts:

Un semplice modello matematico di un singolo neurone è quello rappresentato in figura 2 ed è stato proposto da McCulloch e Pitts alle origini delle reti neurali. Esso può essere visto come una funzione non lineare che trasforma le variabili di ingresso $x_1, ..., x_d$ nella variabile di uscita z. Nel resto dell'elaborato ci riferiremo a questo modello come *unità di elaborazione*, o semplicemente *unità*.

In questo modello, viene effettuata la somma ponderata degli ingressi, usando come pesi i valori $w_1, ..., w_d$ (che sono analoghi alle potenze delle sinapsi nella rete biologica), ottenendo così:

$$a = \sum_{i=1}^{d} w_i x_i + w_0, \tag{1}$$

dove il parametro w_0 viene chiamato *bias* (esso corrisponde alla soglia di attivazione del neurone biologico). Se definiamo un ulteriore ingresso x_0, impostato costantemente a 1, possiamo scrivere la (1) come

$$a = \sum_{i=0}^{d} w_i x_i, \tag{2}$$

dove $x_0 = 1$. Precisiamo che i valori dei pesi possono essere di qualsiasi segno, che dipende dal tipo di sinapsi. L'uscita z (che può essere vista come

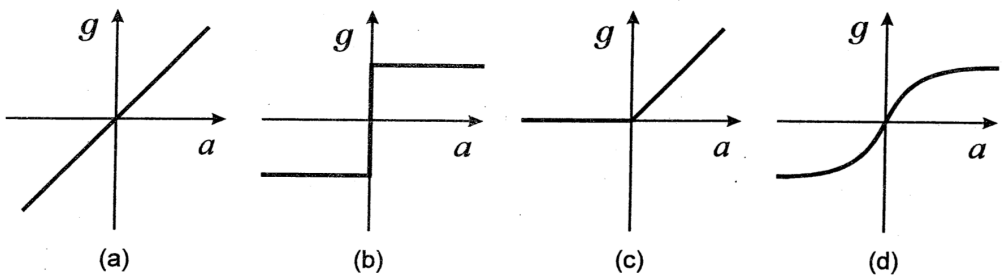

Figura 3: Alcune funzioni di attivazione tipiche.

tasso medio di attivazione del neurone biologico) viene ottenuta applicando

ad a una trasformazione non lineare $g()$, chiamata funzione di attivazione, ottenendo

$$z = g(a) = g\left(\sum_{i=0}^{d} w_i x_i\right). \tag{3}$$

Alcune possibili forme per la funzione $g()$ sono rappresentate in figura 3. Il modello originale di McCulloch-Pitts usava la funzione gradino in figura 3(b).

2.3 Addestramento della rete e la funzione di errore

Abbiamo detto che una rete neurale può essere rappresentata dal modello matematico $\mathbf{y} = \mathbf{y}(\mathbf{x}; \mathbf{w})$, che è una funzione di $\mathbf{x}$ parametrizzata dai pesi $\mathbf{w}$. Prima di poter utilizzare però questa rete, dobbiamo identificare il modello, ovvero dobbiamo determinare tutti i parametri $\mathbf{w}$. Il processo di determinazione di questi parametri è chiamato *addestramento* e può essere un'azione molto intensa dal punto di vista computazionale. Tuttavia, una volta che sono stati definiti i pesi, nuovi ingressi possono essere processati molto rapidamente.

Per addestrare una rete abbiamo bisogno di un insieme di esempi, chiamato *insieme di addestramento*, i cui elementi sono coppie $(\mathbf{x}^q, \mathbf{t}^q)$, $q = 1, ..., n$, dove $\mathbf{t}^q$ rappresenta il valore di uscita desiderato, chiamato target, in corrispondenza del ingresso $\mathbf{x}^q$. L'addestramento consiste nella ricerca dei valori per i parametri $\mathbf{w}$ che minimizzano un'opportuna funzione di errore. Ci sono diverse forme di questa funzione, tuttavia la più usata risulta essere la somma dei quadrati residui. I residui sono definiti come

$$r_{qk} = y_k(\mathbf{x}^q; \mathbf{w}) - t_k^q. \tag{4}$$

La funzione di errore E risulta allora essere

$$E = \frac{1}{2} \sum_{q=1}^{n} \sum_{k=1}^{c} r_{qk}^2. \tag{5}$$

E' facile osservare che E dipende da $\mathbf{x}^q$ e da $\mathbf{t}^q$ che sono valori noti e da $\mathbf{w}$ che è incognito, quindi E è in realtà una funzione dei soli pesi $\mathbf{w}$.

Infine notiamo che questo procedimento di addestramento è molto simile concettualmente al metodo dei minimi quadrati, usato per trovare una funzione y=y(x;**w**) (ovvero una funzione scalare di una variabile) che rappresenti al meglio la tendenza di un insieme di dati.

3 Modelli di reti neurali

3.1 Multilayer perceptron

Parleremo ora più in dettaglio del modello multilayer perceptron (MLP). Introdurremo il concetto di multistrato ed affronteremo in particolare il modello a singolo strato ed il modello a due strati, ricavando le equazioni che li rappresentano. Infine discuteremo un metodo di addestramento della rete, chiamato discesa del gradiente.

Anche se il modello a due strati ha una capacità di approssimazione universale, non è raro l'uso di reti a più strati. Tuttavia le reti generiche vanno oltre lo scopo di questo elaborato e non saranno affrontate.

3.1.1 Architettura del modello MLP

MLP a uno strato:

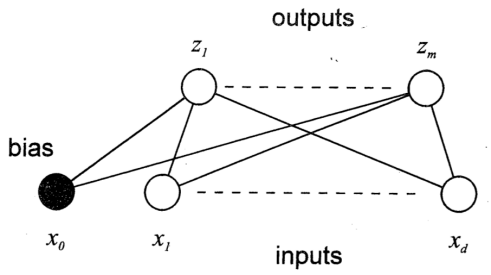

Figura 4: Perceptron a uno strato

Nel capitolo precedente abbiamo parlato della singola unità di elaborazione, descritta dall'equazione (3). Se consideriamo un insieme di m tali unità, aventi ingressi comuni, otteniamo una rete neurale a singolo strato della figura 4. Le uscite di questa rete sono date da

$$z_j = g\left(\sum_{i=0}^{d} w_{ji} x_i\right), \quad j = 1, ..., m, \tag{6}$$

dove w_{ji} rappresenta il peso che connette l'ingresso i con l'uscita j; $g()$ è una funzione di attivazione. Anche in questo caso abbiamo posto $x_0 = 1$.

Reti a singolo strato come questa erano note con il termine *perceptron* ed avevano una capacità computazionale molto limitata. Esse venivano addestrate tramite un algoritmo dedicato chiamato *perceptron learning algorithm* e facevano solitamente uso della funzione di attivazione a gradino di figura 3(b).

