Contenuti del corso di Machine Learning ed Applicazioni – Mod.B

a.a. 2014/2015

- Richiami di concetti di base per i problemi di Classificazione e regressione (Bishop, 1.1,1.2,1.3,1.4,1.5, 1.8, 1.9)
- Reti neurali a singolo strato (Bishop, 3.1, 3.2, 3.3, 3.4)
- Reti neurali multi strato (Bishop, 4.1, 4.3, 4.8)
- Funzioni a base radiale (Bishop, 5.1, 5.2, 5.3, 5.6 (facoltativo), 5.7 (facoltativo), 5.8)
- Funzioni di errore (Bishop, 6.1.1, 6.1.2, 6.1.3, 6.7 (facoltativo))
- Discesa del gradiente e sue varianti (Bishop, 7.5,7.5.2,7.5.3)
- Regolarizzazione (9.2,9.2.1,9.2.3,9.2.4)
- PCA e reti autoassociative (Bishop, 8.6)

Machine Learning ed Applicazioni – Mod. B

a.a. 2014/2015

Prof. R.Prevete

Introduzione

Alcune delle principali problematiche che possono essere affrontate utilizzando tecniche di Machine Learning sono le seguenti:

- *classificazione/trasformazione di pattern*; Ad esempio riconoscimento di immagini, suoni, voci ed altri pattern, estrazione di caratteristiche significative di un certo fenomeno;
- *ottimizzazione*; Ad esempio, un tipico problema di ottimizzazione é quello del "commesso viaggiatore": dato un certo numero di città, trovare il percorso più breve per visitarle tutte.
- *previsione*; rientrano in questo caso, ad esempio, i problemi di previsione meteorologica.
- *controllo*; ad esempio per il controllo senso-motorio di un robot mobile.

Tra questi possiamo concentrare la nostra attenazione su:

- Problemi di *Classificazione*
- Problemi di *Clusterizzazione*
- Problemi di *Regressione*

Cerchiamo di capire meglio cosa sono tali problemi.

Problemi di Classificazione

Classificare è un'attività fondamentale svolta da tutti gli esseri viventi. Tramite i nostri sensi abbiamo esperienza di un oggetto e siamo subito in grado di classificarlo in una particolare categoria. Tutti riconosceranno di aver classificato l'oggetto delle loro esperienze come: una sedia, un tavolo, un animale, una pianta, commestibile, velenoso, profumato, amico, nemico, alto, basso, ecc. E' importate osservare che questa capacità di classificare nella giusta categoria un "oggetto" passa attraverso una fase preliminare di "esperienza con oggetti simili".
In maniera analoga, è possibile pensare ad un algoritmo in grado, tramite l'osservazione di un insieme di casi, di classificare un certo evento nella sua categoria.

Facciamo un esempio semplice. Supponiamo che abbiamo un cesto pieno di palline di colore diverso e vogliamo distinguere le palline di colore rosso da tutte le altre. Quindi noi sappiamo, a priori, che il nostro insieme di palline si suddivide in due classi:

PALLINE ROSSE,
PALLINE NON ROSSE

Il problema è quello di definire in maniera operativa un modo automatico (o semi-automatico) per assegnare ciascuna pallina alla classe PALLINE ROSSE o alla classe PALLINE NON ROSSE. Ad esempio, possiamo definire un primo algoritmo (semplice) che possa risolvere il problema:

IF R[x] > G[x] AND R[x] > B[x] THEN

classe ←PALLINE ROSSE

ELSE classe ← PALLINE NON ROSSE

Dove x è un array di dimensione 3 contente i valori R, G e B del colore di una pallina.

Osserviamo come la classificazione di un oggetto avviene sempre in base a degli attributi, proprietà, caratteristiche dell'oggetto stesso che lo accomunano ad altri oggetti.

Le classi si possono considerare come "etichette" (spesso mutuamente esclusive) che si pongono su un oggetto, che in generale può essere rappresentato da un vettore contenente i suoi attributi significativi.

Nell'esempio precedente, quindi, ogni pallina è rappresentato da un vettore **x**=[R,G, B] costituito da tre valori (caratteristiche dell'oggetto).

Allora un problema di classificazione può, in maniera molto generale, essere caratterizzato nella seguente maniera:

- Un insieme U di elementi, che costituiscono l'*Universo*, tale che ogni suo elemento $u \in U$ appartenga ad una tra m classi C_k, con k=1,2, ...,m, che rappresentano una partizione di U.
- Un *processo automatico* che associa ciascun elemento di U ad un *insieme* di valori. Possiamo dire che esiste una funzione F: $u \in U \rightarrow \mathbf{x}^u = F(u) \in D \subseteq R^d$. Cioè ogni elemento u di U è rappresentato da un unico elemento $\mathbf{x}^u=(x^u_1,x^u_2,...,x^u_d)$ di D, dove D è, in genere, un sottoinsieme d-dimensionale di R^d.
- Un *processo automatico* (un algoritmo) di classificazione che sia capace di assegnare ogni $u \in U$, rappresentato da $\mathbf{x}^u=(x^u_1,x^u_2,...,x^u_d)$ (*vettore di caratteristiche*), ad una ed, in genere, una sola classe C_k.
- Una "misura di bontà" del risultato del processo di classificazione.

Naturalmente una qualsiasi processo che faccia questo è sempre pensabile, il problema è, però, se tale processo è un "*buon"* algoritmo oppure no.

Alcune proprietà di un "*buon*" algoritmo di classificazione:

Classificazione accurata: classificare gli elementi in maniera accurata, anche in presenza di rumore o di dati mancanti.

Ampio dominio di applicabilità. Il classificatore deve essere adatto per classificare qualunque oggetto di qualunque dominio. Questo implica che sia in grado di trattare ugualmente bene attributi di tipo diverso (reali, interi, stringhe, date, ecc.)

In teoria, il miglior algoritmo di classificazione dovrevebbe esere capace di classificare ogni $u \in C_k$ a C_k stesso.

Una possibile "misura di bontà" del risultato del processo di classificazione è dato da due valori: *recall* e *precision*.

Recall e Precision

Cerchiamo di precisare cosa intendiamo per recall e precision. Supponiamo che l'insieme U sia suddiviso in due classi C e U-C. Una volta che abbiamo realizzato il processo di classificazione per N elementi di U, cioè su un sottoinsieme $U' \subseteq U$, di cui conosciamo l'effettiva classificazione, definiamo:

- *True Positive* (TP)= il numero di elementi che sono stati classificati come appartenenti a C e che effettivamente appartengono a C.
- *False Positive* (FP)= il numero di elementi che sono stati classificati come appartenenti a C ma che, in realtà, appartengono a U-C.
- *True Negative* (TN)= il numero di elementi che sono stati classificati come appartenenti a U-C e che effettivamente appartengono a U-C.
- *False Negative* (FN)= il numero di elementi che sono stati classificati come appartenenti a U-C ma che, in realtà, appartengono a C.

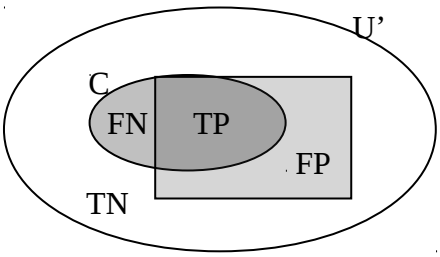
Naturalmente deve essere TP+FP+TN+FN=N. Osserviamo che il numero di elementi che il processo di classificazione a classificato come appartenenti a C è pari a TP+FP.

Allora definiamo (vedi fig):

- Recall = TP/(TP+FN)
- Precision= TP/(TP+FP)

La situazione ideale è quella per cui si ha Recall e Precision pari a 1. Recall pari a 1 significa che tutti gli elementi che appartenevano alla classe C e che sono stati dati in input al processo di classificazione sono stati effettivamente classificati come appartenenti alla classe C. Mentre Precision pari a 1 significa che tutti gli elementi che appartenevano alla classe U-C e che sono stati dati in input al processo di classificazione sono stati effettivamente classificati come appartenenti alla classe U-C.

Naturalmente avere solo uno dei due pari a 1 (Recall o Precision) è facile, ad esempio classificare sempre un elemento come appartenente alla classe C comporta una Recall pari a 1, il difficile è avere entrambi i valori alti.



Recall = TP/(TP+FN)

Precision = TP/(TP+FP)

Figura 1: *Recall e Precision*. L'ovale bianco corrisponde all'insieme U' sottoposto al processo di classificazione. L'ovale grigio rappresenta gli elementi di U' che appartengono alla classe C, i restanti elementi non appartengono alla classe C. Il rettangolo grigio rappresenta gli elementi che un processo di classificazione assegna alla classe C.

Tipi diversi di approcci

Abbiamo visto che, allora, un problema di classificazione può essere suddiviso in due fasi differenti:

a) Dai dati possiamo apprendere un modello per $P(C_k|x)$

b) dato tale modello, c'è un passo di decisione per la scelta della classe

L'insieme dei due passaggi determina le regioni di decisione (e quindi le funzioni discriminanti) di cui abbiamo parlato in precedenza.

Una alternativa è risolvere entrambi i problemi insieme e quindi avere direttamete un mapping da x alla classe.

In effetti possono essere isolati tre differenti approcci nel risolvere un problema di classificazione:

(a) First solve the inference problem of determining the class-conditional densities $p(x| C_k)$ for each class C_k individually. Also separately infer the prior class probabilities $p(C_k)$. Then use Bayes' theorem in the form

$P(C_k|x) = p(x| C_k)P(C_k) /p(x)$

to find the posterior class probabilities $P(C_k|x)$. As usual, the denominator in Bayes' theorem can be found in terms of the quantities appearing in the numerator.

Equivalently, we can model the joint distribution $p(x, C_k)$ directly and then normalize to obtain the posterior probabilities. Having found the posterior probabilities, we use decision theory to determine class membership for each new input x.

Approaches that explicitly or implicitly model the distribution of inputs as well as outputs are known as **generative models**, because by sampling from them it is possible to generate synthetic data points in the input space.

(b) First solve the inference problem of determining the posterior class probabilities $P(C_k|x)$, and then subsequently use decision theory to assign each new x to one of the classes.

Approaches that model the posterior probabilities directly are called **discriminative models**.

(c) Find a function f (x), called a discriminant function, which maps each input x directly onto a class label. For instance, in the case of two-class problems, f (·) might be binary valued and such that f = 0 represents class C_1 and f = 1 represents class C_2 . In this case, probabilities play no role.

Let us consider the relative merits of these three alternatives. Approach (a) is the most demanding because it involves finding the joint distribution over both x and C_k .

For many applications, x will have high dimensionality, and consequently we may need a large training set in order to be able to determine the class-conditional densities to reasonable accuracy. Note that the class priors $p(C_k)$ can often be estimated simply from the fractions of the training set data points in each of the classes.

One advantage of approach (a), however, is that it also allows the marginal density of data p(x) to be determined from (1.83). This can be useful for detecting new data points that have low probability under the model and for which the predictions may be of low accuracy, which is known as outlier detection or novelty detection (Bishop, 1994; Tarassenko, 1995).

However, if we only wish to make classification decisions, then it can be wasteful of computational resources, and excessively demanding of data, to find the joint distribution $p(x, C_k)$ when in fact we only really need the posterior probabilities $P(C_k|x)$, which can be obtained directly through approach (b). Indeed, the class-conditional densities may contain a lot of structure that has little effect on the posterior probabilities, as illustrated in Figure 1.27. There has been much interest in

exploring the relative merits of **generative** and **discriminative** approaches to machine learning, and in finding ways to combine them (Jebara, 2004; Lasserre et al., 2006).

An even simpler approach is (c) in which we use the training data to find a discriminant function f(x) that maps each x directly onto a class label, thereby combining the inference and decision stages into a single learning problem.

With option (c), however, we no longer have access to the posterior probabilities $p(C_k|x)$. There are many powerful reasons for wanting to compute the posterior:

Minimizing risk. If we know the posterior probabilities, we can trivially revise the minimum risk decision criterion appropriately.

Reject option. Posterior probabilities allow us to determine a rejection criterion that will minimize the misclassification rate, or more generally the expected loss, for a given fraction of rejected data points.

Compensating for class priors. Consider a medical X-ray problem, and suppose that we have collected a large number of X-ray images from the general population for use as training data in order to build an automated screening system. Because cancer is rare amongst the general population, we might find that, say, only 1 in every 1,000 examples corresponds to the presence of cancer. If we used such a data set to train an adaptive model, we could run into severe difficulties due to the small proportion of the cancer class. For instance, a classifier that assigned every point to the normal class would already achieve 99.9% accuracy and it would be difficult to avoid this trivial solution. Also, even a large data set will contain very few examples of X-ray images corresponding to cancer, and so the learning algorithm will not be exposed to a broad range of examples of such images and hence is not likely to generalize well. A balanced data set in which we have selected equal numbers of examples from each of the classes would allow us to find a more accurate model.

However, we then have to compensate for the effects of our modifications to the training data. Suppose we have used such a modified data set and found models for the posterior probabilities. From Bayes' theorem , we see that the posterior probabilities are proportional to the prior probabilities, which we can interpret as the fractions of points in each class. We can therefore simply take the posterior probabilities obtained from our artificially balanced data set and first divide by the class fractions in that data set and then multiply by the class fractions in the population to which we wish to apply the model. Finally, we need to normalize to ensure that the new posterior probabilities sum to one. Note that this procedure cannot be applied if we have learned a discriminant function directly instead of determining posterior probabilities.

Problemi di Clusterizzazione

Un problema di *clusterizzazione (segmentazione)* ha molte cose in comune con un problema di classificazione ma, potremmo affermare, che sussiste in una fase precedente. In questo caso, infatti, l'insieme U di elementi che rappresenta il nostro universo non è a priori suddiviso in classi ma si vuole proprio determinare se tale insieme è suddivisibile in classi in base a qualche "misura di distanza" tra gli elementi dell'insieme U.

Anche in questo caso, in generale, possiamo dire che un problema di segmentazione è caratterizzato da:

- Un insieme U di elementi, che costituiscono l'*Universo*.
- Un *processo automatico* che associa ciascun elemento di U ad un *insieme* di valori. Possiamo dire che esiste una funzione F: $u \in U \rightarrow \mathbf{x}^u = F(u) \in D \subseteq R^d$. Cioè ogni elemento u di U è rappresentato da un unico elemento $\mathbf{x}^u=(x^u_1,x^u_2,...,x^u_d)$ di D, dove D è, in genere, un sottoinsieme d-dimensionale di R^d.
- Una *misura di somiglianza* tra due elementi di U. Cioè è definita una funzione s: $x,y \in D \rightarrow s(x,y) \in R$ in modo tale che la distanza tra $u,v \in U$ è definita pari a s(F(u),F(v)).
- Un criterio per raggruppare gli elementi di U in differenti classi in base alla misura di somiglianza.

Problemi di Regressione

Nel caso dei problemi di regressione abbiamo, in generale, un insieme di n valori ordinati $x_1,x_2, \ldots,x_n$ a cui sono associati n valori $y_1,y_2, \ldots,y_n$ tramite una qualche funzione sconosciuta f, cioè $y_1=f(x_1)$, $y_2=f(x_2)$, …, $y_n=f(x_n)$ e si vuole conoscere per il prossimo valore x_{n+1} quale sia il valore associato y_{n+1}. Fondamentalmente, allora, abbiamo il problema di approssimare la funzione f sulla base dei valori $x_1,x_2, \ldots,x_n$ e $y_1,y_2, \ldots,y_n$.

Per tutti e tre i problemi esposti si riducano (o sono già di per sé) a problemi di approssimazione di funzioni. Nel prosieguo, quindi, focalizzeremo la nostra attenzione sui problemi di classificazione.

Problema di classificazione come problema di approssimazione di funzione.

Abbiamo detto che un problema di classificazione può essere visto come il trovare un algoritmo per assegnare ciascun punto dello spazio delle caratteristiche ad una di c classi. Possiamo allora supporre che lo spazio delle caratteristiche sia suddiviso in c *regioni* $R_1,R_2,...R_c$, dette *regioni di decisioni* , in modo tale che se il punto cade in R_k allora appartiene a C_k. Osserviamo che ciascuna R_k può essere composta di varie regioni disgiunte (vedi figura 2). I confini di tali regioni sono detti *confini di decisione* o *superfici di decisione*.

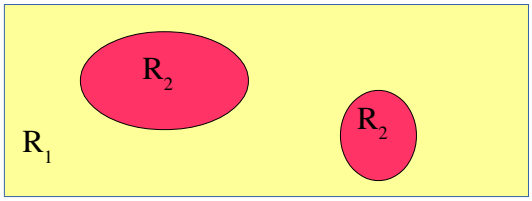


Figura 2: Esempio di un problema a due classi C_1 e C_2. In giallo è raffigurata la regione R_1 corrispondente alla classe C_1, in rosso la regione R_2 corrispondente alla classe C_2.

Un possibile criterio per determinare le regioni di decisione è quello di minimizzare l'errore di *errata classificazione*, cioè l'errore di assegnare x a C_k quando invece questo appartiene a C_m con m≠k. Cominciamo con il considerare un problema a due classi C_1 e C_2. L'errore, allora, sarà dato da:

$$P(\text{error}) = P(x\in R_2, C_1) + P(x\in R_1, C_2) = P(x\in R_2/C_1)P(C_1) + P(x\in R_1/C_2)P(C_2) =$$
$$\int_{R2} p(x/C_1)P(C_1)dx + \int_{R1} p(x/C_2)P(C_2)dx$$

dove $p(x/C_m)$ è la funzione di densità di probabilità di x con la condizione che x è calcolato da elementi appartenenti alla classe C_m, con m=1,2.

Allora, dato un certo x, se risulta $p(x/C_1)P(C_1) > p(x/C_2)P(C_2)$ al fine di minimizzare la probabilità di errore scegliamo R1 e R2 in modo che x appartenga a R1 (in questo modo nel calcolo di P(error) viene escluso il contributo di $p(x/C_1)P(C_1)$).

Se abbiamo c classi, allora, possiamo estendere il ragionamento fatto nel seguente modo:

$$P(\text{error}) = P(x\in R_1, C_2) + P(x\in R_1, C_3) + \ldots + P(x\in R_1, C_c) +$$
$$P(x\in R_2, C_1) + P(x\in R_1, C_3) + \ldots + P(x\in R_1, C_c) + \ldots$$

$$P(x\in R_c, C_1)+ P(x\in R_1, C_2)+\ldots+ P(x\in R_1, C_{c-1})$$

$P(error)= \sum_k\sum_{m\neq k} P(x\in R_k,C_m)= \sum_k\sum_{m\neq k} P(x\in R_k/C_m)P(C_m)= \sum_k\sum_{m\neq k} \int_{Rk} p(x/C_m)P(C_m)dx$

dove $p(x/C_m)$ è la funzione di densità di probabilità di x con la condizione che x è calcolato da elementi appartenenti alla classe C_m, con m=1,2, ...c.

Quindi minimizzare l'errore di assegnare x a C_k quando invece appartiene a C_m con $m\neq k$ significa che se, per un dato x, $p(x/C_k)P(C_k) > p(x/C_m)P(C_m)$ per ogni $m\neq k$ dovremmo scegliere le regioni $\{R_i\}$ in modo che x appartenga a R_k, infatti in tal modo $p(x/C_k)P(C_k)$ non darà alcun contributo nel calcolo dell'errore errore (osserviamo che comunque facciamo una scelta possiamo escludere solo un termine, quindi evidentemente conviene ecludere il termine che assume il valore maggiore).

In maniera equivalente se calcoliamo la probabilità di avere una corretta classificazione

$P(correct)= \sum_k P(x\in R_k,C_k)= \sum_k P(x\in R_k/C_k)P(C_k)= \sum_k \int_{Rk} p(x/C_k)P(C_k)dx$

Per massimizzare tale errore dobbiamo scegliere le regioni $\{R_i\}$ in modo che x è assegnato alla regione per la quale l'integrando è massimizzato, quindi anche in questo caso se , per un dato x, $p(x/C_k)P(C_k) > p(x/C_m)P(C_m)$ per ogni $m\neq k$ dovremmo scegliere le regioni $\{R_i\}$ in modo che x appartenga a R_k.

