Introduzione alle Reti Neurali

Prof. Beatrice Lazzerini
Dipartimento di Ingegneria della Informazione
Via Diotisalvi, 2
56122 PISA

Reti Neurali Artificiali

Le reti neurali artificiali sono nate per riprodurre attività tipiche del cervello umano come la percezione di immagini, il riconoscimento di forme, la comprensione del linguaggio, il coordinamento senso-motorio, ecc.

A tale scopo si sono studiate le caratteristiche del cervello umano.

Nel sistema nervoso esistono miliardi di *neuroni* (cellule nervose). Un neurone è formato da un corpo cellulare e da molti prolungamenti ramificati, detti *dendriti*, attraverso i quali il neurone riceve segnali elettrici da altri neuroni. Ogni neurone ha anche un prolungamento filamentoso chiamato *assone*, la cui lunghezza può variare da circa 1 cm a qualche metro. All'estremità l'assone si ramifica formando terminali attraverso i quali i segnali elettrici vengono trasmessi ad altre cellule (ad esempio ai dendriti di altri neuroni). Tra un terminale di un assone e la cellula ricevente esiste uno spazio. I segnali superano questo spazio per mezzo di sostanze chimiche dette neurotrasmettitori. Il punto di connessione tra terminale e dendrite è detto *sinapsi*.

Un neurone si "attiva", cioè trasmette un impulso elettrico lungo il suo assone quando si verifica una differenza di potenziale elettrico tra l'interno e l'esterno della cellula. L'impulso elettrico provoca la liberazione di un neurotrasmettitore dai terminali dell'assone, che a loro volta possono, ad esempio, influenzare altri neuroni.

I neuroni biologici sono da 5 a 6 ordini di grandezza più lenti dei componenti elettronici convenzionali: un evento in un chip si verifica in alcuni nanosecondi (10^{-9} s) mentre un evento neurale in alcuni millisecondi (10^{-3} s).

Il cervello umano è un calcolatore complesso, non lineare e parallelo. Pur essendo costituito da elementi di elaborazione molto semplici (i neuroni), è in grado di eseguire computazioni complesse, come il riconoscimento, la percezione e il controllo del movimento, molte volte più velocemente del più veloce degli attuali calcolatori.

Il cervello è in grado di modificare le connessioni tra i neuroni in base all'esperienza acquisita, cioè è in grado di imparare.

Nel cervello non esiste un controllo centralizzato, nel senso che le varie zone del cervello funzionano insieme, influenzandosi reciprocamente e contribuendo alla realizzazione di uno specifico compito.

Infine, il cervello è *fault tolerant*, cioè se un neurone o una delle sue connessioni sono danneggiati, il cervello continua a funzionare, anche se con prestazioni leggermente degradate. In particolare, le prestazioni del processo cerebrale degradano gradualmente man mano che si distruggono sempre più neuroni (*graceful degradation*).

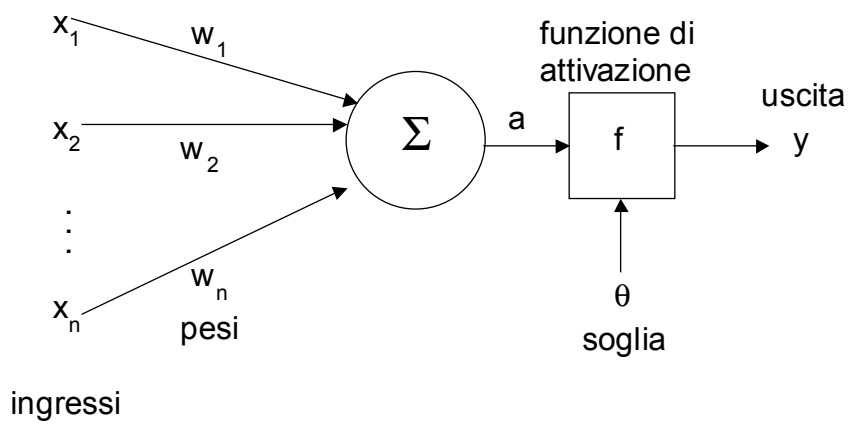
Quindi, per riprodurre artificialmente il cervello umano occorre realizzare una rete di elementi molto semplici che sia una struttura distribuita, massicciamente parallela, capace di apprendere e quindi di generalizzare (cioè produrre uscite in corrispondenza di ingressi non incontrati durante l'addestramento).

Tipicamente, il neurone artificiale ha molti ingressi ed una sola uscita. Ogni ingresso ha associato un peso, che determina la conducibilità del canale di ingresso. L'attivazione del neurone è una funzione della somma pesata degli ingressi.

Il metodo più usato per addestrare una rete neurale consiste nel presentare in ingresso alla rete un insieme di esempi (*training set*). La risposta fornita dalla rete per ogni esempio viene confrontata con la risposta desiderata, si valuta la differenza (errore) fra le due e, in base a tale differenza, si aggiustano i pesi. Questo processo viene ripetuto sull'intero training set finchè le uscite della rete producono un errore al di sotto di una soglia prestabilita.