MLP a due strati:

Per ottenere reti più potenti è necessario considerare reti aventi più strati successivi di unità di elaborazione, chiamate *multilayer perceptron*. L'introduzione di queste reti comporta principalmente un problema, ovvero quello di trovare un algoritmo di addestramento adeguato (il perceptron learning algorithm funziona solo per reti a singolo strato). La soluzione a questo problema consiste nell'usare una funzione di attivazione sigmoidale della forma di figura 3(d). Questa funzione, essendo differenziabile, permette l'uso di strumenti del calcolo differenziale allo scopo di trovare un appropriato algoritmo di addestramento. In figura 5 è illustrata una rete neurale a due strati. Le

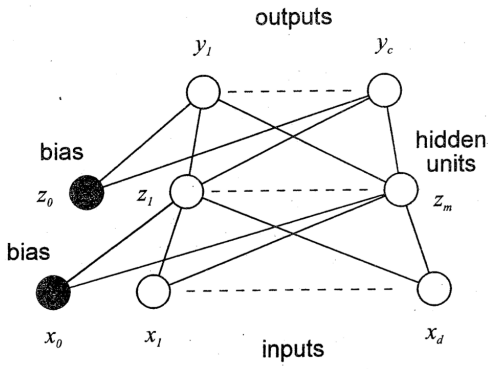

Figura 5: Perceptron a due strati

unità dello strato centrale sono chiamate *unità nascoste*, poiché il loro valore di attivazione non è direttamente misurabile dall'esterno. L'attivazione

di queste unità è data dall'equazione (6). Le uscite della rete vengono ottenute tramite una seconda trasformazione, analoga alla prima, sui valori z_j ottenendo

$$y_k = \tilde{g}\left(\sum_{j=0}^{m} \tilde{w}_{kj} z_j\right), \quad k = 1, ..., c, \tag{7}$$

dove $\tilde{w}_{kj}$ rappresenta il peso del secondo strato che connette l'unità nascosta j all'unità di uscita k. Notiamo che è stata introdotta, anche qui, un'ulteriore unità nascosta z_0, impostata costantemente ad 1. I termini di bias hanno un ruolo importante e permettono di rappresentare funzioni non lineari generiche. Sostituendo infine l'equazione (6) nell'equazione (7) otteniamo l'espressione completa che rappresenta la rete:

$$y_k = \tilde{g}\left(\sum_{j=0}^{m} \tilde{w}_{kj} g\left(\sum_{i=0}^{d} w_{ji} x_i\right)\right), \quad k = 1, ..., c. \tag{8}$$

Notiamo che la funzione di attivazione $\tilde{g}$, applicata alle unità di uscita, non deve essere necessariamente uguale alla funzione g, applicata alle unità nascoste.

Per ottenere una capacità di rappresentazione universale, la funzione di attivazione $g()$, delle unità nascoste, deve essere scelta non lineare. Infatti, se le funzioni $g()$ e $\tilde{g}()$ fossero entrambe lineari, l'equazione della rete (8) diventerebbe un semplice prodotto di matrici, che è esso stesso una matrice. Inoltre abbiamo detto che, per poter trovare un algoritmo di addestramento adeguato, le funzioni di attivazione devono essere differenziabili. Per queste ragioni vengono spesso usate funzioni sigmoidali della forma di figura 3(d). Alcune possibili scelte possono essere la funzione tangente iperbolica e la funzione sigmoidea:

$$g_1(a) = tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}, \tag{9}$$

$$g_2(a) = \frac{1}{1 + e^{-a}}. \tag{10}$$

Entrambe queste funzioni hanno la derivata che può essere espressa in termini

della funzione stessa, ovvero

$$g_1^{'}(a) = 1 + g_1(a)^2, \tag{11}$$

$$g_2^{'}(a) = g_2(a)[1 - g_2(a)]. \tag{12}$$

Questa proprietà risulta molto utile nell'addestramento della rete.
Per quanto riguarda le unità di uscita, la funzione di attivazione può essere scelta sia lineare che non. Questo dipende dal tipo di applicazione a cui è dedicata la rete. Ad esempio, se vogliamo che l'uscita sia limitata ad un intervallo, è utile la funzione sigmoidea (essa restituisce valori nell'intervallo (0,1)), altrimenti possiamo usare la funzione lineare.

E' utile infine notare che, dato un numero di unità nascoste sufficientemente ampio, il perceptron a due strati può rappresentare, con arbitraria approssimazione, una qualsiasi funzione continua, definita su un insieme limitato dello spazio delle variabili d'ingresso.

3.1.2 Algoritmo di addestramento della rete

Come abbiamo già detto, l'addestramento consiste nella ricerca dei valori di $\mathbf{w} = (w_1, ..., w_d)$ che minimizzano una determinata funzione di errore $E(\mathbf{w})$. Questa funzione può essere rappresentata graficamente come un'ipersuperficie in $\mathbb{R}^{d+1}$, come indicato in figura 6 per il caso bidimensionale ($d = 2$). In generale la funzione di errore è una funzione non lineare molto complessa e ha diversi minimi (vedi figura 6). La ricerca del minimo avviene, principalmente, in modo iterativo, partendo da un valore iniziale di $\mathbf{w}$, scelto in modo casuale (o tramite qualche criterio). Alcuni algoritmi trovano il minimo locale più vicino (al punto iniziale), mentre altri riescono a trovare il minimo globale. Tuttavia, data la complessità di E, molte volte è sufficiente trovare un buon minimo locale per ottenere risultati soddisfacienti.

Diversi algoritmi di ricerca del punto di minimo fanno uso delle derivate parziali della funzione di errore E, ovvero del suo vettore gradiente ∇E. Questo vettore, come indicato in figura 6, indica la direzione ed il verso di massima crescita di E nel punto $\mathbf{w}$. Una delle importanti caratteristiche

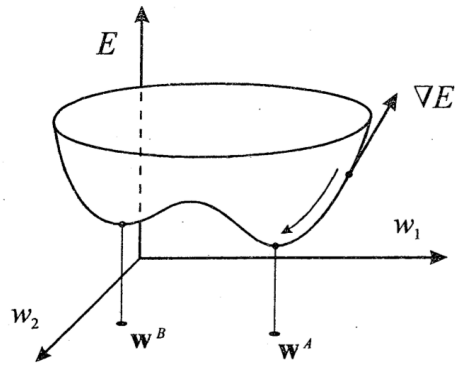

Figura 6: Esempio di funzione di errore $E(\mathbf{w})$; $\mathbf{w}^A$ e $\mathbf{w}^B$ rappresentano, rispettivamente, i punti di minimo globale e minimo locale

del modello MLP è il fatto che esiste un metodo efficiente per calcolare le derivate parziali, basato sull'algoritmo di *error backpropagation*.

Descriviamo ora in dettaglio questo procedimento e, successivamente, vediamo come il vettore ∇E possa essere utilizzato per trovare il minimo della funzione di errore. Consideriamo il modello a due strati, sebbene, con dovute precauzioni, il procedimento possa essere esteso a reti generiche.