Allora una possibile regola di classificazione la seguente

Se $P(C_k /x) > P(C_m /x)$ per ogni $m\neq k$ allora l'elemento rappresentato da x appartiene a C_k.

Infatti $P(C_k /x) > P(C_m /x) \Leftrightarrow$ (dal teorema di Bayes)

$p(x/C_k)P(C_k)/p(x) > p(x/C_m)P(C_m)/p(x) \Leftrightarrow$

$p(x/C_k)P(C_k) > p(x/C_m)P(C_m)$

Quindi la regola precedente è equivalente a dire che

Se $p(x/C_k)P(C_k) > p(x/C_m)P(C_m)$ per ogni $m\neq k$ allora l'elemento rappresentato da x appartiene a C_k.

e, quindi, a *minimizzare l'errore di errata classificazione* e, equivalentemente, *massimizare la probabilità di corretta classificazione*.

Teorema di Bayes

Sia p(x) la densità di probabilità di una variabile x il cui valore rappresenta gli elementi di un insieme S. Siano C_k con k=1,2,...,c le classi a cui tali elementi possono appartenere. Sia $P(C_k)$ la probabilità di occorrere della generica classe C_k (la probabilità a priori di C_k), $P(C_k/x)$ la probabilità di C_k dato x (probabilità a posteriori di C_k) e $p(x/C_k)$ la densità di probabilità di x dato C_k.

Possiamo scrivere:

$P(C_k,x)=p(x)\ P(C_k/x)$

$P(C_k,x)=p(x/C_k)\ P(C_k)$

$p(x)\ P(C_k/x) = p(x/C_k)\ P(C_k) \Rightarrow \quad P(C_k/x) = p(x/C_k)\ P(C_k) / p(x)$ (**teorema di Bayes**)

supposto che sia $p(x) = \sum_k p(x/C_k)\ P(C_k)$

Quest'ultima condizione assicura che $\sum_k P(C_k/x) = \sum_k [p(x/C_k)\ P(C_k) / \sum_k p(x/C_k)\ P(C_k)]=1$

Cioè, dato x la probabilità di appartenere ad una qualunque delle classi è 1

L'importanza del teorema di Bayes è dovuta al fatto che la probabilità a posteriori di C_k è espressa in termini di quantità che sono in genere più facili da calcolare.

Osservazione.

La funzione densità di probabilità p(x) specifica che la probabilità che la variabile x assuma valori nell'intervallo [a,b] è data da:

$P(x\in [a,b])=\int_a^b p(x)dx$ con $P(x\in D)=\int_D p(x)=1$ se D corrisponde all'intero insieme di appartenenza di x.

Se ci sono d variabili $x_1,x_2,...,x_d$ possiamo considerare il vettore $x=(x_1,x_2,...,x_d)$ corrispondente ad un punto in uno spazio d-dimensionale. La distribuzione dei valori di x può essere descritta dalla funzione densità di probabilità p(x) tale che la probabilità che x cada in una regione R dello spazio di x è data da:

$P(x\in R)=\int_R p(x)dx$

Funzioni discriminanti

In generale possiamo riformulare un problema di classificazione con c classi in termini di un insieme di c funzioni discriminanti: $y_1(x)$, $y_2(x)$, ...,$y_c(x)$ in modo tale che un elemento rappresentato da x è assegnato alla classe C_k se

$y_k(x) > y_m(x)\ \forall\ m\neq k$

Se scegliamo come funzioni discriminanti $y_k(x) = P(C_k/x)$ (o equivalentemente $y_k(x) = p(x/C_k)P(C_k)$) abbiamo una regola che minimizza la probabilità di errore di cattiva classificazione.

Osserviamo che:

- I confini di decisione sono dati dalle regioni dove le funzioni discriminanti sono uguali, così che se R_k e R_m sono contigue poi il confine di decisione è dato da $y_k(x) = y_m(x)$
- Possiamo sempre trasformare un funzione discriminante tramite funzioni monotoniche ottenendo ancora funzioni discriminanti.
- Funzioni discriminanti per un problema a due Classi sono tradizionalmente riscritte in un forma più semplice. In questo caso, invece di usare due funzioni discriminanti $y_1(x)$ e $y_2(x)$, possiamo usare una unica funzione discriminante $y(x)=y_1(x) - y_2(x)$ e la regola di classisicazione *if* $y_1(x) > y_2(x)$ *allora* C_1, *altrimenti* C_2 si trasforma in *se* $y(x)>0$ *alllora* C_1 *altrimenti* C_2.

Riassumendo

Un problema di classificazione può essere visto in termini di un insieme finito di *funzioni discriminanti*. Tali funzioni discriminanti possono assumere forme diverse, delimitando dei confini di decisione. Lo scopo, allora, è trovare un algoritmo che permette di approssimare tali funzioni.

Reti neurali per problemi di classificazione

Nelle pagine precedenti abbiamo introdotto il concetto di funzioni discriminanti.

Se abbiamo c classi C_1, ...,C_c con gli elementi da classificare rappresentati da un vettore di caratteristiche **x** di dimensione d, allora, vogliamo fissare c funzioni discriminati $f_1(\mathbf{x}),f_2(\mathbf{x}), \ldots,f_c(\mathbf{x})$ (ad esempio le probabilità a-posteriori in $P(C_1/\mathbf{x})$, $P(C_2/\mathbf{x})$, ...,$P(C_c/\mathbf{x})$) in modo tale che l'algoritmo di classificazione è il seguente:

Se $f_k(\mathbf{x})>f_m(\mathbf{x})$ $\forall m\neq k$ allora l'elemento rappresentato da **x** appartiene alla classe C_k.

Tale regola, ricordo, permette di rendere minimo l'errore di classificare un elemento che appartiene a C_m in C_k , con $m\neq k$, nel caso in cui le funzioni discriminanti corrispondano alle probabilità $P(C_i/\mathbf{x})$.

Vogliamo, quindi, risolvere il problema di classificazione fissando una funzione f: $\mathbf{x}\in R^d \rightarrow f(\mathbf{x}) \in R^c$. **In generale, però, tale funzione non è conosciuta** o è solo parzialmente conosciuta (ad esempio si può conoscere il "tipo" di funzione, cioè il tipo di regioni di decisione – semipiano, regione connessa, etc… - ma non l'esatta forma).

Un problema di classificazione, allora, si riduce ad un problema di *approssimazione di funzione*.

Si tratta, cioè, di approssimare la funzione f: $\mathbf{x}\in R^d \rightarrow f(\mathbf{x}) \in R^c$ tramite una funzione parametrizzata, g : $\mathbf{w}\in R^p$, $\mathbf{x}\in R^d \rightarrow g(\mathbf{x};\mathbf{w}) \in R^c$ dove **w** è un vettore di p parametri.

Vogliamo, quindi, modellare la funzione f(**x**) tramite g(**x;w**).

A tale scopo è necessario:

1. Verificare che effettivamente g(**x;w**) abbia la capacità di rappresentare f(**x**) .
2. Una volta che si è appurato che g(**x;w**) ha la capacità di rappresentare f(**x**) è necessario trovare un procedimento che ci permetta di trovare il "corretto" valore **w= w*** per approssimare f(**x**). A tale scopo, in genere, abbiamo un insieme di *esempio*, TS, di coppie di valori $(\mathbf{x}^1,f(\mathbf{x}^1)=\mathbf{t}^1)$, $(\mathbf{x}^2,f(\mathbf{x}^2)=\mathbf{t}^2)$, ..., $(\mathbf{x}^n,f(\mathbf{x}^n)=\mathbf{t}^n)$ e la determinazione di **w** corrisponde a determinare il valore **w= w* tale che** $g(\mathbf{x}^1,w^*)=\mathbf{t}^1$, $g(\mathbf{x}^1,w^*)=\mathbf{t}^1$,…, $g(\mathbf{x}^n,w^*)=\mathbf{t}^n$. Il processo tramite il quale sono determinati i parametri **w*** è detto *learning* o *training*, e, per tale motivo, TS è detto *training set*.

Per risolvere un problema di classificazione tramite reti neurali dobbiamo, allora:

1. verificare che esistono modelli di reti neurali che realizzano funzioni g(**x;w**) da uno spazio d-dimensionale a uno spazio c-dimensionale.

2. capire quale è la capacità di rappresentazione delle funzioni realizzate da tali modelli.
3. Determinare un processo tramite il quale sono determinati i parametri della rete.

Elementi caratterizzanti un generico modello di rete neurale

Possiamo caratterizzare molti modelli di rete neurale tramite le seguenti proprietà:

1. Un insieme finito di N elementi, detti nodi, neuroni o unità, *S={1,2, …, N}*.
2. Un insieme finito di connessioni (in genere monodirezionali) *C*, ciascuna che connette un nodo *k* ad un nodo *h*. A ciascuna connessione che va da *k* ad *h* è associata un peso w_{hk} .
3. Un insieme finito di d elementi, detti **input della rete (**da cui, in genere, possono solo partire connessioni).
4. Ciascun nodo *h* è caratterizzato da:
   4.1. un valore di input i_h, un valore di attivazione a_h, un valore di output z_h e una valore di bias (o soglia) b_h.
   4.2. una funzione di input s_h, una funzione di attivazione g_h e una funzione di output f_h.
5. Detto $\mathbf{w}_h$ il vettore dei pesi delle connessioni che arrivano su *h* e **o** il vettore degli output dei nodi *k* che hanno connessioni che partono da *k* e arrivano su *h,* l'input input i_h è pari a $i_h = s_h(\mathbf{w}_h, \mathbf{o})$. Normalmente si ha: $i_h = \Sigma_k w_{hk} * o_k$
6. Ciascun a_h è pari a $a_h = g_h (i_h, b_h))$
7. Ciascun o_h è pari a $a_h = f_h (a_h, b_h)$
8. Una sequenza di attivazione dei nodi, cioè l'ordine tramite il quale sono calcolati i valori di output dei nodi. Sequenze di attivazione possibili possono essere *sincrone* o *asincrone*.

Un sottoinsieme dei nodi di S, in genere, sono marcati come **nodi di output e i restanti nodi sono detti nodi interni o nascosti (hidden)**.

Reti Feed-forward

Una modello classica di reti neurali sono le reti neurali **feed-forward**. Queste hanno la proprietà di

- *Non avere cicli*. Possiamo dire che una rete neurale è feed-forward se gode della seguente proprietà: E' possibile associare numeri successivi agli input e a tutti i nodi tale che ciascuna

nodo possa riceve connessioni solamente dagli input o da un nodo avente un numero associato più piccolo**.**

- La sequenza di attivazione è asincrona e segue l'ordine topologico stabilito dalle connessioni.

Allora, l'output di un nodo h di una rete feed-forward e dato da:

$z_h = f_h (\Sigma_k w_{hk} * z_k + b_h)$ (In questo caso abbiamo supposto che la funzione di attivazione sia pari all'identità)

dove la somma va su tutti gli input e i nodi k che inviano connessioni ad h.

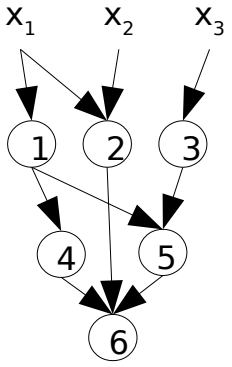


Figura 3: Esempio di rete neurale Feed-Forward

Forward-propagation

Per un dato insieme di valori dell'input della rete, successive applicazioni di $z_h = f_h (\Sigma_k w_{hk} * z_k + b_h)$ permettono l'attivazione di tutti i nodi della rete, inclusi i nodi di output. Questo processo puo' essere visto come una propagazione in avanti (**forward propagation**) di segnali attraverso la rete (secondo l'ordine topologico stabilito dalle connessioni).
Osserviamo che il parametro di bias può essere interpretato come se ci fosse un ulteriore nodo, sempre con valore di output pari a 1, che invia una connessioni ai nodi h, quindi possiamo scrivere:

$z_h = f_h (\Sigma_k w_{hk} * z_k)$, dove la somma, stavolta, è anche estesa al parametro di bias ponendo $w_{h0}=b_h$.

Tali reti, allora, hanno la proprietà che **le attivazioni dei nodi di output possono essere espresse come funzioni deterministiche degli input e così l'intera rete rappresenta un mapping non**

lineare tra uno spazio d-dimensionale ad uno spazio c-dimensionale con parametri dati da i pesi della rete.

Reti neurali a strati

Nella classe di reti feed-forward una sottoclasse fondamentale sono le reti **neurali organizzate a strati (o layer**). In questo caso la rete è suddivisa in *r* sottoinsiemi disgiunti l_1, l_2, …l_r (strati, layer) tale che ci sono solamente connessioni che vanno dagli input ai nodi appartenenti a l_1 e connessioni che vanno dai nodi appartenenti a l_j-*1* ai nodi appartenenti a l_j con *j=2,3, …,r*. I nodi appartenenti a l_r sono i nodi di output i restanti nodi sono i nodi interni. Osserviamo che una rete a *r* strati di nodi può essere vista come una rete di *r* strati di pesi. Nella classe delle reti neurali a strati feed-forward è possibile ancora individuare le reti **full-connected.** In questo caso ciascun neurone di uno strato invia connessioni a tutti i neuroni dello strato successivo.

Quindi una rete neurale Feed-Forward è esprimibile come una funzione parametrizzata g(**x**;**w**) che ha come dominio uno spazio d-dimensionale , come codominio uno spazio c-dimensionale e come parametri tutti i pesi della rete più i bias dei nodi.

Una osservazione

Notiamo che se le funzioni di output di tutti i nodi interni sono funzioni lineari allora esiste sempre una rete neurale equivalente senza nodi interni.

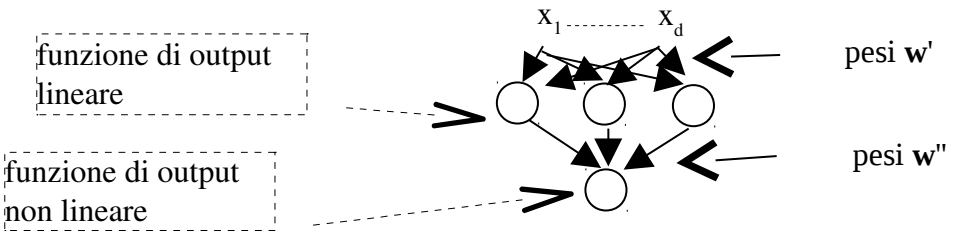


Figura 4: Rete neurale con nodi interni che computano funzioni lineari.

Supponendo, infatti, che la funzione di attivazione sia pari all'identità per tutti i nodi della rete e che tutti i nodi interni abbiano la medesima funzione lineare *f* di output, mentre i nodi di output abbiano come funzione di output la funzione non lineare f_1, possiamo scrivere:

$z_h = f(\Sigma_k w'_{hk} x_k)$ (con f lineare), per i nodi interni

$y_m = f_1(\Sigma_h w''_{mh} z_h) = f_1(\Sigma_h w''_{mh} f(\Sigma_k w'_{hk} x_k)) = f_1(\Sigma_h w''_{mh} \Sigma_k f(w'_{hk}) x_k))$, per i nodi di output

$y_m = f_1(\Sigma_h \Sigma_k w''_{mh} f(w'_{hk}) x_k) = f_1(\Sigma_k x_k * \Sigma_h w''_{mh} f(w'_{hk}))$

$y_m = f_1(\Sigma_k w_{mk} x_k)$

avendo posto $w_{mk} = \Sigma_h w''_{mh} f(w'_{hk})$.

Questo significa che da ora in avanti considereremo sempre reti neurali con nodi interni con funzione di output (o funzione di attivazione) non lineare.

NOTA. Da ora in avanti considereremo sempre o la funzione di output o la funzione di attivazione coincida con la funzione identità.

Cerchiamo ora di capire quale è la capacità rappresentativa delle reti Feed-Forward.

Capacità rappresentativa di una rete neurale feed-forward

Consideriamo, in prima istanza, reti neurali con funzione di attivazione a soglia (funzione di Heaviside)

1 nodo e funzione di attivazione di Heaviside.

Analizziamo, ora, la capacità rappresentativa di una rete neurale feed-forward costituita da un sol nodo e con funzione di attivazione pari alla funzione di Heaviside.
In questo caso abbiamo

$z = \theta(a)$ con $a = \Sigma_k w_k x_k$ dove x_k sono i valori di input più il parametro di bias e $\theta(x)$ funzione di Heaviside.

Osserviamo che in questo caso la rete Feed-Forward si riduce al neurone di MacCulloch & Pitts.

L'output z di un nodo assumerà valori diversi quando a passa da valori negativi a valori positivi, quindi "il confine di decisione" è data da $a=0$, cioè $\Sigma_k w_k x_k = 0$, quindi se abbiamo d input il confine di decisione è un iperpiano d-dimensionale in uno spazio d+1-dimensionale.
Cerchiamo di essere più chiari.

Supponiamo di voler utilizzare tale rete per risolvere un problema di classificazione a due classi C_1 e C_2. In questo caso abbiamo una sola funzione discriminante F(x) e l'algoritmo di classificazione è il seguente:

Se F(x)>0 ➔ elemento rappresentato da x appartiene alla classe C1
altrimenti F(x)<0 ➔ elemento rappresentato da x appartiene alla classe C2

Supponiamo di voler approssimare F(x) tramite la nostra rete $z=y(\boldsymbol{x})=\theta\,(\Sigma_k\, w_k\, x_k)$. Ci dobbiamo chiedere, quali confini di decisione la nostra rete è capace di rappresentare ? Tali confini di decisione permettono di risolvere il problema, cioè $z=\theta\,(\Sigma_k\, w_k\, x_k)$ può approssimare F(x)?
Dato che la risposta del neurone si differenzia quando l'input risulta essere minore o maggiore della soglia, il confine di decisione rappresentabile dalla nostra rete è dato allora dalla equazione
$\Sigma_k\, w_k\, x_k = 0$
Tale equazione corrisponde un (d-1)-iperpiano in uno spazio d-dimensionale
Ad esempio se d=2 abbiamo:
$w_1\, x_1 + w_2\, x_2 + w_0 = 0$
che altro non è che l'equazione di una retta.
Quindi una rete neurale Feed-Forward costituita da un solo neurone e con funzione di attivazione di Heaviside può risolvere solo problemi di classificazione che sono *linearmente separabili*, cioè possiamo rappresentare solo funzioni discriminanti che hanno come confine di decisione un iper-piano.

Una interpretazione geometrica

Osserviamo che l'equazione $\Sigma_k\, w_k\, x_k = 0$ può essere riscritta come $\mathbf{w}^T\mathbf{x}+w_0=0$ dove $\mathbf{w}^T\mathbf{x}$ è il prodotto scalare tra $\mathbf{w}=(w_1,w_2, \ldots,w_d)$ e $\mathbf{x}=(x_1,x_2, \ldots,x_d)$ e w_0 è il bias.