Algoritmo di error backpropagation:

Studiamo il caso in cui la funzione di errore è rappresentata dalla somma dei quadrati residui (equazione (5)) e notiamo subito che essa può essre scritta nella seguente forma

$$E = \sum_{q=1}^{n} E^q, \quad E^q = \frac{1}{2} \sum_{k=1}^{c} [y_k(\mathbf{x}^q; \mathbf{w}) - t_k^q]^2, \tag{13}$$

dove $y_k(\mathbf{x}; \mathbf{w})$ è dato dall'equazione (8). Questo significa che possiamo vedere E come somma di termini E^q, ognuno dei quali corrisponde alla coppia $(\mathbf{x}^q, \mathbf{t}^q)$. Di conseguenza, per la linearità dell'operatore di derivazione, possiamo calcolare la derivata di E come somma delle derivate dei termini E^q. Osserviamo subito che nella seguente discussione ommetteremo l'indice q da diverse variabili, con lo scopo di semplificarne la notazione. Tuttavia,

dovrebbe essere tenuto in mente che tutti gli ingressi e le variabili intermedie sono calcolate per una specifica coppia di valori $(\mathbf{x}^q, \mathbf{t}^q)$ dell'insieme di addestramento.

Consideriamo prima le derivate rispetto ai pesi del secondo strato. Riscriviamo l'equazione (7) nel seguente modo

$$y_k = \tilde{g}(\tilde{a}_k), \quad \tilde{a}_k = \sum_{j=0}^{m} \tilde{w}_{kj} z_j. \tag{14}$$

La derivata di E^q rispetto ad un generico peso $\tilde{w}_{kj}$ del secondo strato può essere scritta come segue

$$\frac{\partial E^q}{\partial \tilde{w}_{kj}} = \frac{\partial E^q}{\partial \tilde{a}_k} \frac{\partial \tilde{a}_k}{\partial \tilde{w}_{kj}}. \tag{15}$$

Dall'equazione (14) è facile trovare che

$$\frac{\partial \tilde{a}_k}{\partial \tilde{w}_{kj}} = z_j, \tag{16}$$

mentre dalle equazioni (13) e (14) abbiamo che

$$\frac{\partial E^q}{\partial \tilde{a}_k} = \tilde{g}'(\tilde{a}_k)[y_k - t_k]. \tag{17}$$

Otteniamo quindi che

$$\frac{\partial E^q}{\partial \tilde{w}_{kj}} = \tilde{g}'(\tilde{a}_k)[y_k - t_k] z_j. \tag{18}$$

Se a questo punto definiamo

$$\tilde{\delta}_k = \frac{\partial E^q}{\partial \tilde{a}_k} = \tilde{g}'(\tilde{a}_k)[y_k - t_k], \tag{19}$$

otteniamo una semplice espressione per la derivata di E^q rispetto a $\tilde{w}_{kj}$, ovvero

$$\frac{\partial E^q}{\partial \tilde{w}_{kj}} = \tilde{\delta}_k z_j. \tag{20}$$

E' da notare a questo punto che, nel caso di funzioni di attivazione sigmoidali (equazioni (9) e (10)) discusse prima, la derivata $\tilde{g}'(a)$ è facilmente espressa in funzione di $\tilde{g}(a)$. Questo offre un piccolo risparmio computazionale nell'implementazione numerica dell'algoritmo. Inoltre, osserviamo che la derivata di E^q, rispetto ad un qualsiasi peso in una rete MLP generica, può essere espressa nella forma dell'equazione (20).

Per quanto riguarda le derivate rispetto ai pesi del primo strato, riscriviamo prima di tutto l'equazione (6) nel seguente modo

$$z_j = g(a_j), \quad a_j = \sum_{i=0}^{d} w_{ji} x_i. \tag{21}$$

Possiamo allora scrivere la derivata come

$$\frac{\partial E^q}{\partial w_{ji}} = \frac{\partial E^q}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}. \tag{22}$$

Anche in questo caso, osservando l'equazione (21), è semplice trovare che

$$\frac{\partial a_j}{\partial w_{ji}} = x_i, \tag{23}$$

mentre il calcolo della derivata di E^q rispetto ad a_j risulta un po' più complesso. Usando la regola della catena di derivazione, abbiamo che

$$\frac{\partial E^q}{\partial a_j} = \sum_{k=1}^{c} \frac{\partial E^q}{\partial \tilde{a}_k} \frac{\partial \tilde{a}_k}{\partial a_j}, \tag{24}$$

dove la derivata di E^q rispetto ad $\tilde{a}_k$ è data dall'equazione (17), mentre la derivata di $\tilde{a}_k$ rispetto ad a_j la troviamo usando le equazioni (14) e (21):

$$\frac{\partial \tilde{a}_k}{\partial a_j} = \tilde{w}_{kj} g'(a_j). \tag{25}$$

Utilizzando allora le equazioni (19),(24) e (25) otteniamo che

$$\frac{\partial E^q}{\partial a_j} = g'(a_j) \sum_{k=1}^{c} \tilde{w}_{kj} \tilde{\delta}_k, \tag{26}$$

e possiamo quindi riscrivere la (22) nella seguente forma

$$\frac{\partial E^q}{\partial w_{ji}} = g'(a_j) x_i \sum_{k=1}^{c} \tilde{w}_{kj} \tilde{\delta}_k. \tag{27}$$

Se infine, come abbiamo fatto nell'equazione (19), poniamo

$$\delta_j = \frac{\partial E^q}{\partial a_j} = g'(a_j) \sum_{k=1}^{c} \tilde{w}_{kj} \tilde{\delta}_k, \tag{28}$$

otteniamo l'equazione

$$\frac{\partial E^q}{\partial w_{ji}} = \delta_j x_i, \tag{29}$$

che ha la stessa semplice forma dell'equazione (20).

Riassumiamo allora i vari passi che è necessario fare per valutare la derivata della funzione E:

1. Per ogni coppia $(\mathbf{x}^q, \mathbf{t}^q)$, valutare le attivazioni delle unità nascoste e delle unità di uscita, usando rispettivamente le equazioni (21) e (14);

2. Valutare il valore di $\tilde{\delta}_k$ per $k = 1, ..., c$, usando l'equazione (19);

3. Valutare il valore di δ_j per $j = 1, ..., m$, usando l'equazione (28).

4. Valutare il valore delle derivate di E^q per questa particolare coppia, usando le equazioni (20) e (29);

5. Ripetere questi 4 passi per ogni coppia $(\mathbf{x}^q, \mathbf{t}^q)$ dell'insieme di addestramento e, successivamente, sommare tutte le derivate per ottenere la derivata della funzione di errore E.