Se $\mathbf{x}^A$ e $\mathbf{x}^B$ sono due punti dell'iperpiano allora
$\mathbf{w}^T\mathbf{x}^A+w_0=0= \mathbf{w}^T\mathbf{x}^B+w_0$
quindi
$\mathbf{w}^T\,\mathbf{x}^A - \mathbf{w}^T\,\mathbf{x}^B= \mathbf{w}^T\,(\mathbf{x}^A - \mathbf{x}^B)=0$
così $\mathbf{w}$ è un vettore perpendicolare ad ogni vettore giacente nell'iperpiano e quindi all'iperpiano stesso. In questo modo $\mathbf{w}$ determina l'orientazione dell'iperpiano.
La distanza dell'iperpiano dall'origine è, inoltre, data da:

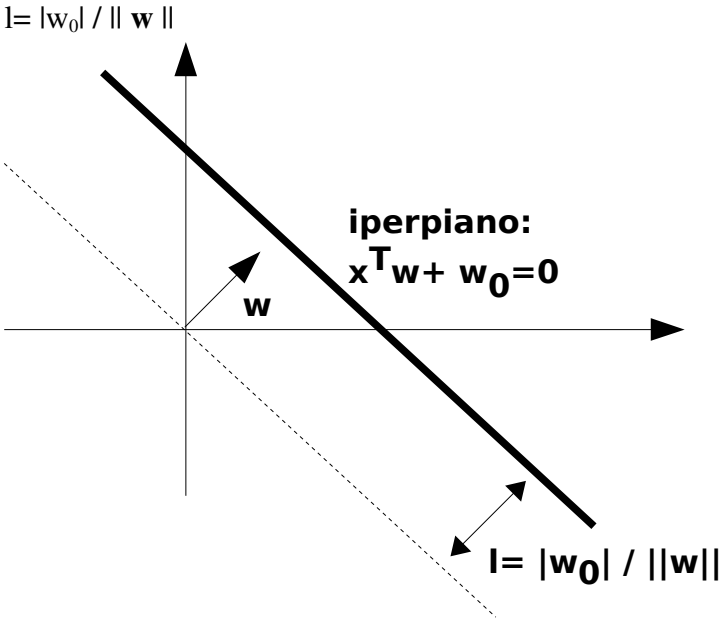


Figura 5: Confine di decisione determinato da una funzione discriminate lineare**.**

Osserviamo ancora che $\mathbf{x}^T\mathbf{w} = ||\mathbf{x}||\,||\mathbf{w}||\cos\alpha$

dove $||\mathbf{x}||$ è la norma (la lunghezza) del vettore **x** , α è l'angolo compreso tra il vettore **x** e il vettore **w**.

Tale prodotto scalare può essere interpretato come la lunghezza del vettore proiezione di **x** lungo la direzione di **w** moltiplicato per la lunghezza di **w**. Se indichiamo con x_w la lunghezza del vettore proiezione di **x** lungo la direzione di **w** possiamo scrivere:

$\mathbf{x}^T\mathbf{w} = x_w\,||\mathbf{w}||$

$x_w = (\mathbf{x}^T\mathbf{w})/||\mathbf{w}||$

Possiamo dire, allora, che:

Fissato il valore di **w** e w_0, per i valori di input tale che $\mathbf{x}^T\mathbf{w} + w_0 = 0$, dato che **w** e w_0 sono costanti, la proiezione $x_w = (\mathbf{x}^T\mathbf{w})/||\mathbf{w}|| = -w_0/||\mathbf{w}||$ è costante. Quindi i valori di **x** che soddisfano $\mathbf{x}^T\mathbf{w} + w_0 = 0$, per dei fissati valori di **w** e w_0, corrispondono ai punti la cui proiezione su **w** da il valore $-w_0/||\mathbf{w}||$, tali punti corrispondono ad un iperpiano perpendicolare a **w** e che lo interseca nel punto $-w_0/||\mathbf{w}||$.

Dato allora un generico vettore **x** ci sono due casi:

1. $x_w > -w_0 / ||\mathbf{w}||$, questo significa che il vettore **x** deve giacere oltre il piano perpendicolare e risulta $\mathbf{x}\cdot\mathbf{w} > -w_0$
2. $x_w < -w_0 / ||\mathbf{w}||$, questo significa che il vettore x deve giacere prima della perpendicolare e risulta $\mathbf{x}\cdot\mathbf{w} < -w_0$

Una digressione storica: il perceptron.

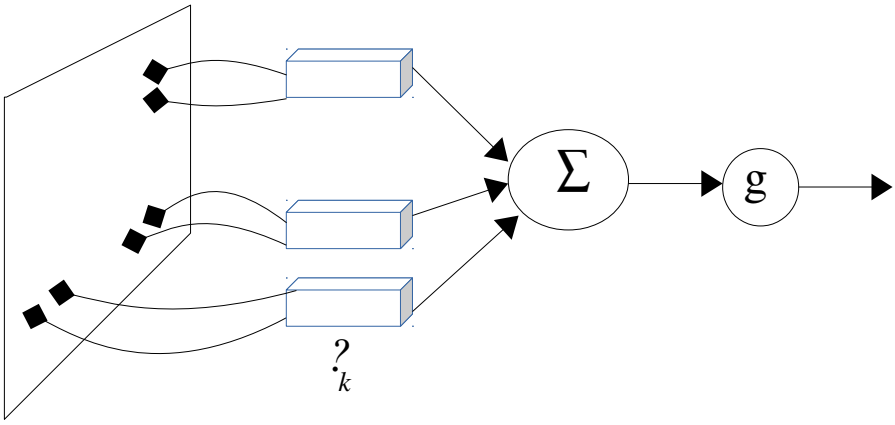


Figura 6:

Reti neurali con un singolo strato di pesi ed un neurone di uscita, con funzione di output a soglia, furono studiate da Rosenblatt (1962). Tali reti furono chiamate *perceptrons*. Queste reti furono a problemi di classificazioni, dove i valori di input erano, in genere, immagini binarie di caratteri o forme semplici. Il valore di output del perceptron è dato da:

$y = g(\Sigma_k w_k \, ?_k(\mathbf{x}))$

dove : **x** è un vettore di 0 ed 1 rappresentate l'immagine binaria, $?_k$ sono funzioni a soglia che dipendono solo da pochi valori dell'immagine, e g è una unzione a soglia che assume valore 0 o 1, cioè g(a)=-1 se a<0, altrimenti g(a)=1 (vedi figura precedente).

In un famoso libro, Perceptrons (1969), Minsky e Papert mostrarono che ci sono molti tipi di problemi di classificazione che un perceptron non può risolvere. Precedentemente abbiamo visto che con un singolo neurone con funzione di output a soglia è possibile risolvere solo problemi linearmente separabili. Il perceptron, grazie alle funzioni $?_k(\mathbf{x})$, potrebbe, però, "mappare" un problema non linearmente separabile in uno linearmente separabile. La critica di Minsky e Papert, perciò, non nasce tanto dal fatto che con un solo neurone risolvo solo problemi linearmente separabili ma dal fatto che le funzioni $?_k(\mathbf{x})$ *sono fissate "a priori" e non possono essere adattate al*

particolare problema (vedremo in seguito che, invece, è possibile considerare una sorta di perceptron in cui anche le funzioni $\varphi_k(\boldsymbol{x})$ sono adattabili, e così superare tale critica). Vediamo, quindi, un tipico problema riportato da Minsky e Papert per cui il perceptron non può essere utilizzato:

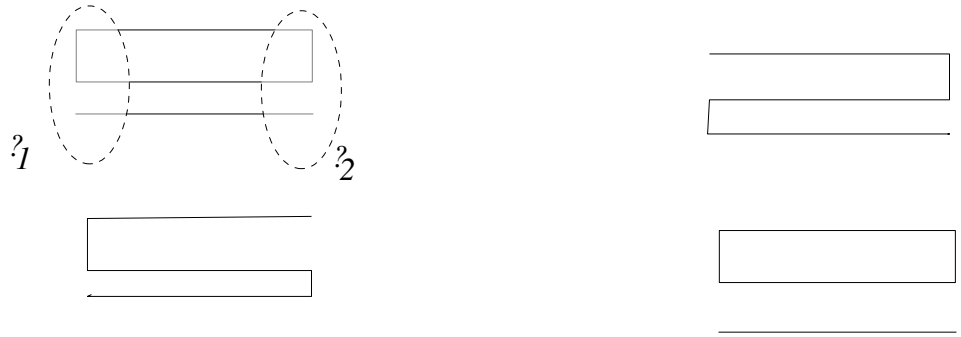


Figura 7: Un problema non risolvibile dal perceptron.

Supponiamo di voler determinare se una immagine geometrica binaria sia semplicemente connessa oppure no, con le funzioni $\varphi_1(\boldsymbol{x})$ e $\varphi_2(\boldsymbol{x})$ come in figura. Supponiamo che le forme connesse debbano essere classificate con 1 , mentre le altre -1. Supponiamo che l'input per le funzioni $\varphi_k(\boldsymbol{x})$ sia piccolo rispetto alle immagini e fissato, come in figura. Allora per la prima immagine (in alto a sinistra) la somma pesata delle funzioni $\varphi_k(\boldsymbol{x})$ deve essere minore di zero, affinché la reti risponda correttamente -1. Per la seconda immagine (in alto a destra) la rete dovrebbe rispondere +1, ma l'unica differenza nell'immagine, rispetto alla prima, è nella parte sinistra, quindi tale cambiamento deve produrre un incremento nella somma pesata. Analogamente per la terza immagine (in basso a sinistra) la rete dovrebbe rispondere +1, ma l'unica differenza nell'immagine, rispetto alla prima, è nella parte destra, quindi tale cambiamento deve produrre ancora una volta un incremento nella somma pesata. Per la quarta immagine la rete dovrebbe rispondere, invece, -1, ma tale immagine differisce dalla prima per la parte sinistra (allo stesso modo della seconda immagine) e per la parte destra (allo stesso modo della terza immagine), quindi tale cambiamento deve produrre, ancora una volta, un incremento nella somma pesata. Allora, sulla quarta immagine la rete non può rispondere -1.

Feed-Forward a due strati e funzione di output di Heaviside

Supponiamo di avere una rete neurale feed-forward multistrato full-connected costituita da *d* input, uno strato di *m* neuroni interni con funzione di output di Heaviside e un neurone *z* di output con

funzione di output g che realizza un AND logico dei neuroni interni (per tutti i nodi supponiamo una funzione di attivazione pari all'identità). Tale rete realizza la seguente funzione:

$z(\boldsymbol{x})=g(\Sigma_h w''_h \theta(\Sigma_k w'_{hk} x_k))$ con $k=1,2, \ldots,d$ e $h=1,2, \ldots m$.

dove abbiamo incluso il bias nella sommatoria.

Tale rete può rappresentare una regione di decisione semplicemente connessa e convessa (Ricordo che: Un sottoinsieme di R^d è detto **semplicemente connesso** se è fatto di un "pezzo solo" e se non ha "buchi". Un sottoinsieme di R^d si dice **convesso** se per ogni coppia di punti x,y appartenenti al sottoinsieme il segmento che li congiunge è interamente contenuto nel sottoinsieme stesso. Un insieme convesso è semplicemente connesso).

Osserviamo che ciascun nodo interno definisce un confine di decisione coincidente con un iper-piano (una retta nel caso di $d=2$), quindi gli m iper-piani definiti dagli m nodi interni vanno a definire una unica regione di decisione (chiusa o aperta) convessa. Il nodo di output permette di decidere, così, se un punto appartiene oppure no alla regione di decisione individuata dagli m nodi interni, cioè se gli m nodi interni hanno valore di output pari a 1 l'input **x** appartiene alla regione di decisione altrimenti no.

Dimostriamo che la regione di decisione K definita da tale rete è convessa.

Data una rete con unità a soglia con m nodi interni (ciascuno rappresentante un confine di decisione dato da un iper-piano) e con un nodo di output che realizza un AND logico, tale rete definisce una regione di decisione, K, convessa. Per dimostrare ciò dobbiamo dimostrare che comunque presi due punti $\mathbf{x}^A$ e $\mathbf{x}^B$ appartenenti alla regione K allora se scelgo un punto $\mathbf{x}^C$ appartenente al segmento congiungente, tale punto appartiene ancora alla regione K.

Tale punto può essere così espresso: $\mathbf{x}^C = \alpha\mathbf{x}^A + (1-\alpha)\mathbf{x}^B$ con $0 \leq \alpha \leq 1$

Sappiamo che il valore di output della rete per i due punti $\mathbf{x}^A$ e $\mathbf{x}^B$ è

$z(\mathbf{x}^A)>0$ e $z(\mathbf{x}^B)>0$, quindi, per come è costruita la rete,

per ciascun nodo interno si ha $\Sigma_k w'_{hk} x^A_k > 0$ e $\Sigma_k w'_{hk} x^B_k > 0$

(ricordo che la funzione g realizza un AND logico e che, quindi, affinché dia 1 come valore di uscita deve essere $\theta(\Sigma_k w'_{hk} x_k)=1$ per ogni h)

Allora possiamo scrivere

$\Sigma_k w'_{hk} x^C_k = \Sigma_k w'_{hk} (\alpha x^A_k + (1-\alpha) x^B_k) = \alpha \Sigma_k w'_{hk} x^A_k + (1-\alpha) \Sigma_k w'_{hk} x^B_k$

dato che $\Sigma_k w'_{hk} x^A_k > 0$ e $\Sigma_k w'_{hk} x^B_k > 0$

risulta, così, $\Sigma_k w'_{hk} x^C_k > 0$ per ogni h.

Quindi $\mathbf{x}^C$ appartiene alla regione K.

Facciamo notare che:

- Se rilassiamo la condizione di avere un AND logico come output possiamo avere anche regioni di decisione più complesse, ma non ogni tipo di regioni di decisione.
- Reti neurali che hanno tre strati di pesi possono generare qualunque regioni di decisione (Lippmann,1987), la quale può definire regioni che possono essere non convesse e disgiunte.

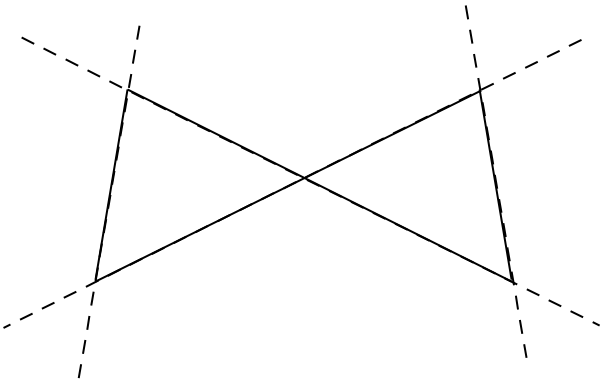


Figura 8: Un esempio di un confine di decisione che non può essere rappresentato da una rete con due strati di unità a soglia (Gibson e Cowan, 1990)

Consideriamo, ora, reti neurali con funzione di output di tipo *sigmoidale*.

Reti a strati con funzione di output di tipo sigmoidale

Reti neurali feed-forward a strati sono in genere utilizzate con:

- Una funzione di attivazione pari all'identità
- Una funzione di output di tipo sigmoidale, ad esempio $s(x)=1/(1+e^{-x})$ la quale assume i seguenti valori $s(0)=1/2$, $\lim_{x\to+\infty} s(x)=1$ e $\lim_{x\to-\infty} s(x)=1$ con una forma somigliante ad una "s" (vedi figura).

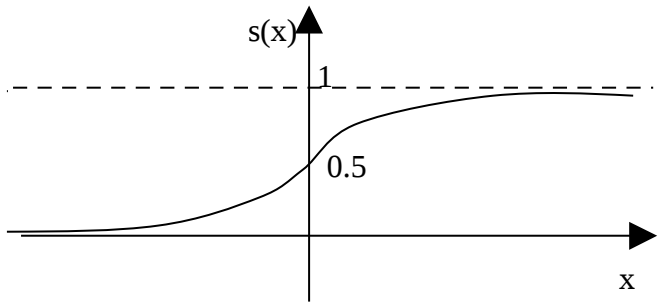


Figura 9: Grafico della funzione sigmoide $s(x)=1/(1+e^{-x})$

Diamo, come prima cosa, una possibile giustificazione della scelta della funzione s(x). Dimostriamo, cioè, che se ipotizziamo che la distribuzione di probabilità di x condizionata alla classe, $p(x/ C_k)$, sia una gaussiana l'uso di una funzione di output sigmoidale permette di interpretare l'output del rete feed-forward come una probabilità a-posteriori.

Supponiamo, per semplicità, di avere d=1.

$$p(x/C_k)=(1/(2\pi)^{1/2}\sigma)\ \exp(-(x-\mu_k)^2/2\sigma^2) \qquad (1)$$

$$P(C_1/x)=p(x/ C_1)P(C_1)/p(x)= p(x/ C_1)P(C_1)/[p(x/C_1)P(C_1)+ p(x/C_2)P(C_2)]= 1/(1+\exp(-a))$$

$$\text{Dove } a=\ln [p(x/C_1)P(C_1)/ p(x/C_2)P(C_2)] \qquad (2)$$

Se sostituiamo (1) in (2) otteniamo:

$a=wx+w_0$ dove $w=(\mu_1-\mu_2)/\sigma^2$ e $w_0=-(1/2)\mu_1/\sigma^2+(1/2)\mu_2^2/\sigma^2+\ln[P(C_1)/P(C_2)]$

Abbiamo visto, quindi, l'importanza di avere come funzione di attivazione la funzione sigmoidale $s(x)=1/(1+e^{-x})$ in quanto in questo caso l'output della rete può essere interpretato come probabilità condizionata P(C/x).

In pratica molto spesso invece di s(x) è utilizzata la funzione tanh, cioè

$\tanh(x)= (e^x -e^{-x})/(e^x +e^{-x})$

in quanto reti neurali con tale funzione spesso "apprendono più velocemente".

Osservazione 1

Osserviamo che s(x) differisce da tanh(x) solo per una trasformazione lineare: 2s(2x)-1=tangh(x), infatti

$2s(2x)-1=2/(1+ e^{-2x}) -1=[2-(1+ e^{-2x})]/(1+ e^{-2x})=(1- e^{-2x})/ (1+ e^{-2x})= e^{x} (1- e^{-2x})/ e^{x} (1+ e^{-2x})=$

$(e^{x} - e^{-x})/ (e^{x} + e^{-x})=\tanh(x)$

quindi una rete neurale con funzione di attivazione pari a s(x) è equivalente a una rete neurale con funzione di attivazione pari a tanh(x) ma con pesi e bias differenti.

Osservazione 2

- Un nodo con funzione di attivazione pari a s(x) (o tangh) può approssimare un nodo con funzione di attivazione lineare con una accuratezza arbitraria e
- Un nodo con funzione di attivazione pari a s(x) (0 tangh) può approssimare un nodo con funzione di attivazione a gradino con una accuratezza arbitraria, semplicemente scalando i pesi e i bias.

Reti neurali con due strati e nodi interni con funzione di attivazione sigmoidale

Consideriamo, ora, reti neurali con due strati, nodi interni con funzione di output sigmoidale e con funzione di attivazione dei nodi di output lineare..

Osserviamo che, in generale, data una rete a due strati, l'output dei nodi interni è dato da

$z_k=f_1(\Sigma_j w^{(1)}_{kj} * x_j)$

l'output dei nodi di output e'

$o_h=f_2(\Sigma_k w^{(2)}_{hk} * z_k)= f_2(\Sigma_k w^{(2)}_{hk} * f_1(\Sigma_j w^{(1)}_{kj} * x_j))$

che con f_2 lineare possiamo scrivere come

$$o_h= \Sigma_k w^{(2)}_{hk} * \Phi_k(x) \qquad [3]$$

con $\Phi_k(x)= f_2(f_1(\Sigma_j w^{(1)}_{kj} * x_j))$

quindi o_h e' dato da una combinazione lineare di funzioni non lineari $\Phi_k(x)$. Per una opportuna scelta delle funzioni $\Phi_k(x)$, talvolta chiamate funzioni di base, la funzione [3] può approssimare qualunque funzione continua con precisione arbitraria.

E' possibile dimostrare che se scegliamo f_1=s(x) (funzione sigmoide) e f_2 lineare una rete neurale a due strati può approssimare ogni funzione continua da uno spazio finito dimensionale ad un altro con precisione arbitraria, con la condizione di avere un numero sufficiente di nodi interni (Funahashi, 1989; Hornik, 1989).

Più formalmente, se definiamo come misura di vicinanza tra due funzioni f: $K \rightarrow R$ e g: $K \rightarrow R$:

$d_K(f, g) = \sup_{x \in K} |f(x) - g(x)|$, con f e g continue e definite sul sottoinsieme compatto K di R^d .