Un'importante caratteristica di questo algoritmo è sicuramente l'efficienza computazionale. Si può infatti dimostrare che il tempo necessario per trovare

le derivate cercate, usando questo algoritmo, è $O(3nN)$ dove n, come sappiamo già, è la cardinalità dell'insieme di addestramento, mentre N rappresenta il numero totale di pesi della rete.

Discesa del gradiente:

Vediamo ora in che modo possiamo usare il vettore $\nabla E(\mathbf{w})$ per trovare il minimo della funzione di errore. Il metodo più semplice è un metodo iterativo e si basa sulla tecnica di discesa del gradiente. Ad ogni iterazione, aggiorniamo i pesi tramite la seguente espressione

$$\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} + \Delta\mathbf{w}^{(\tau)}, \tag{30}$$

dove

$$\Delta\mathbf{w}^{(\tau)} = -\eta\nabla E(\mathbf{w}^{(\tau)}). \tag{31}$$

Il parametro η (> 0) è chiamato *tasso di apprendimento* ed il suo valore deve essere scelto con cura. Infatti, un valore troppo piccolo può rendere eccessivo il tempo di ricerca del minimo, mentre un valore troppo grande può portare l'algoritmo all'instabilità. La situazione è ulteriormente complicata dal fatto che il valore ottimale di η cambia generalmente ad ogni passo di iterazione.

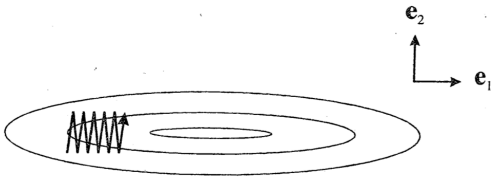

Figura 7: Curve di livello di E nel caso bidimensionale e andamento della ricerca del minimo usando l'equazione (31)

Uno dei problemi principali della discesa del gradiente sorge quando la superficie dell'errore ha la curvatura lungo una direzione $\mathbf{e}_1$ (nello spazio dei pesi) che è sostanzialmente più piccola della curvatura lungo un'altra direzione $\mathbf{e}_2$. Il tasso di apprendimento, in questo caso, deve essere molto piccolo al fine di evitare oscillazioni divergenti lungo la direzione $\mathbf{e}_2$, e questo

comporta un lento avanzamento in direzione $\mathbf{e}_1$. Questa situazione è rappresentata in figura 7, per il caso bidimensionale. Un modo per ovviare a questo problema consiste nell'introdurre un ulteriore termine nell'equazione (31), ovvero

$$\Delta \mathbf{w}^{(\tau)} = -\eta \nabla E(\mathbf{w}^{(\tau)}) + \mu \Delta \mathbf{w}^{(\tau-1)}, \tag{32}$$

dove μ $(0 \leq \mu < 1)$ è un parametro, chiamato *momento*, e viene definito manualmente. Questo termine ha la funzione di aumentare il tasso di apprendimento in direzioni di bassa curvatura, mentre ha l'effetto opposto in direzioni di alta curvatura. Con questa modifica si possono raggiungere i miglioramenti rappresentati in figura 8.

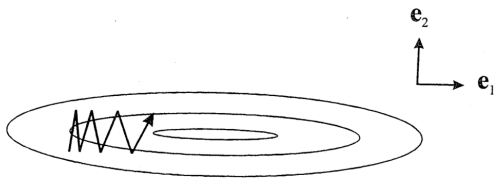

Figura 8: Curve di livello di E nel caso bidimensionale e andamento della ricerca del minimo usando l'equazione (32).

Le funzioni di errore per l'addestramento di reti neurali sono estremamente non lineari e non c'è una regola per decidere quando fermare il processo di ricerca del minimo. In pratica, tuttavia, sono usati diversi criteri di interruzione. Alcuni esempi possono essere: effettuare un numero prestabilito di iterazioni oppure fermarsi quando la riduzione di E nelle ultime x (dove x è un intero predefinito) iterazioni è inferiore ad una soglia prestabilita.

Ci sono diversi algoritmi di addestramento delle reti neurali e la discesa del gradiente è soltanto uno di questi. La scelta sull'algoritmo più addatto da utilizzare varia dal tipo di applicazione. La discussione di altri algoritmi, tuttavia, non sarà qui affrontata.

3.2 Radial basis function network

L'addestramento di una rete MLP può essere molto intenso dal punto di vista computazionale. Nella pratica, infatti, può essere necessario ripetere il processo di addestramento molte volte al fine di ottimizzarne i parametri. Inoltre la struttura interna di una rete MLP, ovvero attivazioni delle unità nascoste, può essere molto difficile da interpretare. Ci sono applicazioni in cui queste limitazioni sono importanti e richiedono lo sviluppo di altri modelli che cerchino di ovviare a questi problemi. Presentiamo ora uno di questi modelli, chiamato rete *radial basis function* (RBF).

3.2.1 Architettura della rete RBF

La rete RBF si basa sul fatto che una funzione arbitraria, $y(x)$, può essere rappresentata, approssimativamente, come sovrapposizione di funzioni localizzate, $\phi(x)$, chiamate *funzioni di base*.

La rete RBF viene rappresentata dal seguente insieme di funzioni

$$y_k(\mathbf{x}) = \sum_{j=0}^{m} \tilde{w}_{kj}\phi_j(\mathbf{x}), \quad k = 1, ..., c, \tag{33}$$

dove m rappresenta il numero di funzioni di base. Questo numero, per ragioni che saranno chiarite più avanti, viene solitamente scelto minore di n (numero di elementi di addestramento). La funzione $\phi_j(\mathbf{x})$ rappresenta l'attivazione dell'unità nascosta j, quando in ingresso è presente il vettore $\mathbf{x}$. Notiamo che è stato incluso, anche in questo caso, il termine di bias per l'uscita ed è stato rappresentato da una funzione di base extra ϕ_0, posta costante a 1.

La funzione di base è una funzione a simmetria radiale e la scelta ricade spesso sulla funzione gaussiana

$$\phi_j(\mathbf{x}) = exp\left(-\frac{|\mathbf{x} - \boldsymbol{\mu}_j|^2}{2\sigma_j^2}\right), \tag{34}$$

dove $\boldsymbol{\mu}_j$ è un vettore che rappresenta il centro della j-esima funzione, mentre σ_j è un parametro che ne rappresenta la larghezza.

La rete RBF può essere rappresentata schematicamente dal diagramma di figura 9. Vediamo che ogni unità nascosta corrisponde ad una funzione di base, mentre le linee che connettono gli ingressi all'unità nascosta j rappresentano il vettore $\boldsymbol{\mu}_j$. Osserviamo inoltre che, invece del parametro di bias, ogni unità nascosta adesso ha il parametro σ_j. Per quanto riguarda il secondo strato della rete, esso è identico a quello del modello MLP nel quale le unità di uscita hanno funzioni di attivazione lineari.

Notiamo infine che, se abbiamo un numero sufficiente di unità nascoste e abbiamo scelto con cura i parametri $(\boldsymbol{\mu}_j, \sigma_j)$, anche le reti RBF possono rappresentare, con arbitraria approssimazione, una qualsiasi funzione continua.

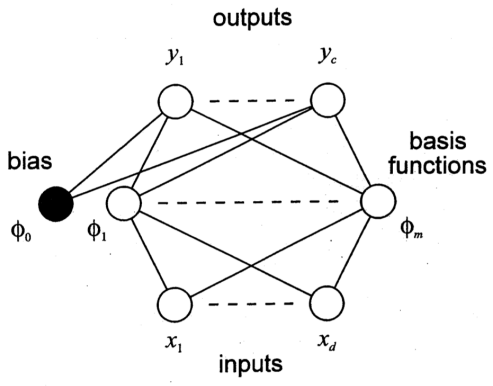

Figura 9: Architettura di una rete RBF

3.2.2 Stima dei parametri

L'addestramento della rete RBF avviene, come nel modello MLP, minimizzando una funzione di errore della forma dell'equazione (5). Essendo la funzione della rete una funzione analitica, l'addestramento potrebbe essere fatto ottimizzando i valori dei parametri tramite il metodo discusso nella sezione 3.1.2. Questo approccio, tuttavia, non porterebbe significativi vantaggi rispetto al modello MLP.

Un approccio molto più veloce deriva dal fatto che le attivazioni delle unità nascoste hanno una forma localizzata (vedi figura 10), ovvero assumono

un valore significativamente diverso da zero soltanto in una regione limitata dello spazio (delle variabili d'ingresso). Questo porta ad un addestramento che consiste in due fasi: nella prima vengono ottimizzati i parametri $\boldsymbol{\mu}_j$ e σ_j della funzione di base, mentre nella seconda vengono determinati i parametri $\tilde{w}_{kj}$ del secondo strato.

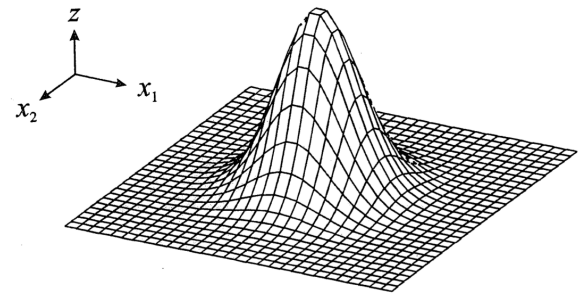

Figura 10: Funzione di base gaussiana nel caso bidimensionale

Scelta dei parametri della funzione di base:

Il metodo più semplice e veloce per scegliere i parametri $\boldsymbol{\mu}_j$ è quello di porli uguali ad un sottoinsieme dei vettori d'ingresso dell'insieme di addestramento. Questo sottoinsieme è solitamente scelto in maniera casuale.

Per quanto riguarda i parametri σ_j, essi possono essere scelti tutti uguali e dati dalla distanza media tra i centri $\boldsymbol{\mu}_j$ delle diverse funzioni di base. Questo garantisce che le funzioni $\phi_j(\mathbf{x})$ si sovrappongano e rappresentino l'insieme dei dati di addestramento con un certo grado di regolarità. Ci sono, ovviamente, tecniche più complicate, che però qui non saranno affrontate.

Scelta dei parametri del secondo strato:

Dopo aver scelto i parametri $\boldsymbol{\mu}_j$ e σ_j delle funzioni di base, vediamo come possiamo scegliere i parametri $\tilde{w}_{kj}$. Consideriamo a questo proposito la funzione di errore dell'equazione (5), dove y_k è rappresentato dall'equazione (33). Notiamo che y_k è una funzione lineare dei parametri $\tilde{w}_{kj}$ e che E è una funzione quadratica di questi pesi. Possiamo allora cercare il punto di

minimo della funzione E ponendo a zero le sue derivate parziali rispetto ai pesi $\tilde{w}_{kj}$, ovvero

$$\frac{\partial E}{\partial \tilde{w}_{kj}} = \sum_{q=1}^{n} \phi_j^q \left(\sum_{j'=1}^{m} \tilde{w}_{kj'} \phi_{j'}^q - t_k^q \right) = 0 \quad k = 1, ..., c, \quad j = 1, ..., m, \tag{35}$$

dove $\phi_j^q = \phi_j(\mathbf{x}^q)$. Se definiamo allora le seguenti matrici:

$$\mathbf{\Phi} = \begin{bmatrix} \phi_1^1 & \dots & \phi_m^1 \\ \vdots & \ddots & \vdots \\ \phi_1^n & \dots & \phi_m^n \end{bmatrix}, \mathbf{W} = \begin{bmatrix} \tilde{w}_{11} & \dots & \tilde{w}_{1m} \\ \vdots & \ddots & \vdots \\ \tilde{w}_{c1} & \dots & \tilde{w}_{cm} \end{bmatrix},$$

$$\mathbf{T} = \begin{bmatrix} t_1^1 & \dots & t_c^1 \\ \vdots & \ddots & \vdots \\ t_1^n & \dots & t_c^n \end{bmatrix},$$

possiamo scrivere l'equazione (35) nella seguente forma matriciale:

$$\mathbf{\Phi}^T(\mathbf{\Phi}\mathbf{W}^T - \mathbf{T}) = 0, \tag{36}$$

dove l'incognita è chiaramente rappresentata dalla matrice $\mathbf{W}$ e l'apice T indica la trasposizione. E' facile allora, dopo alcuni semplici passaggi, arrivare alla soluzione di questa equazione, ovvero

$$\mathbf{W}^T = (\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{T}. \tag{37}$$

Questo significa che i valori ottimali dei pesi $\tilde{w}_{kj}$ possono essere trovati in forma esplicita e sono dati dall'equazione (37). Notiamo tuttavia che essi dipendono dai valori $\boldsymbol{\mu}_j$ e σ_j.

Osserviamo che la matrice $\mathbf{\Phi}$ ha dimensione $n \times m$, dove n rappresenta il numero di esempi di addestramento, mentre m rappresenta il numero di unità nascoste. Se $m = n$ la matrice $\mathbf{\Phi}$ diventa quadrata e l'equazione (37) diventa semplicemente

$$\mathbf{W}^T = \mathbf{\Phi}^{-1}\mathbf{T}. \tag{38}$$

In questo caso le uscite della rete eguagliano i valori di target per ogni coppia $(\mathbf{x}^q, \mathbf{t}^q)$ di addestramento, e la funzione di errore E diventa zero. Questo caso corrisponde al metodo di interpolazione e, generalmente, non è una situazione desiderabile. Infatti le prestazioni della rete, in questo caso, diventano scadenti su nuovi ingressi ed in pratica m dovrebbe essere molto minore di n.