Allora, data la funzione $g(\boldsymbol{x})=f_2(\sum_k w^{(2)}_{hk} * f_1(\sum_j w^{(1)}_{kj} * x_j))$ *definita da una rete neurale con m nodi interni, un nodo di output,* f_2 *funzione lineare e* f_1 *continua, limitata e non costante*

allora se la funzione f_1 è continua, limitata e non costante, abbiamo il seguente teorema (Hornik,1991):

Per ogni funzione continua $f(\boldsymbol{x})$ *definita su K e per ogni* $\varepsilon > 0$, *esiste un* m_ε *tale che* $d_K(g, f) < \varepsilon$ *per ogni* $m > m_\varepsilon$ *(cioè, se la rete ha un numero sufficiente di nodi interni).*

Importante corollario di tale teorema è che, nel contesto dei problemi di classificazione, tali reti possono approssimare qualunque regione di decisione con accuratezza arbitraria.

Diamo un dimostrazione per grandi linee di tale teorema.

Consideriamo il caso di due variabili di input x_1 e x_2 e una singola variabile di output y (l'estensione ad un più largo numero di input o output è abbastanza immediato), cioè abbiamo una funzione continua $y= y(x_1,x_2)$.

Per ogni fissato valore di x_1 possiamo approssimare $y(x_1,x_2)$ da una serie di Fourier:

$y(x_1,x_2) \cong \Sigma_s A_s (x_1)\cos(sx_2)$

gli stessi coefficienti $A_s (x_1)$ possono essere espresso da una serie di Fourier:

$y(x_1,x_2) \cong \Sigma_s \Sigma_l A_{sl} \cos(lx_1)\cos(sx_2)$

sapendo che $\cos\alpha\cos\beta=1/2[\cos(\alpha+\beta)+\cos(\alpha-\beta)]$

possiamo allora scrivere $y(x_1,x_2)$ come una combinazione lineare di termini della forma

$\cos(z_{sl})$ e $\cos(z'_{sl})$.

Osservando, infine, che la funzione cos(x) può essere approssimata con accuratezza arbitraria da una combinazione lineare di funzioni a gradino, allora $y(x_1,x_2)$ può essere approssimata da una rete neurale con due strati avente unità interne con funzione di attivazione pari a quella di Heaviside e unità di output lineare.

A questo punto sapendo che "Un nodo con funzione di attivazione pari a s(x) può approssimare un nodo con funzione di attivazione a gradino con una accuratezza arbitraria semplicemente scalando i pesi e i biases", il teorema resta dimostrato.

Alcune osservazioni:

- Tale rete può approssimare simultaneamente sia la funzione sia le sue derivate (Hornik, 1990)
- Se proviamo ad approssimare una funzione h(x) con una rete che ha un numero M finito di nodi interni, poi ci sarà sempre un errore residuo, tale errore decresce come $O(1/M)$ (Jones,1992; Barron,1993).
- L'utilizzo di reti con più strati potrebbe portare ad una approssimazione più efficiente nel senso di raggiungere la stessa approssimazione con un numero più piccolo di pesi.

Un altro risultato interessante (anche se solo dal punto di vista teorico) scaturisce dal teorema di **Kolmogorov** : *Ogni funzione continua di n variabili (per un dominio chiuso e limitato) può essere rappresentata esattamente con la sovrapposizione di un piccolo numero di funzioni di una variabile*.
Cioè, se abbiamo una funzione y: $(x_1,x_2,\ldots x_d) \rightarrow (y_1, y_2, \ldots, y_m)$ possiamo scrivere,

$$y_h = \Sigma_k g_h(z_k), \quad k=1,2, \ldots,2d+1 \qquad [4]$$

con $z_k = \Sigma_i \lambda_i h_k(x_i)$, i=1,2, …,d

Evidentemente tale espressione può essere realizzata da una rete neurale a tre strati.
Purtroppo, l'importanza pratica, per ora, di tale risultato è molto limitata in quanto:
1) le funzioni h_i sono tutt'altro che "dolci" (smooth). La presenza di funzioni non "dolci" fa si che la rete è estremamente sensibile alle variabili di input, quindi processi di apprendimento sono difficili da realizzare.
2) Le funzioni g_h dipendono fortemente dalla particolare funzione y(x) che si desidera rappresentare. Questo implica che tale approccio non può essere sufficientemente generale (per ogni funzione da rappresentare è necessario scegliere le corrette g_h, se tali funzioni non si conosconono ritorniamo di nuovo in un problema di approssimazione per tali funzioni!).

Apprendimento e generalizzazione di una rete neurale

Supervised learning

Abbiamo visto che, tra i risultati più importanti, una rete neurale feed-forward a due strati di pesi può approssimare una funzione continua con il grado di accuratezza voluto, supposto che abbia un numero sufficiente di nodi interni.

Il problema ora si sposta su come fare ad avere una tale rete in quanto della funzione da modellare f: $\mathbf{x}=(x_1, x_2, ..., x_d) \rightarrow f(\mathbf{x})=(y_1, y_2, ..., y_c)$, in genere, si conosce solo un insieme finito di coppie $(\mathbf{x}^k, \mathbf{t}^k)$ dove $\mathbf{t}^k$ è il valore della funzione $f(\mathbf{x})$ in $\mathbf{x}^k$ più un eventuale errore $\boldsymbol{\varepsilon}(\mathbf{x}^k)$ (Ad esempio $\mathbf{t}_k$ è ottenuto da misurazioni sperimentali). La funzione, come abbiamo visto, è allora modellata in termini di funzioni matematiche che contengono un certo numero di parametri regolabili

$y_k= y_k(\mathbf{x};\mathbf{w})$ con k=1, 2, …, c

dove, come visto, **w** sono i pesi e i bias della rete neurale.

Il processo di determinare il valore di tale parametri a partire da un insieme finito di coppie $(\mathbf{x}_k, \mathbf{t}_k)$, *training set*, è chiamato *training* della rete.

Una proprietà fondamentale dei training set è che i dati dovrebbero esibire una regolarità di base, rappresentata dalla funzione $f(\mathbf{x})$, che è disturbata da un rumore random, cioè i valori $\mathbf{t}^k= f(\mathbf{x}^k)+\boldsymbol{\varepsilon}^k$ del training set sono valori che presentono una regolarità determinata dalla funzione f che vogliamo approssimare più un errore (rumore).

Il gol del training è di ottenere una rete neurale che catturi le regolarità presenti nel training set e che, quindi, sappia fare delle buone previsioni per nuovi dati, cioè che abbia capacità di *generalizzare*. Per misurare la capacità di generalizzare della rete, viene generato un secondo insieme di coppie, detto *validation set*, sul quale verificare la generalizzazione del sistema. Vogliamo subito sottolineare che il processo di apprendimento deve essere capace di catturare dal training set le regolarità sottostanti e non i dettagli specifici (cioè il contributo del rumore). Una volta terminato il processo di apprendimento tramite il training e il validation set, si verifica la bontà del modello ottenuto su un ulteriore insiemi di coppie $(\mathbf{x},\mathbf{t})$ detto *test set*.

Il processo di apprendimento, allora, consiste nel:

1. Modificare i pesi e i bias della rete in modo da minimizzare una fissata funzione di errore $E(\mathbf{w})$ sul training set con il vincolo che anche sul validation set ci sia un errore "piccolo" (in genere utilizzando un diversa funzione di errore $E'(\mathbf{w})$)

2. Una volta ottenuto, attraverso il passo 1, un valore $\mathbf{w}=\mathbf{w}^*$ per i pesi e i bias**,** verificare la "bontà" della rete ottenuta sul test set. Utilizzando, ad esempio, come misura della "bontà" della rete la funzione di errore $E'(\mathbf{w}^*)$ sul test set, oppure, in un problema di classificazione, la recall e precision (che abbiamo visto in precedenza) della rete.

Una classica funzione di errore sul training set è data da

$$E=(\Sigma_n[y(\mathbf{x}^n;\mathbf{w}) - \mathbf{t}^n]^2)/2 \qquad [1]$$

Dove $y(\mathbf{x}^n;\mathbf{w})=(y^n_1, y^n_2, \ldots, y^n_c)$ sono i valori di uscita dei nodi di output della rete con input pari a all'input della n-sima coppia del training set $\mathbf{x}^n=(x^n_1, x^n_2, \ldots,, x^n_d)$ e $\mathbf{t}^n=(t^n_1, t^n_2, \ldots,, t^n_c)$

Una classica funzione di errore sul validation set, invece, è data da una funzione di errore che in genere coincide con l'errore quadratico medio:

$$E= (\Sigma_n[y(\mathbf{x}^n; \mathbf{w}^*) - \mathbf{t}^n]^2)/N \qquad [2]$$

In questo caso si preferisce tale funzione d'errore siccome la forte dipendenza dal numero di punti considerata è rimossa.

Osserviamo che la funzione di errore, in maniera del tutto generale, può essere riscritta come

$E=\Sigma_n E^n$,

dove E^n è l'errore per il singolo punto del training set, (ad esempio nel caso particolare della funzione di errore [1] si ha $E^n =([y(\mathbf{x}^n;\mathbf{w}) - \mathbf{t}^n]^2)/2$)

Una piccola digressione. Un processo di apprendimento che consiste nella minimizzazione di una funzione di errore, e quindi la presenza di valori target $\mathbf{t}^k$, è chiamato *supervised learning*. Un secondo tipo di apprendimento è l'*unsupervised learning* che non comporta la presenza di dati target. In questo caso invece di apprendere un mapping input-output il gol può essere quello di modellare la distribuzione di probabilità dei dati di input o scoprire cluster o altre strutture nei dati. C'è anche una terza forma di apprendimento, *reinforcment learning*, il quale è simile al supervised learning ma in questo caso gli output della rete sono solo classificati come buoni o cattivi, ma non c'è nessun valore desiderato. **Fine della digressione**

Per minimizzare la funzione di errore uno degli algoritmi più usati è la **Discesa del gradiente**.

Discesa del gradiente e Back-propagation

Molti algoritmi di apprendimento hanno una procedura iterativa per minimizzare la funzione di errore, con la determinazione dei valori dei pesi fatta in passi successivi. Ad ogni passo si possono distinguere due differenti fasi:

1) Nella prima fase, sono calcolate le derivate della funzione d'errore rispetto ai pesi. Un metodo computazionale molto efficiente per il calcolo di tali derivate è la *back-propagation*.
2) Nella seconda fase, le derivate sono usate per calcolare i nuovi pesi della rete. La più semplice di tale tecniche comporta la discesa del gradiente (Rumelhart, 1986).

E' fondamentale sottolineare che le due fasi sono distinte.

Si ha, cioè, una variazione del generico peso w_{ji}, ad ogni passo del processo iterativo prima accennato, data da:

$w_{ji} = w_{ji} - \eta \partial E / \partial w_{ji}$

con η coefficiente costante in genere compreso tra 0 e 1.

Vediamo, ora, come calcolare $\partial E / \partial w_{ji}$ utilizzando gli stessi valori di output dei nodi della rete.

Calcolo delle derivate della funzione di errore $\partial E / \partial w_{ji}$: Back Propagation.

Ricordo, innanzitutto, che in una rete feed-forward ciascun nodo computa una somma pesata dei suoi input

$a_j = \Sigma_i w_{ji} * z_i$ [3]

dove z_i è l'output di un nodo il quale invia una connessione al nodo j.

$z_j = g(a_j)$ dove con g indichiamo la funzione di output. [4]

Per calcolare $\partial E / \partial w_{ji}$, dato che $E = \Sigma_n E^n$, possiamo focalizzare la nostra attenzione sul calcolo di $\partial E^n / \partial w_{ji}$.

Una prima ipotesi necessaria per calcolare le derivate parziali della funzione di errore rispetto a i pesi è la seguente:

Supponiamo che E^n possa essere espresso come funzione differenziabile delle variabili d'output.

Dato una coppia ($\mathbf{x}^n$, $\mathbf{t}^n$) del training set, supponiamo di aver dato in input alla rete $\mathbf{x}^n$ e di aver calcolato le attivazioni di tutti i nodi interni e dei nodi d'output, per successive applicazioni della [3] e [4] (come visto, in questo caso, si ha una forward-propagation).

Per calcolare $(\partial E^n/\partial w_{ji})$ osserviamo che E^n dipende da w_{ji} attraverso a_j. Si può dimostrare allora che:

$$(\partial E^n / \partial w_{ji}) = \delta_j z_i \text{ dove } \delta_j = (\partial E^n / \partial a_j) \qquad [5]$$

L'equazione [5] ci dice che la deriva richiesta è ottenuto semplicemente moltiplicando il valore di output del nodo allo "start" della connessione associata al peso w_{ji} per un certo valore δ_j associato al nodo alla fine della connessione con peso w_{ji} . Osserviamo che, quindi, la derivata della funzione di errore rispetto al peso della connessione che va dal nodo i al nodo j è calcolato tramite una formula *locale*.

Come calcolare le δ_j?

Per le unità di output si ha: $\delta_k = (\partial E^n / \partial a_k) = g'(a_k) (\partial E^n / \partial y_k)$

Per le restanti unità: $\delta_j = (\partial E^n / \partial a_j) = g'(a_j) \Sigma_k w_{kj} * \delta_k$ [6]

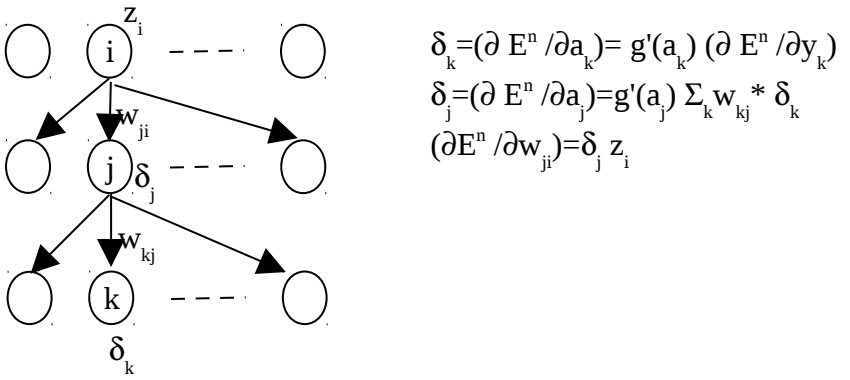
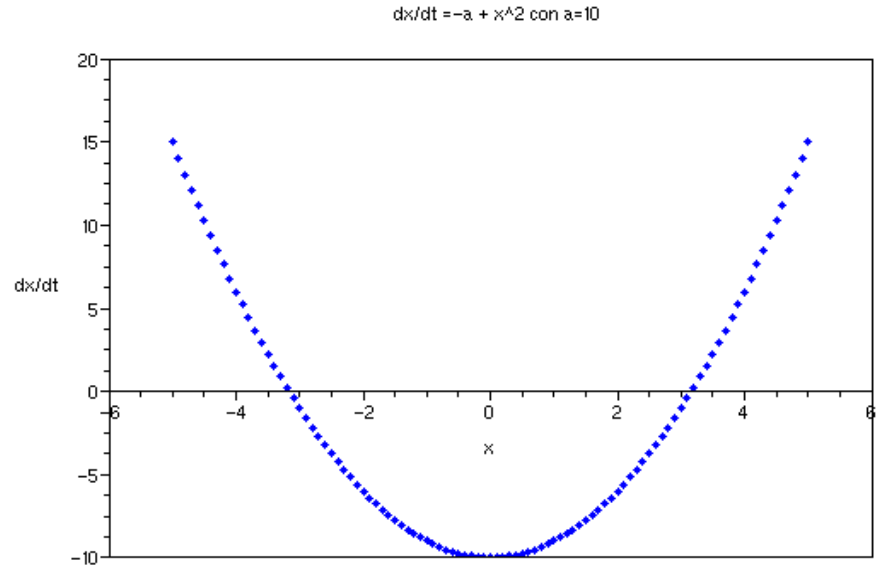


Figura 10: *Calcolo della derivata parziale della funzione di errore rispetto ad un peso tramite back-propagation.*

si ha, quindi, una formula ricorsiva tramite la quale a partire dalle nodi "più esterni" (i nodi di output) possiamo "risalire", in maniera inversa alla forward-propagation delle attivazioni dei nodi, al valore δ_j di tutti i nodi, da qui il nome di back-propagation.

Possiamo allora riassumere la back-propagation per valutare le derivate della funzione di errore E^n , rispetto ai pesi, in quattro fasi

1) Applicare un vettore di input (pattern) $\mathbf{x}^n$ alla rete in provocando una propagazione avanti (forward propagation).



2) Valutare $\delta_k = (\partial E^n/\partial a_k)$ per i nodi di output, dove a_k è l'input del nodo k di output.
3) Valutare tutti gli altri δ_j utilizzando la [6], cioè propagando all'indietro il valore dei δ.
4) Valutare le derivate richieste utilizzando la [5], cioè $(\partial E^n/\partial w_{ji})=\delta_j z_i$

Le derivate rispetto all'intera funzione di errore allora sono date da

$(\partial E/\partial w_{ji})= \Sigma_n (\partial E^n/\partial w_{ji})$

Su tale sommatoria, però, sono da fare alcune osservazioni legate al "tempo" in cui aggiorniamo i pesi. L'aggiornamento dei pesi può procedere in due modi:

1) *on-line learning*. I pesi vengono aggiornati dopo che ciascun pattern $\mathbf{x}^n$ è presentato alla rete e sono calcolate le derivate di E^n, cioè $\Delta w_{ji} = -\eta\delta_j z_i$. In questo caso la rete "cambia" ad ogni presentazione di un pattern e l'errore totale $E=\Sigma_n E^n$ deve essere calcolato al termine di un "ciclo" di addestramento, cioè dopo che sono stati modificati i pesi e i bias per tutti gli elementi del training set.
2) *batch learning*. Oppure dopo che sono state calcolate le derivate della funzione di errore totale E (cioè della funzione di errore relativo a tutto il training set), si ha allora un aggiustamento di pesi dato da $\Delta w_{ji} = -\eta\Sigma_n\delta_j^n z_i^n$.

Quindi il valore di $(\partial E/\partial w_{ji})= \Sigma_n (\partial E^n/\partial w_{ji})$ cambia a seconda se operiamo un on-line learning o un batch learning.

Quale è il vantaggio di utilizzare la back-propagation per calcolare le derivate della funzione di errore?

Supponiamo di avere una rete con W pesi, se noi scriviamo esplicitamente le formule delle derivate della funzione di errore e poi calcoliamo numericamente tale derivate, il calcolo di tali derivate richiede un complessità computazionale pari a $O(W^2)$ per ogni input (Infatti per ogni peso w_{ji} dobbiamo calcolare il valore della funzione di errore in w_{ji} e in $w_{ji}+dw_{ji}$, il che significa calcolare il valore di output della rete per due differenti pesi con un tempo di calcolo pari a O(W). Dato che bisogna calcolare la derivata rispetto a tutti i pesi c'e' necessità di un tempo di calcolo pari a $O(W^2)$). Utilizzando la back-propagation, invece, si ha una complessità computazionale pari a $O(W)$ per ogni input. La back-propagation risulta, allora, una tecnica per il calcolo della derivata della funzione di errore rispetto ai pesi che permette un drastico guadagno da un punto di vista della complessità computazionale.

Dimostriamo, ora, la formula $(\partial E^n / \partial w_{ji})=\delta_j z_i$ e quelle per il calcolo dei δ_j.

Backpropagation: Dimostrazione

Nell'ambito dell'algoritmo di discesa del gradiente abbiamo visto come è fondamentale calcolare le derivate della funzione di errore rispetto ai pesi. Per fare ciò abbiamo introdotto le seguenti formule:
$(\partial E^n / \partial w_{ji})=\delta_j z_i$ dove $\delta_j=(\partial E^n / \partial a_j)$

L'equazione ci dice che la deriva richiesta è ottenuto semplicemente moltiplicando il valore di output del nodo allo start della connessione associata al peso w_{ji} per il valore di δ del nodo alla fine della connessione associata al peso w_{ji} (cioè abbiamo una formula *locale*).