4 Tecniche di ottimizzazione

4.1 Generalizzazione

Nei capitoli precedenti, abbiamo parlato di due particolari modelli di reti neurali. Abbiamo anche mostrato come possiamo determinare i parametri di queste reti in modo da rappresentare, con ragionevole accuratezza, i dati di addestramento. Molto più importante è, tuttavia, il problema di ottenere delle buone prestazioni in risposta ad ingressi che non fanno parte dell'insieme di addestramento. Questo problema è chiamato *generalizzazione*.

Il problema della generalizzazione consiste, in pratica, nella scelta del numero ottimale di unità nascoste in modo da: (a) catturare la tendenza dei dati di addestramento; (b) avere delle risposte affidabili quando sono presentati nuovi dati in ingresso (ossia diversi da quelli di addestramento). Si potrebbe pensare che la soluzione migliori all'aumentare del numero di unità nascoste; tuttavia, la pratica dice che non è così. Infatti, nella maggior parte di applicazioni pratiche, i dati di addestramento sono affetti da rumore, e una rete che rappresenta troppo bene questi dati imparerebbe anche gli errori in essi contenuti. La rete avrebbe, quindi, delle prestazioni scadenti in presenza di nuovi dati in ingresso. Questo fenomeno è chiamato *overfitting*. Allo stesso tempo comunque, un valore troppo basso di unità, non permetterebbe alla rete di catturare la tendenza dei dati di addestramento. Esiste, quindi, un valore ottimale $\widehat{m}$ del numero di unità nascoste. Vediamo, allora, il procedimento usato per trovare questo valore.

Consideriamo due insiemi indipendenti di dati che chiamiamo *insieme di addestramento* e *insieme di prova*. Usiamo il primo insieme per addestrare una rete neurale per diversi valori di unità nascoste m, e disegniamo l'andamento della funzione di errore E in funzione di m. Ci si aspetterà di ottenere una funzione monotona decrescente, poiché l'aggiunta di unità non dovrebbe aumentare l'errore (vedi figura 11). Presentiamo, allora, i dati di prova alle diverse reti addestrate e valutiamo il corrispondente valore di E. Si ottiene una funzione che inizialmente decresce all'aumentare di m; quando però m raggiunge un valore abbastanza alto, il fenomeno del overfitting entra

in gioco e la funzione comincia a crescere. Il valore ottimale del numero di unità nascoste m è quello che minimizza la funzione di errore di prova, ed è dato da $m = \widehat{m}$ in figura 11. Nelle applicazioni pratiche, il metodo più

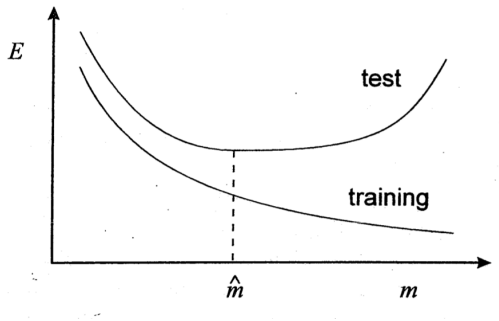

Figura 11: Rappresentazione dell'errore rispetto agli insiemi di addestramento e di prova, in funzione del numero di unità nascoste m; $m = \widehat{m}$ rappresenta il valore ottimale di m.

semplice per scegliere i due insiemi, di cui abbiamo appena parlato, consiste nel partizionare, in modo casuale, i dati di addestramento disponibili.

Notiamo, infine, che la figura 11 rappresenta un caso idealizzato. Nella pratica, grafici di questo tipo avranno diversi minimi locali e le funzioni saranno molto meno regolari. Un esempio è dato dalla figura 12.

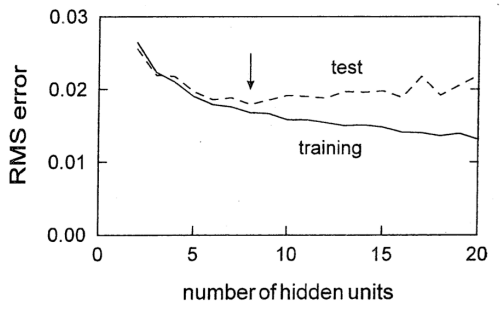

Figura 12: Rappresentazione dell'errore, presa da un applicazione reale, rispetto agli insiemi di addestramento e di prova, in funzione del numero di unità nascoste.

Un metodo più sofisticato per la generalizzazione è quello della *cross-validazione*. Questo metodo è particolarmente utile quando la quantità di dati per l'addestramento è limitata; tuttavia, è molto esigente dal punto di vista computazionale. Consiste nel dividere i dati disponibili in S sezioni di dimensione uguale. Ogni rete è allora addestrata usando $S - 1$ sezioni e, successivamente, testata usando la sezione rimanente. I due passi vengono ripetuti per le S possibili scelte della sezione di prova ed infine viene fatta una media dei risultati ottenuti. Questo procedimento viene allora ripetuto per le diverse topologie di rete ed infine viene scelta la rete che presenta il minor errore rispetto ai dati di prova.

4.2 Data preprocessing

Finora abbiamo supposto che, nella fase di addestramento, i dati siano applicati direttamente agli ingressi della rete. Si dimostra tuttavia che, questo approccio, in pratica, porta a reti con scarse prestazioni. Generalmente, infatti, i dati di ingresso vengono prima elaborati e successivamente applicati alla rete.

La pre-elaborazione consiste in una trasformazione applicata ai dati di ingresso. Nello stesso modo, le uscite della rete devono essere post-elaborate per ottenere dei valori coerenti con quelli di ingresso. Questo fatto è illustrato in figura 13. Analogamente, la trasformazione inversa della post-elaborazione

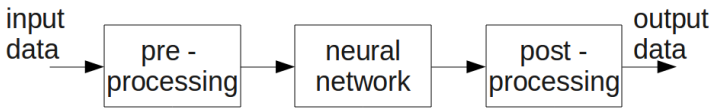

Figura 13: Pre/post-elaborazione dei dati

deve essere applicata ai valori di target in modo da ottenere dei valori corretti per l'addestramento. Osserviamo che, se si utilizza la pre/post-elaborazione, le trasformazioni devono essere impiegate sia nella fase di addestramento che in quella di utilizzo della rete.

Vediamo ora alcune possibile forme di pre-elaborazione.

Rescaling lineare:

Una delle più semplici forme di pre-elaborazione è sicuramente il rescaling lineare. Esso consiste in una semplice trasformazione invertibile volta a fare in modo che ogni dato di ingresso abbia media nulla e deviazione standard unitaria. Nelle applicazioni pratiche, infatti, non è raro incontrare grandezze fisiche che possono assumere un valore appartenente ad un ampio intervallo di valori. Un esempio è la frequenza. In casi come questo il rescaling lineare risulta essenziale e può essere ottenuto tramite la seguente trasformazione

$$\tilde{x}_i = \frac{x_i - \overline{x}_i}{s_i}, \tag{39}$$

dove

$$\overline{x}_i = \frac{1}{n}\sum_{q=1}^{n} x_i, \quad s_i^2 = \frac{1}{n-1}\sum_{q=1}^{n}(x_i - \overline{x}_i)^2. \tag{40}$$

Estrazione delle feature:

Un'altra forma di pre-elaborazione è la riduzione del numero di ingressi. E' stato scoperto che, in alcuni casi, può portare ad una migliore abilità di generalizzazione.