Per le unità di output si ha:

$\delta_k=(\partial E^n / \partial a_k)= g'(a_k) (\partial E^n / \partial y_k)$

Per le restanti unità

$\delta_j=(\partial E^n / \partial a_j)=g'(a_j) \Sigma_k w_{kj} * \delta_k$

Dimostriamo tali equazioni.

La funzione di errore E^n dipende da un peso w_{ji} solo attraverso l'input al nodo j, cioè a_j. Allora possiamo scrivere:

$(\partial E^n / \partial w_{ji})=(\partial E^n / \partial a_j) (\partial a_j / \partial w_{ji})$

siccome

$a_j =\Sigma_h w_{jh} z_h$

$(\partial a_j / \partial w_{ji}) = z_i$

quindi

$(\partial E^n / \partial w_{ji}) = (\partial E^n / \partial a_j) z_i$

poniamo $\delta_j = (\partial E^n / \partial a_j)$, allora

$(\partial E^n / \partial w_{ji}) = \delta_j z_i$

resta da vedere come calcolare i δ_j.

Per un generico nodo della rete, E^n dipende da a_j tramite gli a_k dei nodi che ricevono connessioni dal nodo j, cioè

$\partial E^n / \partial a_j = \Sigma_k (\partial E^n / \partial a_k)(\partial a_k / \partial a_j)$

dato che

$a_k = \Sigma_h w_{kh} z_h = \Sigma_h w_{kh} g_h(a_h)$

si ha

$\partial a_k / \partial a_j = w_{kj} g'_j(a_j)$, quindi

$\partial E^n / \partial a_j = \Sigma_k (\partial E^n / \partial a_k) w_{kj} g'_j(a_j) = g'_j(a_j) \Sigma_k w_{kj} (\partial E^n / \partial a_k)$

e dato che per definizione $\delta_j = (\partial E^n / \partial a_j)$, si ottiene

$\delta_j = g'_j(a_j) \Sigma_k w_{kj} \delta_k$ (1)

dove l'indice k corre su i nodi che ricevono connessione dal nodo j.

Se consideriamo un generico nodo di output k, invece, dato che $E^n = E(y_1, y_2, ..., y_m)$ e $y_k = g_k(a_k)$ si ha

$\delta_k = (\partial E^n / \partial a_k) = (\partial E^n / \partial y_k) g'_k(a_k)$ (2)

Ancora una volta, allora, riassumiamo che la back-propagation per valutare le derivate della funzione di errore E_n , rispetto ai pesi, si può suddividere in quattro fasi

1) Applicare un vettore di input $\mathbf{x}^n$ alla rete provocando una propagazione in avanti (forward propagation)
2) Valutare $\delta_k = (\partial E^n / \partial y_k) g'_k(a_k)$ per i nodi di output
3) Valutare tutti gli altri delta utilizzando $\delta_j = g'_j(a_j) \Sigma_k w_{kj} \delta_k$, cioè propagando all'indietro il valore dei δ.
4) Valutare le derivate richieste, cioè $(\partial E^n / \partial w_{ji}) = \delta_j z_i$

Un semplice esempio.

Supponiamo di avere

1. Una rete neurale a due strati di pesi con funzione di output g_2 dei neuroni di output lineare e funzione di output $g_1$dei neuroni interni non lineare
2. $g_1(x)=1/(1+\exp(-x))$
3. Funzione di errore $E=(\Sigma_n[y(\mathbf{x}^n;\mathbf{w}) - \mathbf{t}^n]^2)/2= \Sigma_n E^n$

Per ciascuna coppia $(\mathbf{x}^n,\mathbf{t}^n)$ inviamo in input $\mathbf{x}^n$ alla rete e calcoliamo tramite propagazione feed-forward il vettore di uscita $\mathbf{y}^n$.

Come prima cosa, allora, dobbiamo calcolare i δ_k dei nodi di output, cioè

$\delta_k= g_2'(a_k)\ (\partial E^n / \partial y_k)$

dato che, nel nostro caso, è

1. $g_2(x)=x$ lineare e quindi $g_2'(a_k)=1$
2. $E^n =1/2\Sigma_k (y^n_k - t^n_k)^2$ o più semplicemente $E^n =1/2\Sigma_k (y_k - t_k)^2$, e quindi $\partial E^n / \partial y_k=(y_k - t_k)$

Si ottiene

$\delta_k=(y_k - t_k)$

mentre per i nodi interni si ha:

$\delta_j=(\partial E^n / \partial a_j)= g_1'(a_j)\ \Sigma_k w_{kj} * \delta_k$

Datto che $g_1'(x)=(1/(1+\exp(-x)))'=(\exp(-x))/ (1+\exp(-x))^2=$

$=(1+\exp(-x))/ (1+\exp(-x))^2-1/(1+\exp(-x))^2=1/(1+\exp(-x)) -1/(1+\exp(-x))^2= g_1(x) – (g_1(x))^2=$

$= g_1(x)(1- g_1(x))$

si ottiene

$\delta_j= g_1(a_j)(1- g_1(a_j))\Sigma_k w_{kj} * \delta_k$

ricordando che, per definizione, $z_j=g_1(a_j)$ si ha:

$\delta_j= z_j(1- z_j)\Sigma_k w_{kj} * \delta_k$

allora le derivate della funzione di errore rispetto al secondo (e in questo caso ultimo) e al primo layer di pesi sono date, rispettivamente, da:

$(\partial E^n / \partial w''_{kj})=\delta_k z_j=(y_k - t_k) z_j$

$(\partial E^n / \partial w'_{ji})=\delta_j x_i$, con $\delta_j= z_j(1- z_j)\Sigma_k w_{kj} * \delta_k$

e i pesi sono aggiornati in una dei seguenti due modi:

- (on-line learning) $w''_{kj}= w''_{kj}-\eta(y_k - t_k)z_j$ e $w'_{ji}= w'_{ji}-\eta\delta_j x_i$
- (batch learning) $w''_{kj}= w''_{kj}-\eta\sum_n(y^n_k - t^n_k)z^n_j$ e $w'_{ji}= w'_{ji}-\eta\sum_n\delta^n_j x^n_i$

Un algoritmo di on-line learning:

```
1. Propagazione in avanti dell'input
2. FOR j←1 TO len(hiddenLayer)
3.     hiddenLayer[j] ← hiddenBiases[j]
4.     FOR i←1 TO len(inputLayer)
5.         hiddenLayer[j]← hiddenLayer[j]+weightLayer1[j][i]* inputLayer[i]
6.     hiddenLayer[j] ← 1/(1+exp(-hiddenLayer[j]))
7. FOR k←1 TO len(outputLayer)
8.     outputLayer[k] ← outputBiases[k]
9.     FOR j←1 TO len(hiddenLayer)
10.        outputLayer[k]← outputLayer[k]+weightLayer2[k][j]* hiddenLayer[j]
11. Propagazione all'indietro dei delta e calcolo dei nuovi pesi
12. FOR k←1 TO len(outputLayer)
13.    delta2[k] ←(outputLayer[k] – targets[k])
14. FOR j←1 TO len(hiddenLayer)
15.    delta1[j] ←hiddeLayer[j](1- hiddeLayer[j])
16.    temp←0
17.    FOR k←1 TO len(outputLayer)
18.        temp←temp+ weightLayer2[k][j]*delta2[k]
19.    delta1[j]← temp* delta1[j]
20. FOR k←1 TO len(outputLayer)
21.    FOR j←1 TO len(hiddenLayer)
22.        weightLayer2[j][k]← weightLayer2[j][k]-eta*delta2[k]*hiddenLayer[j]
23. FOR j←1 TO len(hiddenLayer)
24.    FOR i←1 TO len(inputLayer)
       weightLayer1[j][i]← weightLayer1[j][i]-eta*delta1[j]*inputLayer[i]
```

Funzione d'errore e alcune precisazioni

Funzione di errore

Perché minimizzare $E= \sum_n \sum_k [y_k(\mathbf{x}^n,\mathbf{w}) - t_k^n]^2/2$?

Molte funzioni di errore possono essere derivate dal principio di massima verosimiglianza.

Dato un training set $\{\mathbf{x}^n,\mathbf{t}^n\}$ rappresentante una data funzione $\mathbf{h}(\mathbf{x})$, vogliamo massimizzare la probabilità di avere proprio tale training set nel caso in cui al posto della funzione $\mathbf{h}(\mathbf{x})$ mettiamo un funzione modello $\mathbf{y}(\mathbf{x}^n,\mathbf{w})$. La verosimiglianza del training set può essere scritta nel seguente modo.

Sia $p(\mathbf{x},\mathbf{t})$ la la densità di probabilità di avere la coppia di valori $(\mathbf{x},\mathbf{t})$ (ricordo che la probabilità di avere $\mathbf{x}$ appartenente ad un dato sottoinsieme $\mathbf{X}$, $\mathbf{x} \in \mathbf{X}$, e $\mathbf{t}$ appartenente ad un dato sottoinsieme $\mathbf{T}$, $\mathbf{t} \in \mathbf{T}$, è data da $P(\mathbf{x} \in \mathbf{X},\mathbf{t} \in \mathbf{T})= \int_{x \in X} \int_{t \in T} f(x,t)\, dx\, dt$) allora la verosimiglianza del training set è data da:

$L=\prod_n p(\mathbf{x}^n,\mathbf{t}^n)$ assumendo che i punti $(\mathbf{x}^n,\mathbf{t}^n)$ siano indipendenti

allora

$L=\prod_n p(\mathbf{x}^n,\mathbf{t}^n)= \prod_n p(\mathbf{t}^n/\mathbf{x}^n)\, p(\mathbf{x}^n)$

Volgiamo massimizzare la probabilità di avere il training set dato. Per fare questo possiamo minimizzare la seguente funzione:

$E=-\ln L$

Cioè

$E=-\ln L=-\ln\prod_n p(\mathbf{t}^n/\mathbf{x}^n)\, p(\mathbf{x}^n)= -\sum_n \ln p(\mathbf{t}^n/\mathbf{x}^n)\, p(\mathbf{x}^n)= -\sum_n \ln p(\mathbf{t}^n/\mathbf{x}^n) - \sum_n \ln p(\mathbf{x}^n)$

Il secondo termine non dipende dai parametri della rete (cioè indipendentemente dalla scelta dei parametri $\mathbf{w}$ di $\mathbf{y}(\mathbf{x}^n,\mathbf{w})$ la densità di probabilità a priori di $\mathbf{x}$ sarà sempre la stessa).

Quindi possiamo minimizzare la funzione

$E=-\sum_n \ln p(\mathbf{t}^n/\mathbf{x}^n)$ dove E è detta *funzione di errore*.

Ancora possiamo scrivere:

$E=-\sum_n \ln \prod_k p(t_k^n/\mathbf{x}^n)= -\sum_n \sum_k \ln p(t_k^n/\mathbf{x}^n)$

Assumiamo che la variabile target $t_k= h_k(\mathbf{x})+ \varepsilon_k$ e che l'errore ε_k abbia una distribuzione normale, cioè

$p(\varepsilon_k)= \exp(-\varepsilon_k^2/2\sigma^2)/(\sqrt{2\pi}\sigma)$

Da qui possiamo scrivere che

$p(t_k^n/\mathbf{x}^n)= \exp(-[y_k(\mathbf{x}^n,\mathbf{w}) -t_k^n]^2/2\sigma^2)/(\sqrt{2\pi}\sigma)$

dove abbiamo sostituito $h_k(\mathbf{x})$ con il suo modello $y_k(\mathbf{x}^n,\mathbf{w})$

quindi

$E= -\sum_n \sum_k \ln \exp(-[y_k(\mathbf{x}^n,\mathbf{w}) -t_k^n]^2/2\sigma^2)/(\sqrt{2\pi}\sigma)$

$E= -\sum_n \sum_k \ln \exp(-[y_k(\mathbf{x}^n,\mathbf{w}) -t_k^n]^2/2\sigma^2)+ Nd \ln(\sqrt{2\pi}\sigma)$

$E= \sum_{n=1..N} \sum_{k=1..d} [y_k(\mathbf{x}^n,\mathbf{w}) -t_k^n]^2/2\sigma^2)+ Nd \ln(\sqrt{2\pi}\sigma)$

Il termine $Nd \ln(\sqrt{2\pi}\sigma)$ è indipendente dai parametri della rete quindi la funzione da minimizzare è $E= \sum_n \sum_k [y_k(\mathbf{x}^n,\mathbf{w}) -t_k^n]^2/2$ dove abbiamo omesso il termine σ^2 (o abbiamo supposto $\sigma =1$), ottenendo la classica somma degli errori quadratici.

Osservazione.

Come errore sul test set spesso utilizziamo l'errore quadratico medio (RMS, Root Mean Square):

$E^{RMS}=\sum_n \sum_k [y_k(\mathbf{x}^n,\mathbf{w}^*) -t_k^n]^2/\sum_n \sum_k [y_k(\mathbf{x}^n,\mathbf{w}^*) -m_k]^2)$

dove $\mathbf{w}^*$è il vettore di pesi della rete già addestrata e m è il vettore valore medio dei target del test set, cioè $m_k=(1/N') \sum_n t^n_k$ e N' è il numero di elementi del test set.

Interpretazione dell'output della rete

Si può dimostrare che addestrare la rete andando a minimizzare la funzione di errore $E=\sum_n\sum_k[y_k(\mathbf{x}^n,\mathbf{w})-t_k^n]^2/2$ corrisponde ad avere un valore di output dei nodi di output della rete, dato un pattern **x**, interpretabile come media condizionata su **x** dei vettori target del training set, cioè $y_k(\mathbf{x},\mathbf{w}^*)=<t_k/\mathbf{x}>=\int t_k\, p(t_k/x)\, dt_k$

Questo risultato è vero sotto le seguenti ipotesi

1. Il training set deve essere sufficientemente grande da poter approssimare un insieme infinito, cioè il numero di punti $N\rightarrow+\infty$
2. La funzione della rete $y_k(\mathbf{x},\mathbf{w})$ deve essere sufficientemente generale, cioè il numero di nodi hidden deve essere sufficientemente grande

Dimostrazione:

$E= \lim_{N\to+\infty} \sum_{n=1..N} \sum_{k=1..d} [y_k(\mathbf{x}^n,\mathbf{w}) - t_k^n]^2/2N)=(1/2)\sum_{k=1..d} \iint [y_k(\mathbf{x},\mathbf{w}) - t_k]^2 p(t_k,x) dtk dx$

$E= (1/2)\sum_{k=1..d} \iint [y_k(\mathbf{x},\mathbf{w}) - t_k]^2 \, p(t_k/x) \, p(x) dt_k dx \qquad (1)$

Definiamo

$\langle t_k/\mathbf{x}\rangle = \int t_k \, p(t_k/x) \, dt_k$

$\langle t^2_k/\mathbf{x}\rangle = \int t^2_k \, p(t_k/x) \, dt_k$

Possiamo scrivere

$[y_k - t_k^n]^2 = [y_k - \langle t_k/\mathbf{x}\rangle + \langle t_k/\mathbf{x}\rangle - t_k]^2 = [y_k - \langle t_k/\mathbf{x}\rangle]^2 + [\langle t_k/\mathbf{x}\rangle - t_k]^2 + 2[y_k - \langle t_k/\mathbf{x}\rangle][\langle t_k/\mathbf{x}\rangle - t_k]$

sostituendo in (1)

$E= (1/2)\sum_{k=1..d} \int [y_k - \langle t_k/\mathbf{x}\rangle]^2 \, p(x) \, dx + (1/2)\sum_{k=1..d} \int [\langle t^2_k/\mathbf{x}\rangle - \langle t_k/\mathbf{x}\rangle^2] \, p(x) \, dx$

Osserviamo che il secondo termine è indipendente dai parametri della rete, quindi minimizzare E significa rendere minimo il primo termine il quale diventerà nullo se e solo se $y_k - \langle t_k/\mathbf{x}\rangle = 0$.

Quindi il vettore di pesi $\mathbf{w}^*$ che rende minimo E sarà tale che;

$y_k(\mathbf{x},\mathbf{w}^*) = \langle t_k/\mathbf{x}\rangle$

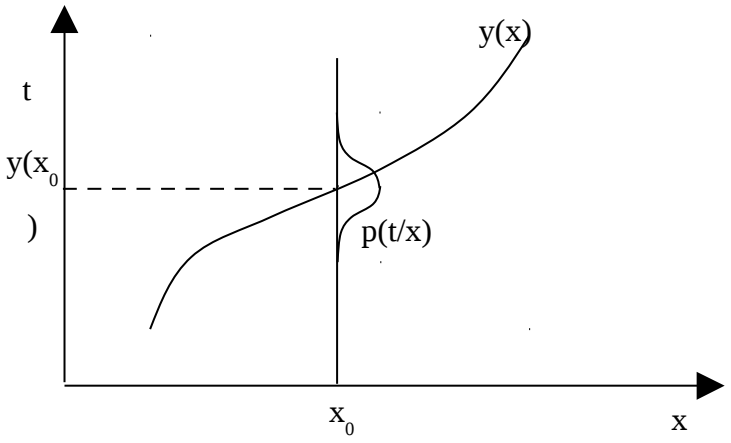


Figura 11: Il valore di output dei nodi di output della rete, dato un pattern x, è interpretabile come media condizionata su **x** dei vettori target del training set, cioè $y_k(\mathbf{x},w^*) = \langle t_k/\mathbf{x}\rangle = \int t_k \, p(t_k/x) \, dt_k$.

La maledizione della dimensionalità

Abbiamo detto che dato una funzione continua f: K -->R con K sottoinsieme compatto di R^d, tale funzione può essere approssimata bene come si vuole con una rete a due strati di pesi supposto di avere un numero sufficiente di nodi interni. Abbiamo mostrato un processo di apprendimento che ci permette di calcolare la rete cercata. Tale processo si basa sulla presenza del Training set composto da coppie $(\mathbf{x}^k,t^k)$. Ci domandiamo quale è il numero sufficiente di coppie affinché il processo abbia buon fine? Tale numero è strettamente legato alle dimensioni del vettore di input, cioè a d. Infatti, se supponiamo che $\mathbf{x}=(x_1,x_2, ...,x_d)$ sia composto da d variabili indipendenti affinché il training set rappresenti un campione rappresentativo del *fenomeno* (la funzione f) che vogliamo rappresentare possiamo procedere nel seguente modo:

- suddividere il dominio di ciascuna variabile in M sotto-intervalli (M circa pari a 20)
- Ottenere, così, M^d celle che partizionano l'intero insieme K
- Da ciascuna cella estrarre una coppia $(\mathbf{x},t)$

In questo modo otteniamo M^d elementi appartenenti al Training Set. Come si vede questo fa sì che, in linea generale, non possiamo scegliere una dimensione del vettore di caratteristiche troppo grande, in quanto già con d=5 otteniamo un numero di coppie paria a $(20)^5 \approx (10)^6$, cioè un numero di coppie dell'ordine di un milione!

Cenni sul problema della generalizzazione

Complessità del modello

Le migliori prestazioni in termini di capacità di generalizzare si hanno per reti la cui *complessità* non è né troppo piccola né troppo grande. Con il termine *complessità della rete* si fa riferimento al numero di parametri della rete, cioè al numero di pesi e di biases (quindi, al numero di nodi). Una rete estremamente complessa può adattarsi ai dati estremamente bene (errore sul training set molto piccolo), ma darà una rappresentazione molto povera delle regolarità espresse dal training set, cioè della funzione da approssimare. Per tale ragione avrà una scarsa capacità di generalizzazione. Viceversa una rete troppo poco complessa non riuscirà ad adattarsi ai dati.

Tale considerazione ci porta ad affermare che è necessario un corretto bilanciamento (trade-off) tra il raggiungere un errore piccolo sui dati del training set e ottenere una rete che realizzi una funzione ragionevolmente *smooth* (o dolce, cioè funzioni che non presentano "brusche" variazioni, ad esempio funzioni appartenenti a C^∞) in modo tale che sia capace di seguire le regolarità del training set (non *over-fitted*). Tale trade-off può essere messo in luce considerando che la funzione di

errore (la somma dei quadrati, $E= \sum_n \sum_k [y_k(\mathbf{x}^n,\mathbf{w}) - t_k^n]^2/2$) può essere espressa, considerando il training set composto da un numero molto grande di coppie, come somma di due termini:

$E= (bias)^2 + variance$

dove

$(bias)^2= (½) \int\{ ?_D [\mathbf{y}(\mathbf{x})] - \langle \mathbf{t/x} \rangle\}^2 p(\mathbf{x})\, d\mathbf{x}$

$variance= (½) \int ?_D [\{ y(\mathbf{x}) - ?_D[\mathbf{y}(\mathbf{x})]\}^2] p(\mathbf{x})\, dx$

e con $?_D$ [] che rappresenta il valore medio calcolato su un numero molto grande di differenti training set tutti composti di N coppie e tutti presi da una popolazione avente una densità di probabilità p(**x**,t).

I due termini in cui è stata suddivisa la funzione di errore hanno il seguente significato:

- Il termine di *bias* misura *di quanto* la media (su tutti i training set) delle funzioni realizzate dalla rete (cioè $?_D [\mathbf{y}(\mathbf{x})]$) differisce dalla funzione desiderata (cioè $\langle \mathbf{t/x} \rangle$).
- Il termine *variance* misura *quanto* le funzioni realizzate dalla rete sono sensibili alla particolare scelta del training set, infatti $y(\mathbf{x}) - ?_D[\mathbf{y}(\mathbf{x})]$ misura proprio la differenza tra la risposta della rete per un particolare training set e la risposta media (su tutti i training set) della rete, sempre avente **x** come input.

Si può dimostrare che al diminuire di un termine l'altro tende a crescere. Ad esempio se il termine *variance* è molto piccolo (o zero) questo implica che $y(\mathbf{x}) \approx ?_D[\mathbf{y}(\mathbf{x})]$. Questa è la tipica situazione in cui la rete è poco, o per niente, sensibile al training set (la risposta della rete sarà sempre la stessa comunque addestrata). Tuttavia questo comporta un alto bias (in quanto, come detto, il bias misura di quanto la media delle funzioni realizzate dalla rete differisce dalla funzione desiderata $\langle \mathbf{t/x} \rangle$).

Tale risultato può essere esteso a molte altre funzioni di errore.

Regolarizzazione

Un modo per ottimizzare la capacità di generalizzare di una rete neurale è di controllare la sua *effettiva complessità*: si considera una rete con molti parametri – alta complessità- ma si cerca di controllare il processo di training, aggiungendo un opportuno termine alla funzione di errore, in modo da ottenere solamente funzioni smooth.

L'errore, allora, diventa:

$E'=E+\nu\Omega$

dove Ω è detto *termine di regolarizzazione* che va a penalizzare le funzioni rappresentate dalla rete che non sono smooth e ν è un coefficiente che esprime quale è l'influenza di Ω. In altre parole, il

termine di regolarizzazione *limita* la classe delle possibili funzioni che la rete può apprendere, durante il processo di training, alle sole funzioni smooth.
In letteratura sono presenti differenti termini di regolarizzazione che hanno tutti lo stesso scopo di far sì che la rete possa apprendere solo funzioni smooth.
Una delle più semplici forme di regolarizzazione è detta *decadimento dei pesi* (*weight decay*).
In questo caso si ha:
$\Omega = (½) \sum_i w_i^2$ dove la somma corre su tutti i pesi.
La scelta di tale termine può essere giustificato da un punto di vista qualitativo dalla seguente considerazione. Un termine di regolarizzazione come quello precedente fa sì che le reti addestrate assumano dei "piccoli" valori per i pesi, quindi i nodi interni di una rete con piccoli valori dei pesi realizzano funzioni quasi lineari (siccome la regione centrale di una sigmoide può essere approssimata tramite una funzione lineare). Questo comporta che la rete rappresenti funzioni con un andamento quasi lineare e, quindi, funzioni smooth.
Un alternativa a tale termine è il seguente.

$(\nu_1/2)\sum_{w\in W1} w^2 + (\nu_2/2)\sum_{w\in W2} w^2$
dove W1 denota l'insieme dei pesi (senza i biases) del primo strato e W2 quello del secondo strato.

Tale termine di regolarizzazione permette di trovare reti che sono indipendenti da una trasformazione lineare degli input.

Funzione di errore per problemi di classificazione a due classi.

In precedenza abbiamo ricavato la somma dei quadrati come funzione di errore partendo dall'ipotesi che il target *t* fosse una variabile continua generata da una funzione smooth a cui è addizionato un rumore Gaussiano. Nel caso di un problema a due classi *t* può assumere solo due valori, cioè $t \in \{0,1\}$. Risulta più opportuno, allora, utilizzare la seguente funzione di errore:

$E= - \sum_n [t^n \ln y^n + (1-t^n)\ln(1-y^n)]$ (detta *cross-entropy error*)
dove y^n rappresenta l'output della rete relativo alla n-sima coppia del treining set e tn il relativo valore target (osservo che in questo caso stiamo considerando un problema a due classi e, quindi, una rete con un solo nodo di output).
Tale funzione di errore è sempre giustificata andando a massimizzare la verosimiglianza. Quindi:
$L= \prod_n p(t^n, \mathbf{x}^n)= \prod_n p(t^n/\mathbf{x}^n)p(\mathbf{x}^n)$
considerando

$E= - \ln L = - \ln \prod_n p(t^n/\mathbf{x}^n)p(\mathbf{x}^n) = -\sum_n \ln p(t^n/\mathbf{x}^n) - \sum_n \ln p(\mathbf{x}^n)$

possiamo allora, per massimizzare la verosimiglianza, minimizzare solo la parte che dipende dai parametri della rete, cioè definire la seguente funzione di errore:

$E= -\sum_n \ln p(t^n/\mathbf{x}^n)$

Come già detto in precedenza noi vorremmo che l'uscita y della rete rappresenti la probabilità a posteriori $P(C_1/x)$ per la classe C_1 (la probabilità per la classe C_2 sarà data da $P(C_2/x)=1-y$). Allora se indichiamocon t=1 il caso in cui il vettore di input appartiene a C_1 e con t=0 il caso in cui il vettore appartiene alla classe C_2, possimo esprimere la probabilità di osserevare entrambi i valori targete con la seguente formula:

$p(t/x)= y^t(1-y)^{1-t}$

(ad esempio la probabilità di osservare t=1 quando y=1, risulta $p(1/x)= 1^1(1-1)^{1-1}=1$).

(Nota che questo e' un caso particolare della distribuzione binomiale)

Quindi:

$E= -\sum_n \ln p(t^n/\mathbf{x}^n) =-\sum_n \ln (y^n)^{t^n}(1-y^n)^{(1-t^n)}$

$E= -\sum_n \ln (y^n)^{t^n} - \sum_n \ln (1-y^n)^{(1-t^n)}=-\sum_n t^n \ln (y^n) - \sum_n (1-t^n) \ln (1-y^n)$

$E = - \sum_n [t^n \ln (y^n) + (1-t^n) \ln (1-y^n)]$

In questo caso le formule per il calcolo della derivate dell'errore rispetto ai pesi resteranno invariate , cioè

$\partial E^n /\partial w_{ij}= z_j \delta_i$

Per le unità di output (in questo caso unica):

$\delta=(\partial E^n /\partial a)= g'(a) (\partial E^n /\partial y)$

Per le restanti unità:

$\delta_j=(\partial E^n /\partial a_j)=g'(a_j) \sum_k w_{kj} * \delta_k$

In questo caso, allora, si ha che cambia solo la derivata dell'errore rispetto ai valori di uscita della rete, si ha cioè:

$(\partial E^n /\partial y)= (y^n-t^n)/(y^n (1-y^n))$ e $\delta=g'(a) (y^n-t^n)/(y^n (1-y^n))$

Si noti che se scegliamo come funzione di output (per il nodo di uscita) l'identità, si ottiene:

$\delta=(y^n-t^n)/(y^n (1-y^n))$

Se, invece, scegliamo come funzione di output la sigmoide $g(x)=s(x)= 1/(1+ \exp(-x))$, dato che $s'(a)= s(a)(1-s(a))$ si ottiene:

$\delta=s'(a^n) (y^n-t^n)/(y^n (1-y^n)) = s'(a) (y^n-t^n)/(y^n (1-y^n)) = s(a^n)(1-s(a^n))(y^n-t^n)/(y^n (1-y^n))$

ricordando che $s(a^n)$ è proprio y^n, allora

$\delta = y^n(1-y^n)(y^n-t^n)/(y^n(1-y^n)) = y^n - t^n$

cioe' lo stesso risultato del caso in cui consideriamo contemporaneamente come funzione di errore la somma dei quadrati e come funzione di output (dei nodi di uscita) l'identità.

Variazioni sulla discesa del gradiente

Il momento

Possiamo calcolare la variazione del peso come dipendente dalla variazione precedente, cioè:

$\Delta w^{(t)}_{ij} = -\eta \partial E/\partial w_{ijj} + \mu \Delta w^{(t-1)}_{ij}$

dove μ è detto *coefficiente del momento* e $\Delta w^{(t-1)}_{ij}$ *momento*

Cosa comporta tale variazione della regola della discesa del gradiente? Cerchiamo di capire cosa accade con l'aggiunta del momento.

Osserviamo che se indichiamo $\Delta w_{ij} = \Delta w^{(1)}_{ij} + \Delta w^{(2)}_{ij} + \Delta w^{(3)}_{ij} + ...$

allora se supponiamo che la derivata non cambia (cioè siamo in una regione in cui la superficie di errore ha curvatura bassa), abbiamo

$\Delta w_{ij} = -\eta \partial E/\partial w_{ij} (1+\mu+ \mu^2+...)$

Se μ è scelto tra zero ed uno, cioè $0< \mu<1$, si ha allora

$\Delta w_{ij} = -(\eta/1-\mu) \partial E/\partial w_{ij}$

Da tale risultato si evince che nel caso in cui durante il processo di apprendimento siamo in una zona della superficie di errore con curvatura bassa il termine momento, allora, ha l'effetto di incrementare la velocità di apprendimento da η a $\eta/(1-\mu)$

Al contrario, se siamo in una regione con alta curvatura nella quale la discesa del gradiente è oscillatoria successivi contributi del termine momento tenderanno a elidersi e la velocità di apprendimento sarà vicino ad η.

Scegliendo il parametro η

Un ovvio problema della discesa del gradiente è che la scelta dei parametri η e μ è fissata a priori. Ad esempio, una possibile modo per aggiornare il parametro η è, ad ogni ciclo del processo di apprendimento, è il seguente

$$\eta_{new} = \begin{cases} \rho\eta_{old} \text{ se } \Delta E<0 \\ \sigma\eta_{old} \text{ se } \Delta E >0 \end{cases}$$

Dove il parametro ρ è scelto poco più grande che l'unità (un tipico valore è ρ=1.1), mentre il parametro σ è scelto abbastanza più piccolo che l'unità (un tipico valore è σ=0.5).

Un altro possibile scelta della regola di aggiornamento dei pesi è la seguente.

Quick-propagation

Tale approccio si basa sull'idea di approssimare la superficie di errore, nei pressi di un minimo, con una parabola. Sotto questa assunzione si ottiene la seguente formula:

$$\Delta w_i^{(t+1)}=[g_i^{(t)} / (g_i^{(t-1)}-g_i^{(t)})] \Delta w_i^{(t)}$$

con $g_i^{(t)} = \partial E/\partial w_i^{(t)}$ e $g_i^{(t-1)} = \partial E/\partial w_i^{(t-1)}$, cioè la derivata della funzione di errore rispetto al peso al tempo t e al tempo t-1, rispettivamente.

In tale algoritmo è necessario "stare attenti" ai casi in cui il denominatore diventa zero o prossimo a zero.

Un altro approccio è il seguente.

Resilient back Propagation (RProp)

In questo caso le variazioni dei pesi sono determinate nel seguente modo:

$\Delta w_{ij}^{(t)}= -\Delta_{ij}^{(t)}$ SE $\partial E/\partial w_{ij}^{(t)}>0$,

$\Delta w_{ij}^{(t)}= +\Delta_{ij}^{(t)}$ SE $\partial E/\partial w_{ij}^{(t)}<0$,

$\Delta w_{ij}^{(t)}= 0$ ALTRIMENTI

dove

$\Delta_{ij}^{(t)} = \eta^{+} \Delta_{ij}^{(t-1)}$ se $\partial E/\partial w_{ij}^{(t)} * \partial E/\partial w_{ij}^{(t-1)}>0$

$\Delta_{ij}^{(t)} = \eta^- \Delta_{ij}^{(t-1)}$ se $\partial E/\partial w_{ij}^{(t)} * \partial E/\partial w_{ij}^{(t-1)} < 0$

$\Delta_{ij}^{(t)} = \Delta_{ij}^{(t-1)}$ altrimenti

con $0 < \eta^- < 1 < \eta^+$ (ad esempio $\eta^- = 0.5$ e $\eta^+ = 1.2$)

Reti Neurali con funzioni a base radiale – RBF

Interpolazione esatta

Abbiamo visto come una rete neurale feed-forward, sotto opportune ipotesi, sia capace di approssimare bene quanto si vuole qualunque funzione continua definita su un compatto. Introdurremmo, ora, un tipo diverso di reti neurali feed-forward dette RBF (Radial Basis Functions, Funzioni a base radiale). A tale scopo consideriamo, allora, sempre il medesimo problema di approssimare una funzione, a partire da un insieme di coppie input-target conosciute, utilizzando il metodo della *interpolazione esatta*.

Si consideri, quindi, un mapping da uno spazio input d-dimensionale $X \subseteq R^d$ ad uno spazio target uno-dimensionale $Y \subseteq R$:

$$f : \mathbf{x} \in X \rightarrow y=f(\mathbf{x}) \in Y \qquad (1)$$

In generale di tale mapping, come già ampiamente detto, non si conosce l'andamento della funzione $f(\mathbf{x})$ ma solo alcune coppie input-output. Si supponga di conoscere n valori di input $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^n \in X$ a cui corrispondono n scalari di output $t^1, t^2, \ldots, t^n \in Y$ (il Training Set, TS).

Il problema dell'interpolazione esatta consiste nel trovare una funzione $h(\mathbf{x})$ che soddisfi i seguenti n vincoli di interpolazione:

$$h(\mathbf{x}^i) = t^i, \; i = 1, \ldots, n. \qquad (2)$$

L'approccio con funzioni a base radiale consiste nel ricercare la funzione $h(\mathbf{x})$ come combinazione lineare di n funzioni, $\phi_j(\mathbf{x})$, che dipendono dalla distanza (in genere euclidea) di $\mathbf{x}$ dall'origine o da un centro $\boldsymbol{\mu}_j$.

La forma di tali funzioni è la seguente:

$$\phi_j(\mathbf{x}) = \phi(\| \mathbf{x} - \boldsymbol{\mu}_j \|) \text{ con } j = 1, \ldots, n \qquad (3)$$

Dove:

- ϕ è una funzione, in genere, non lineare. Una scelta tipica di tali funzioni è la gaussiana $\phi(x)=\exp(-x^2/2\sigma^2)$
- $\| \mathbf{x} - \boldsymbol{\mu}^j \|$ è, in genere, la distanza euclidea di $\mathbf{x}$ da un *centro* $\boldsymbol{\mu}^j$.

Dunque h(x) avrà la forma:

$$h(\mathbf{x}) = \Sigma_{j=1\ldots n}\, w_j\, \phi_j(\mathbf{x}) \qquad (4)$$

Il centro della funzione j-esima, nel caso di interpolazione esatta viene scelto uguale al vettore di input j-esimo, cioè $\boldsymbol{\mu}^j = \mathbf{x}^j$.

Ponendo:

$$\phi_{ij} = \phi_j\,(\mathbf{x}^i) \qquad (5)$$

la condizione di interpolazione descritta dalla (2) può essere espressa, tenendo conto della (4), come:

$\Sigma_{j=1\ldots n}\, w_j \phi_{ij} = t^i$ con i=1,2, …,n

La (5), in forma matriciale, può essere così espressa:

$$\begin{pmatrix} \phi_{11}\,\phi_{12}\ldots\phi_{1n} \\ \phi_{21}\,\phi_{22}\ldots\phi_{2n} \\ \ldots. \\ \phi_{n1}\,\phi_{n2}\ldots\phi_{nn} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \ldots. \\ w_n \end{pmatrix} = \begin{pmatrix} t^1 \\ t^2 \\ \ldots \\ t^n \end{pmatrix}$$

oppure in forma compatta:

$$\boldsymbol{\phi}\mathbf{w}^T = \mathbf{t} \qquad (6)$$

con **t** e **w** vettori di dimensione n e $\boldsymbol{\phi}$ matrice quadrata n × n dove ϕ_{ij} è il valore di $\phi_j(\mathbf{x})$ calcolato nel punto i-simo del TS. Quindi sulla colonna j di $\boldsymbol{\phi}$ ci sono i valori di $\phi_j(\mathbf{x})$ calcolati per tutti i punti del TS.

Una volta scelto il tipo di funzione di base ϕ da utilizzare, il problema dell'interpolazione esatta si riduce alla stima dei valori del vettore **w**.

Dalla 5, se $\boldsymbol{\phi}$ è invertibile, si deduce che:

$$\mathbf{w}^T = \boldsymbol{\phi}^{-1}\mathbf{t} \qquad (7)$$

dunque per la i-esima componente di w risulta che:

$$w_i = \Sigma_{j=1..n}\, (\boldsymbol{\phi}^{-1})_{ij}\, t^j \text{ con } i = 1, \ldots, n \qquad (8)$$

Ovviamente è necessario provare che per il tipo di funzione di base scelta l'inverso della matrice ϕ esista!

Il caso di un mapping da un insieme multi-dimensione ad uno monodimensionale può essere facilmente esteso al caso di un mapping fra due insiemi multi-dimensionali.
Si consideri, allora, un mapping da uno spazio input d-dimensionale $X \subseteq R^d$ ad uno spazio target c-dimensionale $Y \subseteq R^c$:

$$f : \mathbf{x} \in X \rightarrow \mathbf{y}=f(\mathbf{x}) \in Y \qquad (9)$$

In tal caso ogni vettore di input $\mathbf{x}^i$ dovrà essere mappato esattamente in un vettore di output $\mathbf{t}^i$ di componenti $t^i_1, t^i_2, \ldots, t^i_c$. Per cui l'equazione (2) diventa:

$$h_k(\mathbf{x}^i) = t^i_k \text{ con } i = 1, \ldots, n \text{ e } k=1,2, \ldots, c \qquad (10)$$

dove $h_k(\mathbf{x}^i)$ è ottenuta sempre come combinazione lineare di funzioni di base, cioè:

$$h_k(\mathbf{x}) = \Sigma_{j=1\ldots n}\, w_{kj}\phi_j(\mathbf{x}) \qquad (11)$$

e la (10) diventa

$$\Sigma_{j=1\ldots n}\, w_{kj}\phi_j(\mathbf{x}^i) = t^i_k \text{ con } i = 1, \ldots, n \text{ e } k=1,2, \ldots, c \qquad (12)$$

Che in forma matriciale possiamo scrivere come

$\boldsymbol{\phi}\mathbf{W}^T=\mathbf{T}$

Dove ϕ è la stessa matrice n ×n descritta precedentemente, mentre si ha

$$\mathbf{W}= \begin{pmatrix} w_{11}\, w_{12} \ldots w_{1n} \\ w_{21}\, w_{22} \ldots w_{2n} \\ \ldots. \\ w_{c1}\, w_{c2} \ldots w_{cn} \end{pmatrix}$$

e

$$\mathbf{T}= \begin{pmatrix} t^1_1\, t^1_2 \ldots t^1_c \\ t^2_1\, t^2_2 \ldots t^2_c \\ \ldots. \\ t^n_1\, t^n_2 \ldots t^n_c \end{pmatrix}$$

cioè è una matrice nXc di coefficienti da determinare mentre la j-sima riga di **T** corrisponde a $\mathbf{t}^j$.