In generale questo approccio consiste nel trovare un numero di combinazioni delle variabili d'ingresso, chiamate feature, che rendano il compito delle reti neurali il più semplice possibile. Scegliendo un numero di feature inferiore al numero di variabili d'ingresso, si ottiene la riduzione. Il numero ottimale di feature dipende molto dal problema e può influire fortemente sulle prestazioni della rete.

Conoscenze a priori:

Ci sono delle applicazioni in cui, oltre all'insieme di addestramento, si hanno delle informazioni sulla rete addizionali, chiamate conoscenze a priori. Ad esempio invarianze che la funzione della rete deve rispettare.

Si possono allora sfruttare queste conoscenze incorporandole nella fase di pre-elaborazione. Inoltre, osserviamo che le conoscenze a priori possono essere usate, in alcuni casi, per decidere la topologia della rete.

5 Esempi

Nelle sezioni precedenti abbiamo parlato delle reti neurali da un punto di vista prevalentemente teorico. Vediamo ora due esempi di progetto di reti neurali MLP. Il primo riguarda un problema di regressione, mentre il secondo affronta un problema di classificazione.

Notiamo che gli esempi sono stati fatti usando il programma *Matlab* e sono state sfruttate diverse funzioni, riguardanti le reti neurali, che questo programma mette a disposizione.

Esempio 1

Iniziamo da un semplice problema di stima di una funzione sinusoidale.

Per questo esempio usiamo una rete a due strati con funzioni di attivazione sigmoidali e lineari, rispettivamente per lo strato nascosto e quello di uscita. Per addestrare la rete abbiamo bisogno di un insieme di dati che generiamo con le seguenti istruzioni

$x = -1 : 0.04 : 1;$ - ingressi
$t = sin(2 * pi * x) + 0.1 * randn(size(x));$ - target

dove i valori di target sono generati aggiungendo alla funzione sinusoidale numeri casuali con distribuzione normale. Da questo insieme generiamo due sottoinsiemi, uno per la fase di addestramento ed uno per la fase di prova:

$indTrain = 1 : 3 : length(x);$
$indTest = 2 : 6 : length(x);$
$xTrain = x(indTrain);$ - ingressi di addestramento
$xTest = x(indTest);$ - ingressi di prova
$tTrain = t(indTrain);$ - target di addestramento
$tTest = t(indTest);$ - target di prova

Notiamo che non vengono usati tutti gli elementi dell'insieme iniziale. I dati di addestramento e di prova sono rappresentati in figura 14. Creiamo adesso una rete neurale MLP a due strati, con le seguenti istruzioni

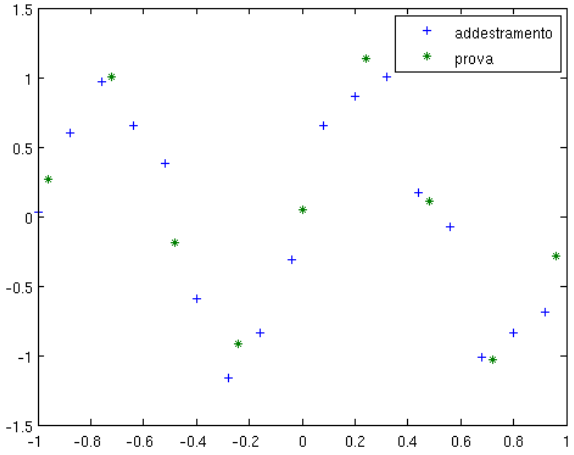

Figura 14: Insieme di addestramento ed insieme di prova

$net = fitnet(hiddenLayerSize)$; - crea la rete
$net.divideFcn = ''$; - disattiva il partizionamento automatico dei dati

ed impostiamo come funzione di pre e post elaborazione il rescaling lineare

$net.inputs1.processFcns = 'mapminmax'$;
$net.outputs2.processFcns = 'mapminmax'$;

che trasforma l'intervallo dei valori di ingresso e di uscita a $[-1, 1]$. Impostiamo, infine, le funzioni ed i parametri di addestramento:

$net.performFcn = 'sse'$; - somma dei quadrati residui
$net.performParam.squaredWeighting = false$; - no somma pesata
$net.trainFcn = 'traingdm'$; - discesa gradiente con momento (eq. (32))
$net.trainParam.lr = 0.01$; - tasso di addestramento
$net.trainParam.mc = 0.9$; - momento
$net.trainParam.epochs = 500$; - numero di cicli di ricerca del minimo
$net.trainParam.min\_grad = 0$;

A questo punto possiamo addestrare la rete e calcolare le funzioni di errore rispetto ai due insiemi di addestramento

$[net, tr] = train(net, xTrain, tTrain);$
$Etrain = perform(net, tTrain, yTrain);$
$Etest = perform(net, tTest, yTest);$

Ripetendo questi passi per un diverso numero di unità nascoste h, possiamo vedere l'andamento delle funzioni di errore al variare di h (vedi figura 15). Da

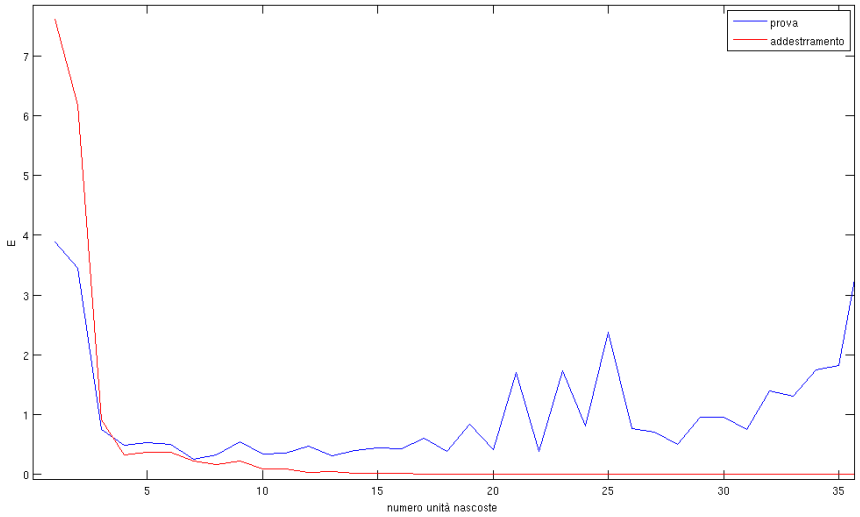

Figura 15: Funzioni di errore rispetto agli insiemi di addestramento e di prova

questa figura notiamo che l'andamento delle funzioni rispetta quello descritto nella sezione 4.1. Si trova che la funzione di errore di prova assume il valore minimo quando lo strato nascosto ha 7 unità. In questo caso il risultato che otteniamo è rappresentato in figura 16.