Anche in questo caso, allora, se ϕ è invertibile, abbiamo

$$\mathbf{W}^T=\phi^{-1}\mathbf{T} \qquad (13)$$

Dalla interpolazione esatta alle reti RBF

L'interpolazione esatta, vista nel paragrafo precedente, costruisce una funzione interpolante che passa esattamente per ogni punto del TS. In presenza di rumore tale funzione risulta molto oscillante ed ha una scarsa capacità di generalizzazione. A tal proposito una funzione più smussata otterrebbe migliori risultati. Un ulteriore limite dell'interpolazione esatta è il numero di funzioni di base che viene scelto uguale al numero di punti nel campione a disposizione, rendendola computazionalmente molto costosa quando questo è abbastanza grande.

Il modello delle reti neurali con funzioni a base radiale introduce le seguenti modifiche per superare i limiti dell'interpolazione esatta:

1) Si hanno a disposizione un numero m di funzioni di base che non dipende dal numero di punti a disposizione (m << n);
2) I centri delle funzioni di base non sono più legati ai vettori di input ma la loro determinazione diventa parte del processo di apprendimento;
3) Anche la determinazione dei parametri della funzione di base scelta sarà parte del processo di apprendimento. Per esempio nel caso di funzioni Gaussiane verrà determinato il parametro σ per ogni funzione;
4) Parametri di bias sono inclusi nella sommatoria.

Applicate le modifiche sopra descritte alla formula (11) si perviene alla seguente espressione:

$$y_k(\mathbf{x}) =\Sigma_{j=1\ldots m}\, w_{kj}\phi_j(\mathbf{x})+ w_{k0} \qquad (14)$$

dove l'interpretazione in termini di rete neurale è immediata (Figura 12):

- Una rete feed-forward composta da 1 strato di d input, 1 strato di m nodi interni e 1 strato di c nodi di output

- Il j-simo nodo interno riceve d-connessioni dagli input con pesi pari a μ^j_i, con i=1,2, …d. Ciascuna connessione porta in input al nodo, dato un valore di input alla rete **x**, un contributo pari a $c^j_i=x_i - \mu^j_i$. L'input del nodo j sarà, allora, calcolato come radice di $\Sigma_{i=1\ldots d} (c^j_i)^2$, cioè $\|\mathbf{x} - \boldsymbol{\mu}^j\|$. La funzione di output del nodo sarà pari a ϕ. Il valore di output del j-simo nodo interno sarà, allora, pari a $\phi_j(\mathbf{x})= \phi(\|\mathbf{x} - \boldsymbol{\mu}^j\|)$.
- $y_k(\mathbf{x})$ è la risposta del k-esimo output della rete all'input **x**. Cioè il k-simo nodo di output riceve m connessioni dai nodi interni con pesi pari a w_{kj}, ha bias pari a w_{k0} e ha funzione di output pari all'identità.

Nel caso in cui la è una Gaussiana l'output del k-simo nodo interno è pari a:

$$y_k(\mathbf{x}) =\Sigma_{j=1\ldots m} w_{kj} \exp(-\|\mathbf{x} - \boldsymbol{\mu}^j\|/2\sigma^2)+ w_{k0} \qquad (15)$$

Osservo che, evidentemente, la matrice **W** della sezione precedente corrisponde alla matrice dei pesi tra lo strato dei nodi interni e lo strato dei nodi di output, cioè la k-sima riga di W corrisponde ai pesi delle connessioni che incidono sul k-simo nodo di output.

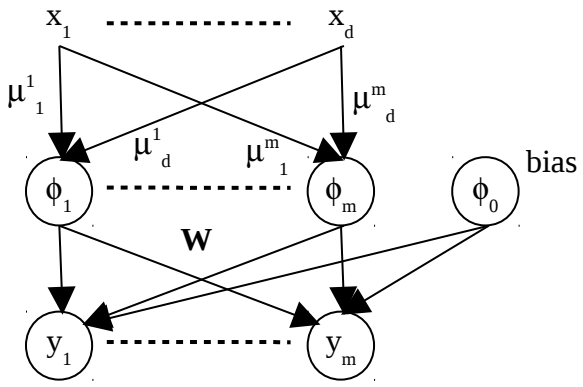


Figura 12: Rete neurale RBF

Addestramento delle reti neurali RBF

L'apprendimento di una rete neurale RBF avviene in due fasi:

1) nella prima fase vengono stimati i parametri delle funzioni di base,
2) mentre nella seconda vengono determinati i pesi del secondo strato mediante il metodo dei minimi quadrati.

La ricerca dei parametri delle funzioni di base avviene, di solito, utilizzando un training non supervisionato mentre la scelta dei pesi del secondo strato con un addestramento supervisionato. Di

seguito supporremo di aver già fissato i centri e i parametri delle funzioni di base e ci concentreremo sulla determinazione dei pesi del secondo strato.

A tale fine, possiamo considerare la nostra rete come una rete con un solo strato di pesi con input pari a $\phi_1(\mathbf{x})$, $\phi_2(\mathbf{x})$, …,$\phi_m(\mathbf{x})$.

In tal caso la determinazione di tali pesi può essere fatta con il metodo dei minimi quadrati. La funzione di errore somma dei quadrati su tutti gli elementi del training set e su tutti gli output può essere scritta nel seguente modo:

$$E=(1/2)\ \Sigma_{n=1\ldots N}\Sigma_{k=1\ldots c}(y_k(\mathbf{x}^n)-t^n_k)^2 \qquad (16)$$

dove $y_k(\mathbf{x}^n)=\Sigma_{j=1\ldots m}\ w_{kj}\phi_j(\mathbf{x}^n)+ w_{k0}$ rappresenta l'output del nodo di output k in funzione del vettore input $\mathbf{x}^n$ e della matrice dei pesi $\mathbf{W}$.

La (16) è una funzione quadratica dei pesi e dunque la sua derivata è una funzione lineare dei pesi. E', quindi, possibile trovare esattamente il minimo di tale funzione.

Prima di ricavare il minimo di tale funzione nel caso generale, mostriamo una interpretazione geometrica del problema dei minimi quadrati in un caso semplice.

Interpretazione geometrica del metodo dei minimi quadrati

Come accennato precedentemente, fissati i parametri delle funzioni di base, la nostra rete può essere vista come una rete ad un solo strato di pesi in cui gli ingressi sono le risposte delle funzioni ϕ_j ai vettori $\mathbf{x}$. Di seguito, quindi, supporremo che la rete abbia un solo nodo di uscita.

Per un particolare input $\phi_{nj}=\phi_j(\mathbf{x}^n)$ l'output della rete sarà:

$$y^n=\Sigma_{j=0\ldots m}\ w_j\phi_j(\mathbf{x}^n) \qquad (17)$$

Consideriamo, ora, gli n valori target del TS e raggruppiamo tali valori target per formare un vettore $\mathbf{t}$ di componenti $t^1, t^2, \ldots, t^n$. Questo vettore apparterrà ad uno spazio Euclideo n-dimensionale D.

Per ogni funzione di base possiamo ripetere lo stesso procedimento visto per $\mathbf{t}$ ed ottenere m vettori $\boldsymbol{\phi}_j$ di componenti $\phi_{1j}, \phi_{2j}, \ldots, \phi_{nj}$, con ciascun vettore di dimensione n che potrà giacere nello stesso spazio n-dimensionale del vettore $\mathbf{t}$.

Adesso supponiamo che il numero di funzioni di base (incluso il bias) sia minore del numero di elementi del TS; ovvero sia m + 1 < n. Gli m + 1 vettori $\boldsymbol{\phi}_j$, con j=0,1, …,m, corrispondenti alle m + 1 funzioni di base, giacciono in un sottospazio Euclideo S a m + 1 dimensioni (cioè i vettori $\boldsymbol{\phi}_j$ definiscono un sottospazio S a m-dimensioni dello spazio a n-dimensioni D in cui giacciono i vettori stessi).

Anche gli output della rete possono essere raggruppati in un unico vettore **y** di dimensione n e componenti y^1, y^2 …, y^n.

Osserviamo che:

$$\mathbf{y} = \Sigma_{j=0\ldots m}\, w_j \boldsymbol{\phi}_j \qquad (18)$$

cioè il vettore n-dimensionale **y** è una combinazione lineare degli m +1 vettori n-dimensionali $\boldsymbol{\phi}_j$.

Quindi **y** deve per forza giacere sullo stesso sottospazio S.

Osserviamo che noi vogliamo sempre che i vincoli (2) siano soddisfatti, cioè

$$y^i = t^i \text{ e, quindi, } \mathbf{y} = \mathbf{t} \qquad (19)$$

D'altra parte abbiamo visto che, in generale, **t** appartiene allo spazio n-dimensionale D e **y** allo spazio m dimensionale S. Quindi affinché la (19) possa essere soddisfatta dovrebbe accadere che **t** giaccia nel sottospazio S, altrimenti la (19) non può essere soddisfatta. Nel caso in cui la (19) non può essere soddisfatta, cioè quando **t** non giace nel sottospazio S possiamo, però, sempre far si che i due vettori **y** e **t** siano il "più vicino possibile" cioè possiamo minimizzare la distanza tra i due vettori, cioè

$$\Sigma_{k=1\ldots n}\, (y^k - t^k)^2 \qquad (20)$$

Che altro non è che l'errore somma dei quadrati.

Vogliamo, quindi, trovare il vettore dei pesi **w** che minimizza tale quantità.

Siccome il vettore **y** è obbligato a stare sul sottospazio S, la soluzione migliore, come detto, è quella che rende minima la distanza tra **y** e **t** che si ha nel caso in cui **y** coincide con la proiezione ortogonale di **t** su S.

Dunque in questo paragrafo abbiamo mostrato come calcolare i pesi del secondo strato di pesi di una rete RBF nel caso particolare in cui abbia un solo nodo di uscita. Nel paragrafo successivo vedremo come calcolare tali pesi nel caso generale in cui la rete abbia c nodi di uscita.

Soluzione ai minimi quadrati

Ritorniamo al caso generale di una rete con un solo strato di pesi, con m nodi di input e c nodi di output. L'errore somma dei quadrati sugli n vettori di training può essere scritto nel seguente modo:

$E = (1/2)\Sigma_{h=1\ldots n} \Sigma_{k=1\ldots c} [\Sigma_{j=0\ldots m} w_{kj} \phi_{hj} - t^h_k]^2$

Per calcolare il minimo di tale errore calcoliamo la derivata dell'errore rispetto a w_{kj} e imponiamo che sia uguale a 0:

$\partial E/\partial w_{kj} = \Sigma_{h=1\ldots n} [\Sigma_{j'=0\ldots m} w_{kj'} \phi_{hj'} - t^h_k] \phi_{hj} = 0$

cioè

$\Sigma_{h=1\ldots n} [(\Sigma_{j'=0\ldots n} w_{kj'} \phi_{hj'} \phi_{hj}) - t^h_k \phi_{hj}] = 0$

Se indichiamo con $\mathbf{w}^k$ la k-sima riga di $\mathbf{W}$ e con $\boldsymbol{\phi}^h$ la h-sima riga di $\boldsymbol{\phi}$, si ottiene

$\Sigma_{h=1\ldots n} [\boldsymbol{\phi}^h (\mathbf{w}^k)^T \phi_{hj} - t^h_k \phi_{hj}] = 0$

Se indichiamo con $\boldsymbol{\phi}_j$ la j-sima colonna di $\boldsymbol{\phi}$ e con $\mathbf{T}_k$ la k-sima colonna di $\mathbf{T}$, si ottiene

$\Sigma_{h=1\ldots n} [\boldsymbol{\phi}^h (\mathbf{w}^k)^T \phi_{hj} - t^h_k \phi_{hj}] = \Sigma_{h=1\ldots n} \boldsymbol{\phi}^h (\mathbf{w}^k)^T \phi_{hj} - \Sigma_{h=1\ldots n} t^h_k \phi_{hj} = (\boldsymbol{\phi}_j)^T \boldsymbol{\phi} (\mathbf{w}^k)^T - (\boldsymbol{\phi}_j)^T \mathbf{T}_k$

E quindi

$(\boldsymbol{\phi}_j)^T \boldsymbol{\phi} (\mathbf{w}^k)^T - (\boldsymbol{\phi}_j)^T \mathbf{T}_k = 0$ per j=0,1, …,m e k=1,2, ..c (21)

Riscrivendo la (21) in forma compatta abbiamo

$(\boldsymbol{\phi}^T \boldsymbol{\phi}) \mathbf{W}^T = \boldsymbol{\phi}^T \mathbf{T}$ (22)

Dove :

- $\boldsymbol{\phi}$ ha dimensioni n x m (per semplicità rinominiamo m+1 con m)
- $\mathbf{W}$ ha dimensioni c x m (per semplicità rinominiamo m+1 con m)
- $\mathbf{T}$ ha dimensioni n x c

Ricordo che ϕ_{ij} è il valore di $\phi_j(\mathbf{x})$ calcolato nel punto i-simo del TS, w_{kj} è il peso associato alla connessione che va dal j-simo nodo interno al k-simo nodo di output ed, infine, la h-sima riga di $\mathbf{T}$ corrisponde al h-simo valore target $\mathbf{t}^h$.

L'equazione (22) è detto *sistema di equazioni normali*. La soluzione di tale sistema è:

$\mathbf{W}^T = (\boldsymbol{\phi}^T \boldsymbol{\phi})^{-1} \boldsymbol{\phi}^T \mathbf{T}$ (23)

Se esiste l'inversa di $(\boldsymbol{\phi}^T \boldsymbol{\phi})$

La matrice $(\boldsymbol{\phi}^T \boldsymbol{\phi})^{-1} \boldsymbol{\phi}^T$ viene spesso indicata con il simbolo $\boldsymbol{\phi}^\dagger$ e viene chiamata *pseudo-inversa* di $\boldsymbol{\phi}$.

Per cui è possibile riscrivere l'equazione precedente nel seguente modo:

$$\mathbf{W}^T = \boldsymbol{\phi}^{\dagger}\,\mathbf{T} \qquad (24)$$

Una soluzione diretta del sistema di equazioni (23) non è numericamente opportuna per via della possibile singolarità della matrice $(\boldsymbol{\phi}^T\boldsymbol{\phi})$ per questo di solito viene utilizzata la tecnica della *Singular Value Decomposition* (SVD).

Grazie alla SVD è possibile decomporre la matrice $\boldsymbol{\phi}$ nel seguente modo:

$$\boldsymbol{\phi} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T \qquad (25)$$

dove

- **U** è una matrice n × n costituita da n vettori ortonormali, cioè **U** è ortonormale.
- $\boldsymbol{\Sigma}$ è una matrice n × m con r ≤ m valori diversi da zero (detti valori singolari σ_i, pari a $\sigma_i = \sqrt{\lambda_i}$ dove λ_i sono gli r autovalori distinti e maggiore di zero della matrice $\boldsymbol{\phi}^T\boldsymbol{\phi}$) sulla diagonale principale e zero altrove,
- **V** è una matrice m × m costituita da m vettori ortonormali, cioè **U** è ortonormale

Le matrici U e V, quindi, godono delle seguenti proprietà:

$$U^TU = UU^T = I \text{ e } V^TV = VV^T = I \qquad (26)$$

dove I è la matrice identità.

Nel seguito indicheremo con $\boldsymbol{\Sigma}^{-1}$ l'*inversa modificata* di $\boldsymbol{\Sigma}$. Tale matrice ha sulla diagonale i reciproci dei valori singolari, quando questi sono diversi da zero, ed ha dimensioni m × n.

Notiamo che per $\boldsymbol{\Sigma}^{-1}$ è vero la seguente: $\boldsymbol{\Sigma}^{-1}(\boldsymbol{\Sigma}^T)^{-1}\boldsymbol{\Sigma}^T = \boldsymbol{\Sigma}^{-1}$.

Calcoliamo, ora, la soluzione dell'equazione (23):

$$\mathbf{W}^T = (\boldsymbol{\phi}^T\boldsymbol{\phi})^{-1}\boldsymbol{\phi}^T\,\mathbf{T}$$

Che applicando (25) e (26) si trasforma in

$$\begin{aligned}
\mathbf{W}^T &= ((\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T)^T\,\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T)^{-1}\,(\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T)^T\,\mathbf{T} \\
&= (\mathbf{V}\boldsymbol{\Sigma}^T\mathbf{U}^T\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T)^{-1}\,(\mathbf{V}\boldsymbol{\Sigma}^T\mathbf{U}^T)\,\mathbf{T} \\
&= (\mathbf{V}\boldsymbol{\Sigma}^T\boldsymbol{\Sigma}\mathbf{V}^T)^{-1}\,\mathbf{V}\boldsymbol{\Sigma}^T\mathbf{U}^T\,\mathbf{T} \\
&= \mathbf{V}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\Sigma}^T)^{-1}\mathbf{V}^T\mathbf{V}\boldsymbol{\Sigma}^T\mathbf{U}^T\,\mathbf{T} \\
&= \mathbf{V}\boldsymbol{\Sigma}^{-1}(\boldsymbol{\Sigma}^T)^{-1}\boldsymbol{\Sigma}^T\mathbf{U}^T\,\mathbf{T} \\
&= \mathbf{V}\boldsymbol{\Sigma}^{-1}\mathbf{U}^T\,\mathbf{T}
\end{aligned}$$

quindi

$$\mathbf{W}^T = \mathbf{V}\boldsymbol{\Sigma}^{-1}\mathbf{U}^T\mathbf{T} \quad (27)$$

Quindi, per trovare la matrice dei pesi $\mathbf{W}^T$ della rete RBF si può procedere nel seguente modo:

1) Decomporre la matrice $\boldsymbol{\phi}$ con la SVD in $\boldsymbol{\phi} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$;
2) Calcolare $\boldsymbol{\Sigma}^{-1}$ l'*inversa modificata* di $\boldsymbol{\Sigma}$. Tale matrice, ricordo, ha sulla diagonale i reciproci dei valori singolari σ_i presenti sulla diagonale di $\boldsymbol{\Sigma}$, quando questi sono diversi da zero, mentre tutti gli altri valori sono nulli ed ha dimensioni $m \times n$.
3) Calcolare la matrice $\mathbf{W}^T$ come prodotto tra le matrici $\mathbf{V}\boldsymbol{\Sigma}^{-1}\mathbf{U}^T\mathbf{T}$

Soluzione ai minimi quadrati con regolarizzazione

In questo paragrafo illustreremo come calcolare i pesi del secondo strato di pesi della rete RBF illustrato nel paragrafo precedente quando aggiungiamo alla funzione di errore somma dei quadrati un termine di regolarizzazione del tipo *weigth-decay*.
La formula della funzione di errore somma dei quadrati, quando viene aggiunto il termine di regolarizzazione, diventa:

$$E = (1/2)\Sigma_{h=1\ldots n}\Sigma_{k=1\ldots c}[\Sigma_{j=0\ldots n} w_{kj}\phi_{hj} - t^h_k]^2 + (\nu/2)\Sigma_{k=1\ldots c}\Sigma_{j=0\ldots n}(w_{kj})^2 \quad (28)$$

Per minimizzare tale funzione calcoliamone la derivata e imponiamo che sia uguale a zero. Pervenendo alla seguente espressione:

$(\boldsymbol{\phi}^T\boldsymbol{\phi})\mathbf{W}^T - \boldsymbol{\phi}^T\mathbf{T} + \nu\mathbf{W}^T = 0$

risolvendo rispetto a $\mathbf{W}^T$ otteniamo:

$\mathbf{W}^T = (\boldsymbol{\phi}^T\boldsymbol{\phi} + \nu\mathbf{I})^{-1}\boldsymbol{\phi}^T\mathbf{T}$

Decomponendo la matrice $\boldsymbol{\phi}$ con la SVD in $\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$.

$\mathbf{W}^T = ((\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T)^T\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T + \nu\mathbf{I})^{-1}(\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T)^T\mathbf{T}$

e tenendo conto dell'ortogonalità della matrice U

$\mathbf{W}^T = (\mathbf{V}\boldsymbol{\Sigma}^T\boldsymbol{\Sigma}\mathbf{V}^T + \nu\mathbf{I})^{-1}(\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T)^T\mathbf{T}$

sfortunatamente la componente$(\mathbf{V}\boldsymbol{\Sigma}^T\boldsymbol{\Sigma}\mathbf{V}^T + \nu\mathbf{I})$ di tale espressione non può essere ulteriormente semplificata. Una soluzione al problema consiste nell'aggiungervi un termine che ovviamente modificherà anche l'espressione del termine di regolarizzazione, cioè:

$\mathbf{W}^T = (\mathbf{V}\boldsymbol{\Sigma}^T\boldsymbol{\Sigma}\mathbf{V}^T + \nu\,\mathbf{V}\mathbf{V}^T)^{-1}(\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T)^T\mathbf{T}$

ovvero:

$\mathbf{W}^T = (\mathbf{V}(\boldsymbol{\Sigma}^T\boldsymbol{\Sigma} + \nu\,\mathbf{I})\mathbf{V}^T)^{-1}(\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T)^T\mathbf{T}$

e ricordando che l'inversa di una matrice ortonormale è uguale alla sua trasposta:

$\mathbf{W}^T = \mathbf{V}(\boldsymbol{\Sigma}^T\boldsymbol{\Sigma} + \nu\,\mathbf{I})^{-1}\mathbf{V}^T\mathbf{V}\boldsymbol{\Sigma}^T\mathbf{U}^T\mathbf{T}$

$$\mathbf{W}^T = \mathbf{V}(\boldsymbol{\Sigma}^T\boldsymbol{\Sigma} + \nu\,\mathbf{I})^{-1}\boldsymbol{\Sigma}^T\mathbf{U}^T\mathbf{T} \qquad (29)$$

A questo punto ragioniamo solamente sul termine $(\boldsymbol{\Sigma}^T\boldsymbol{\Sigma} + \nu\mathbf{I})^{-1}$.

Cominciamo con l'osservare che la matrice nxm $\boldsymbol{\Sigma}$ ha valori diverso da zero solo sulla diagonale $\boldsymbol{\Sigma}_{ii}=\sigma_i$, con i=1,2, …m , quindi la matrice $\boldsymbol{\Sigma}^T\boldsymbol{\Sigma}$ è una matrice quadrata nxn con valori non nulli solo sulla diagonale e pari σ^2_i.

Notiamo adesso che anche $\nu\mathbf{I}$ è una matrice diagonale m×m in cui sulla diagonale vi è sempre ν. La somma di questa matrice con la precedente da come risultato ancora una matrice diagonale che può essere invertita in modo tale da ottenere una matrice mxm non nulla solo sulla diagonale e con valori pari ai reciproci dei valori sulla diagonale di $\boldsymbol{\Sigma}^T\boldsymbol{\Sigma} + \nu\,\mathbf{I}$, ovvero:

$(\boldsymbol{\Sigma}^T\boldsymbol{\Sigma} + \nu\,\mathbf{I})^{-1}_{ii} = 1/(\nu + \sigma^2_i)$ con i=1,2, …,m.

moltiplicando questa matrice per $\boldsymbol{\Sigma}^T$, di dimensione m×n, otteniamo una matrice m × n non nulla solo sulla diagonale e con valori $\sigma_i/(\nu + \sigma^2_i)$, indicando tale matrice con $\boldsymbol{\Sigma}_R^{-1}$ si ha cioè

$$(\boldsymbol{\Sigma}_R^{-1})_{ij} = \sigma_i/(\nu + \sigma^2_i) \text{ con } i=j \text{ e } i=1,2, \ldots,m, \text{ e } (\boldsymbol{\Sigma}_R^{-1})_{ij} = 0 \text{ altrimenti.} \qquad (30)$$

La (29) tramite la (30) si trasforma allora in

$$\mathbf{W}^T = \mathbf{V}\,\boldsymbol{\Sigma}_R^{-1}\mathbf{U}^T\mathbf{T} \qquad (31)$$

Che è analoga alla (27), cioè $\mathbf{W}^T = \mathbf{V}\boldsymbol{\Sigma}^{-1}\mathbf{U}^T\mathbf{T}$, valida nel caso di assenza di un termine di regolarizzazione. Notiamo che la (31) si riduce alla (27) nel caso in cui ν è pari a zero

Scelta dei parametri delle funzioni di base

In questo paragrafo ci concentreremo sulla stima dei parametri delle funzioni di base, in particolare sulla scelta dei centri e dell'ampiezza di tali funzioni.

La scelta dei centri delle funzioni di base dovrebbe essere tale da rispecchiare la densità dei dati di ingresso. Vi sono vari approcci per scegliere questi centri. Un metodo è quello di avere tante funzioni quanti sono i punti a disposizione e scegliere come centri delle funzioni proprio questi punti.

Questa strada non è tuttavia praticabile, per due ragioni fondamentali: la prima è il costo computazionale che crescerebbe troppo al crescere dei punti a disposizione; la seconda è che ciò costringerebbe la rete a creare un mapping che passi esattamente per i punti di training che di solito sono affetti da errore creando problemi di overfitting e dunque scarsa capacità di generalizzazione.

Un altro approccio è quello di scegliere i centri delle funzioni di base in modo casuale fra i punti di training. Il risultato di tale approccio dipende molto da quanto il campione di training sia effettivamente rappresentativo della popolazione.

Un ulteriore metodo per scegliere i centri delle funzioni di base consiste nell'utilizzare un algoritmo di clustering sui punti di training. Anche in questo caso è importante che i dati di training sia effettivamente rappresentativi della popolazione. Un algoritmo semplice di clusterizzazione è il K-Means descritto di seguito.

Anche la scelta dell'ampiezza delle funzioni di base può essere fatta seguendo varie strade. Una di queste consiste nello scegliere l'ampiezza delle funzioni di base come media della distanza Euclidea dei centri delle p funzioni di base più vicine. Il calcolo dell'ampiezza in questo caso può essere, però, computazionalmente abbastanza costoso.

Nel caso dell'uso di algoritmi di clustering come metodo per scegliere i centri delle funzioni di base si possono calcolare le ampiezze delle funzioni di base come la deviazione standard dei punti appartenenti ai cluster rispetto al centro del cluster stesso.

Algoritmo di clustering K-means

K-means è un algoritmo di clustering in cui il numero di cluster è deciso a priori. Il funzionamento dell'algoritmo è il seguente:

Supponiamo di aver a disposizione N punti $\mathbf{x}^n$ e di voler trovare un insieme di m vettori rappresentativi $\boldsymbol{\mu}^j$ con j = 1, . . . ,m. L'algoritmo prova a partizionare l'insieme dei punti x^n in m sottoinsiemi disgiunti S_j contenenti N_j punti in modo da minimizzare la funzione somma dei quadrati data da:

$J=\sum_{j=1,\ldots,k} \sum_{\mathbf{x}\in Sj} ||\mathbf{x} - \boldsymbol{\mu}^j||^2$

Dove $\boldsymbol{\mu}^j$ è il vettore "media" dei punti appartenenti a Sj, cioè:

$\boldsymbol{\mu}^j = (1/N_j) \sum_{\mathbf{x}\in Sj} \mathbf{x}$

La versione batch di tale algoritmo consta dei seguenti passi:

1. Assegna gli N punti a K insiemi S_j in modo casuale
2. Calcola il punto medio $\boldsymbol{\mu}^j$ per ogni insieme S_j
3. Riassegna ogni punto al sottoinsieme S_j che ha il punto medio più vicino al punto stesso.
4. Se almeno un punto è stato spostato da un insieme ad un altro ritorna al passo 2, altrimenti l'algoritmo termina.

Recall, Precision and ROC curve in classification problems

We introduce here the concepts of *recall* and *precision*.

Let us suppose that U is a set of N elements to be classified in two different classes C_1 and C_2 . And we know which U elements belong to which classes. Once the classification process has been completed, we can define:

True Positive (TP)= An element which is classified as belonging to C_1 and the actual classification is C_1 .

False Positive (FP)= An element which is classified as belonging to C_1 and the actual classification is C_2 .

True Negative (TN)= An element which is classified as belonging to C_2 and the actual classification is C_2 .

False Negative (FN)= An element which is classified as belonging to C_2 and the actual classification is C_1

Thus, we can define (see Table 1)

Recall = (number of TP) /((number of TP)+(number of FN)).

It measures the proportion of actual positives which are correctly identified.

Precision = (number of TP) /((number of TP)+(number of FP)).

It measures the proportion of element with positive classification who are correctly classified.

Specifity = (number of TN) /((number of FP)+(number of TN)).

It measures the proportion of actual negatives which are correctly identified.

		Actual Values		
		p	**n**	Total
Prediction Outcome	**p'**	*Number of TP*	*Number of TN*	P'
	n'	*Number of FP*	*Number of FP*	N'
	Total	P	N	

Table 1

The "best" case is: Recall=1 and Precision=1.

Recall=1 corresponds to the case of all the the elements belonging to the class C_1 have been correctly assigned to the class C_1 .

Precision=1 corresponds to the case of all the elements belonging to the class C_2 have been correctly assigned to the class C_2 .

So, recall and precision can be used to measure the "soundness" of a classification system

Note that the output values of a two-layer neural networks are continuous values, however in a classification problem we want a discrete output (for example 0 or 1).

Thus, we need a *decision threshold* T so that:

- if the output value of the network is greater than T, then the output of the classification system is 1.

- otherwise the output of the classification system is 1.

The parameter T is not set during the learning process.

How can one choose the threshold T?

A possible approach is the ROC (Receiver Operating Characteristic) curve.

In a ROC curve the Racall (also said True Positive Rate) is plotted in function of the False Positive Rate (1- specificity) for different decision threshold values.

Each point on the ROC (see for example Figure 1) curve represents a recall/specificity pair corresponding to a particular decision threshold T.

The area under the ROC curve is a measure of how well a parameter can distinguish between two classes.

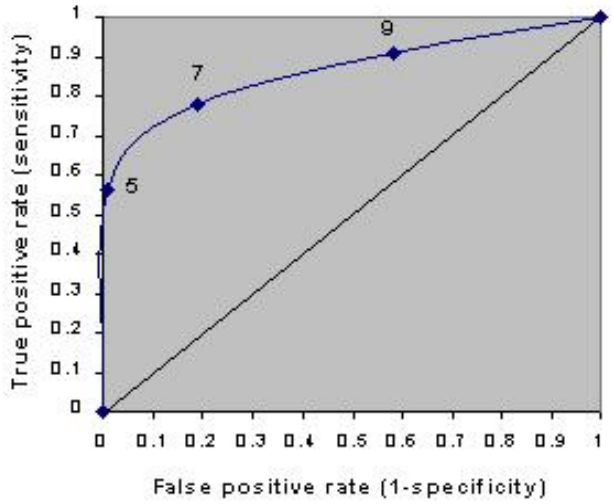
A ROC curve demonstrates several things:

1. It shows the trade-off between sensitivity and specificity (any increase in sensitivity will be accompanied by a decrease in specificity).

2. The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the classification system.

3. The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the classification system.

4. The area under the curve is a measure of classification system accuracy.



T=0.3
T=0.5
T=0.7

Figure 1: An example of ROC Curve.

Analisi alle Componenti Principali (Principal Component Analysis – PCA) (vedi capitolo 8.6 del Bishop)

Come già detto in precedenza la dimensione dei vettori di caratteristiche gioca un ruolo importante. Per tale motivo può essere decisivo *ridurre la dimensionalità dei dati*. In tale operazione è fondamentale preservare quanta più informazione possibile. Qui discuteremo di una tecnica lineare per la riduzione della dimensionalità, detta Analisi alle Componenti Principali (in inglese, Principal Component Analysis – PCA). Analizzeremo i suoi principali aspetti teorici e ne vedremo una sua realizzazione in termini di reti neurali artificiali addestrate con una tencnica non supervisionata.

Aspetti teorici della PCA

Vogliamo realizzare una mapping lineare da vettori **x** d-dimensionali a vettori **z** m-dimensionali con m <d. Senza perdere di generalità ciascun vettore x può essere scritto come:

$x = \sum_{i=1}^{d} z_i u_i$ dove z_i sono dei coefficienti reali e u_i sono d vettori d-dimensionali ortonormali (cioè rappresentano una base dello spazio d-dimensionale). Ricordo che la proprietà di essere ortonormali significa che $u_i u_j^T = 0 \, se \, i \neq j \, , altrimenti \, è \, uguale \, a \, 1$.

Si osservi che il generico z_i può essere calcolato come prodotto scalare tra **x** e $\mathbf{u}_i$, cioè

$z_i = x u_i^T$

(Si ricorda che stiamo considerando i vettori **x** e $\mathbf{u}_i$ come vettori riga)

In questo modo ciascun vettore **x** d-dimensionale può essere rappresentato tramite ancora *d* coefficienti z_i.

Supponiamo ora di avere solo m < d vettori $\mathbf{u}_i$, così che noi avremmo solo m coefficienti per rappresentare ciascun vettore **x**.

In altre parole, un vettore **x** è approssimato da un vettore $\tilde{x} = \sum_{i=1}^{m} z_i u_i + \sum_{i=m+1}^{d} b_i u_i$ dove b_i sono dei valori costanti.

Dato un insieme di N vettori $\mathbf{x}^n$ con n=1,...,N, dobbiamo, ora, scegliere i "migliori" vettori $\mathbf{u}_i$ e le migliori costanti b_i in modo tale da preservare quanta più informazione possibile.

L'errore sul vettore $\mathbf{x}^n$ introdotto dalla riduzione di dimensionalità è dato da:

$$x^n - \tilde{x}^n = \sum_{i=m+1}^{d} (z_i^n - b_i)$$

Vogliamo, allora, minimizzare tale errore sull'intero dataset. Questo significa che vogliamo minimizzare il seguente errore:

$$E_m = \frac{1}{2}\sum_{n=1}^{N} \|x^n - \tilde{x^n}\|^2 = \frac{1}{2}\sum_{n=1}^{N}\sum_{i=m+1}^{d} (z_i^n - b_i)^2$$

Se minimizziamo rispetto a b_i , significa che dobbiamo porre a zero le derivate di E_m rispetto a b_i. Ottenendo così:

$$-\sum_{n=1}^{N} (z_i^n - b_i) = 0 \quad \text{cioè}$$

$$b_i = \frac{1}{n}\sum_{n=1}^{N} z_i$$

dato che $z_i^n = x^n u_i^T$ possiamo anche scrivere

$$b_i = \bar{x} u_i^T$$

Quindi l'errore minimo rispetto ai bi risulta

$$E_m = \frac{1}{2}\sum_{n=1}^{N}\sum_{i=m+1}^{d} ((x^n - \bar{x})u_i^T)^2$$

Tale formula può anche essere riscritta in termini della matrice di covarianza di dimensioni $d \times d$, indicata in genere con Σ , dell'insieme dei vettori $\mathbf{x}^n$. Tale matrice è calcolata nel seguente modo:

$\Sigma = \sum_{n=1}^{N} (x^n - \bar{x})^T (x^n - \bar{x})$ dove sulla diagonale sono presenti le varianze delle componenti del vettore x, mentre nella generica posizione (riga i-sima, colonna j-sima con i differente da j) c'è la covarianza tra la componete i e la componente j del vettore **x** (un valore positivo indica che al crescere di una componente cresce anche l'altra, un valore negativo viceversa).

Allora, in termini di covarianza, abbiamo

$$E_m = \frac{1}{2}\sum_{i=m+1}^{d}\sum_{n=1}^{N} u_i (x^n - \bar{x})^T (x^n - \bar{x}) u_i^T = \frac{1}{2}\sum_{i=m+1}^{d} u_i [\sum_{n=1}^{N} (x^n - \bar{x})^T (x^n - \bar{x})] u_i^T$$

cioè

$$E_m = \frac{1}{2} \sum_{i=m+1}^{d} u_i \Sigma u_i^T$$

Ora resta da minimizzare l'errore rispetto alla scelta dei vettori $\mathbf{u}_i$.

Sotto l'ipotesi che tali $\mathbf{u}_i$ siano ortonormali si può dimostrare che tale problema di minimizzazione si riduce al calcolo degli autovalori ed autovettori della matrice di covarianza (vedi appendice E del Bishop). Cioè:

$$\Sigma u_i^T = \lambda_i u_i^T \quad .$$

Quindi l'errore sarà dato da:

$$E_m = \frac{1}{2} \sum_{i=m+1}^{d} u_i \Sigma u_i^T = \frac{1}{2} \sum_{i=m+1}^{d} \lambda_i$$

Ciascun autovettore è chiamato componente principale.

Gli autovettori corrispondenti ai primi m autovalori più grandi sono quelli scelti. Cioè la prima componente pricipale sarà quella con autovalore più grande, la seconda componente principale sarà quella con il successivo autovalore più grande, e così via. L'autovalore corrispondente rappresenta la varianza spiegata dei dati nella direzione dell'autovettore.

Reti neurali per la riduzione della dimensionalità

Si supponga di avere un dataset costitituito da N vettori dell'insieme dei vettori $\mathbf{x}^n$ di caratteristiche d-dimensionali, con n=1,2, ...N. Si consideri, allora, una rete neurale a strati con due strati di pesi ed funzione di output dei nodi pari all'identità. La rete in esame puo' essere addestrata in modo tale che i valori di uscita dei noi interni rappresentino una versione ridotta dimensionalmente dei vettori $\mathbf{x}^n$ di ingresso. In particolare, si costruisca tale rete con d valori di ingresso, m < d nodi interni e d nodi di output. Tale rete può essere addestrata considerando come input un vettore di caratteristiche $\mathbf{x}^n$ e come target corrispondente $\mathbf{t}^n=\mathbf{x}^n$.

Tale rete è detta rete auto-associativa, in quanto vogliamo far si che risponda come output proprio con i valori dati in input. Una volta addestrata, ad ogni input $\mathbf{x}^n$ corrisponderà un vettore m-dimensionale $\mathbf{z}^n$ dato dai valori dei nodi interni.

Nella fase di addestramento l'errore da minimizzare è il seguente:

$$E=\frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{d}\left(y_k(x^n)-x_k^n\right)^2$$

dove

$y_k(x^n)=\sum_{h=0}^{m} w_{kh}^{(2)} z_h^n$ con $z_h^n=\sum_{i=0}^{d} w_{hi}^{(1)} x_i^n$ dove $\mathbf{w}^{(1)}$ e $\mathbf{w}^{(2)}$ rappresentato i pesi del primo e secondo strato, rispettivamente.

Si noti che in questo modo la funzione di errore risulta essere una funzione quadratica nei pesi, risultando così un funzione covessa ed avendo in questo modo un unico minimo globale. Questo assicura la convergenza al minimo globale con le classiche tecniche di discesa del gradiente.

Si può dimostrare (Bourland and Kamp, 1988; Baldi and Hornik, 1989), che quando si raggiunge tale minimo, gli m vettori dei pesi che portano alle unità interne, cioè

$$w_1^{(1)}=[w_{11}^{(1)}, w_{12}^{(1)}, \ldots, w_{1d}^{(1)}] \quad , \quad w_2^{(1)}=[w_{21}^{(1)}, w_{22}^{(1)}, \ldots, w_{2d}^{(1)}] \quad \mathbf{, \ldots,} \quad w_m^{(1)}=[w_{m1}^{(1)}, w_{m2}^{(1)}, \ldots, w_{md}^{(1)}]$$

formano una base per lo stesso sottospazio individuato dalle prime m componenti principali.

In altre parole, tali vettori di pesi NON corrispondono necessariamente alle componenti principali e NON ne godono le stesse proprietà di ortogonalità, tuttavia vanno ad individuare lo stesso sottospazio individuato dalle prime m componenti principali.

Si noti che nel caso di una rete auto-associativa abbiamo una sorta di apprendimento non supervisionato, in quanto non abbiamo bisogno di un target esplicito, ma il target è rappresentato dagli input stessi.