Bisogna tuttavia precisare una cosa. I valori iniziali dei pesi e di bias sono fissati usando l'algoritmo di Nguyen-Widrow. Per lo scopo di questo esempio non sono importanti i dettagli di funzionamento di questo algoritmo ma soltanto il fatto che i valori iniziali vengono scelti con un certo grado di casualità. Infatti, questo implica che ripetendo l'esercizio diverse volte, non si otterrà, necessariamente, lo stesso valore del punto di minimo.

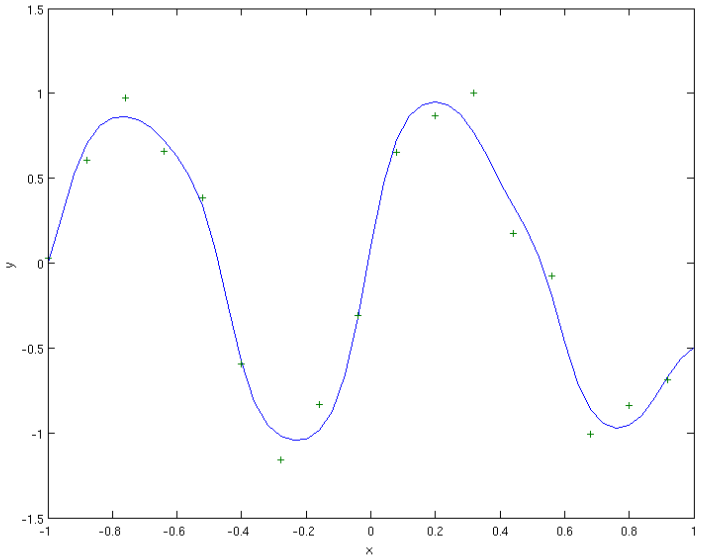

Figura 16: Uscita della rete avente 7 unità nello strato nascosto

Esempio 2

Vediamo ora un problema di classificazione sfruttando uno dei dataset che Matlab mette a disposizione. Questo dataset riguarda 3 diversi produttori di vino e può essere utilizzato per creare una rete neurale che riconosca da quale di questi 3 produttori provenga il vino basandosi su un insieme di 13 costituenti, trovati tramite un'analisi chimica.

Per questo esempio usiamo nuovamente una rete a due strati, avente però funzioni di attivazione sigmoidali in ogni strato. Carichiamo il dataset con la seguente istruzione

$load\ wine\_dataset;$

Nello stesso modo dell'esempio precedente, dividiamo l'insieme dei dati in due sottoinsiemi, creiamo la rete ed impostiamo la funzione di pre e post elaborazione

$indTrain = 1 : 3 : length(x);$
$indTest = 2 : 3 : length(x);$

$xTrain = x(indTrain)$; - ingressi di addestramento
$xTest = x(indTest)$; - ingressi di prova
$tTrain = t(indTrain)$; - target di addestramento
$tTest = t(indTest)$; - target di prova
$net = fitnet(hiddenLayerSize)$; - crea la rete
$net.divideFcn = ''$; - disattiva il partizionamento automatico dei dati
$net.inputs1.processFcns = 'mapminmax'$;
$net.outputs2.processFcns = 'mapminmax'$;

Infine, impostiamo i parametri di addestramento

$net.performFcn = 'sse'$; - somma dei quadrati residui
$net.performParam.squaredWeighting = false$; - no somma pesata
$net.trainFcn = 'traingd'$; - discesa gradiente (eq. (31))
$net.trainParam.lr = 0.01$; - tasso di addestramento
$net.trainParam.epochs = 1000$; - numero di cicli di ricerca del minimo
$net.trainParam.min\_grad = 0$;

Notiamo che, a differenza del primo esempio, abbiamo usato la discesa del gradiente semplice ed un numero di cicli di ricerca pari a 1000. Gli altri parametri sono gli stessi. Addestrando a questo punto la rete con le seguenti istruzioni

$[net, tr] = train(net, xTrain, tTrain)$;
$Etrain = perform(net, tTrain, yTrain)$;
$Etest = perform(net, tTest, yTest)$;

e ripetendo il processo per diversi numeri di unità nascoste otteniamo il risultato di figura 17.

Dalla figura 17 osserviamo che la funzione di errore rispetto ai dati di prova assume il valore minimo quando lo strato nascosto ha 19 unità nascoste. In questo caso la rete classifica correttamente tutti i dati dell'insieme di prova (59/59), mentre classifica correttamente il 98.9% di dati dell'intero insieme (176/178).

Osserviamo inoltre che nel grafico di figura 17 non si notano segni di overfitting. L'overfitting è, infatti, un fenomeno che si può presentare solo

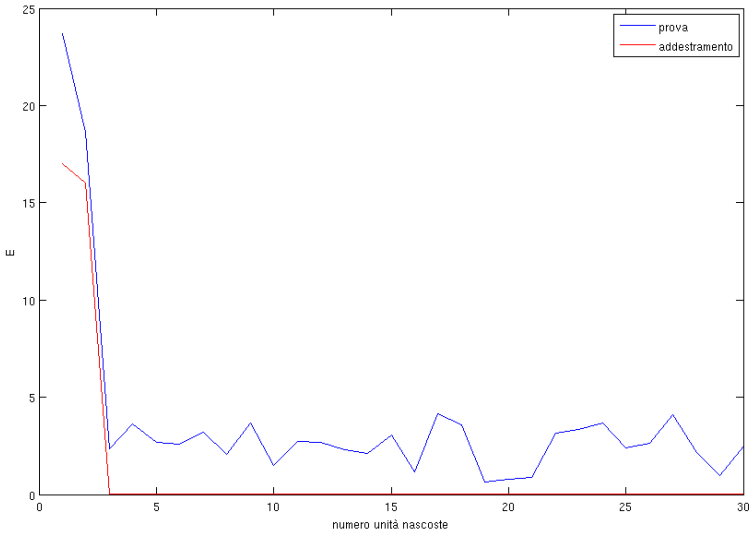

Figura 17: Funzioni di errore rispetto agli insiemi di addestramento e di prova

quando il numero di parametri della rete si avvicina al numero di dati di addestramento.

Notiamo infine che anche in questo esempio valgono le stesse considerazioni fatte, riguardo i valori iniziali dei pesi e di bias, dell'esempio precedente.

Riferimenti bibliografici

[1] Chris M. Bishop, *Neural networks and their applications*, Review of Scientific Instruments / Volume 65 / Issue 6 , (1994).

[2] David J.C. MacKay, *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press 2003, pp. 467-482, (2005).

[3] *http://archive.ics.uci.edu/ml/datasets.html*, Machine Learning Repository.