Anche se di recente introduzione, le reti neurali trovano valida applicazione in settori quali predizione, classificazione, riconoscimento e controllo, portando spesso contributi significativi alla soluzione di problemi difficilmente trattabili con metodologie classiche.

MODELLO DI UN NEURONE



Abbiamo n canali di ingresso $x_1, \ldots, x_n$, a ciascuno dei quali è associato un peso. I pesi w_i sono numeri reali che riproducono le sinapsi. Se $w_i > 0$, il canale è detto *eccitatorio*, se $w_i < 0$, il canale è *inibitorio*. Il valore assoluto di un peso rappresenta la forza della connessione.

L'uscita, cioè il segnale con cui il neurone trasmette la sua attività all'esterno, è calcolata applicando la *funzione di attivazione* alla somma pesata degli ingressi. Indicando con $a = \sum_{i=1}^{n} w_i x_i$ la somma pesata degli ingressi, abbiamo:

$$y = f(a) = f\left(\sum_{i=1}^{n} w_i x_i\right).$$

Spesso, nella letteratura, la somma pesata degli ingressi è indicata con la parola *net*. Inoltre, la funzione di attivazione è detta anche *funzione di trasferimento*.

Nel modello di neurone rappresentato nella figura precedente è stata inclusa anche una *soglia* (*threshold*), che ha l'effetto di abbassare il valore in ingresso alla funzione di attivazione. Quindi, più correttamente, dobbiamo scrivere

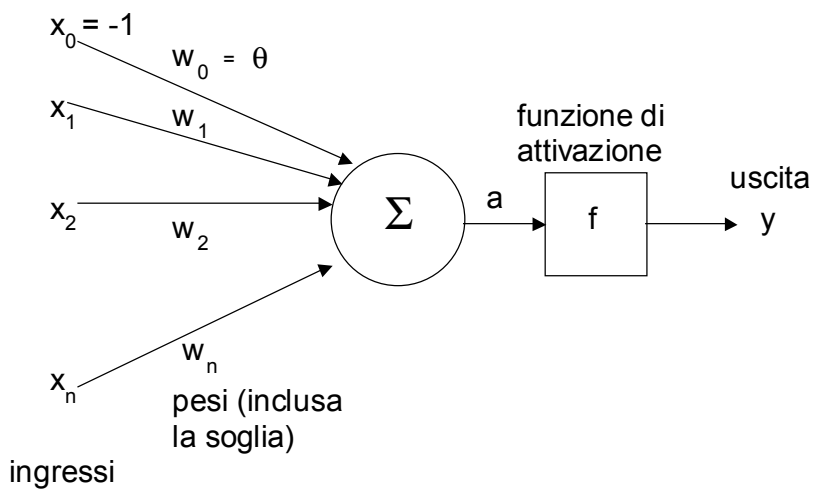
$$y = f(a) = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right).$$

In questo caso, interpretando la soglia come il peso associato ad un ulteriore canale di ingresso x_0, di valore sempre costante e pari a -1, potremmo anche scrivere

$$y = f(a) = f\left(\sum_{i=0}^{n} w_i x_i\right)$$

con $w_0 = \theta$.

Il modello di un neurone diventa quindi:



Osserviamo che in alcuni casi, invece di considerare la soglia, si considera l'opposto della soglia, detto *bias*, che può quindi essere visto come il peso associato ad un ulteriore canale di ingresso di valore costante pari a 1. Avremo ancora

$$y = f(a) = f(\sum_{i=0}^{n} w_i x_i)$$

dove $x_0 = 1, w_0 = b$ (*b* è il bias).

Funzione di attivazione

La funzione di attivazione definisce l'uscita di un neurone in funzione del livello di attivazione $a = \sum_{i=0}^{n} w_i x_i$.

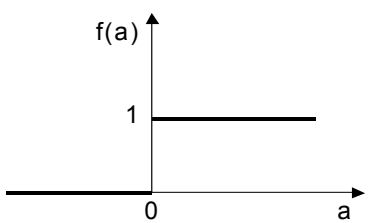
L'uscita può essere un numero reale, un numero reale appartenente ad un certo intervallo (ad esempio, [0,1]), oppure un numero appartenente ad un insieme discreto (tipicamente, {0,1} oppure {-1,+1}).

Vediamo alcuni esempi di funzione di attivazione.

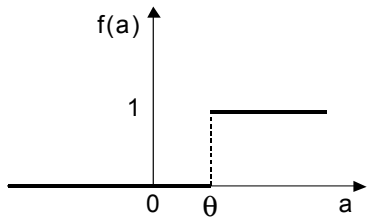
- **Funzione a soglia**

L'uscita di un neurone che usa una funzione di attivazione a soglia è

$$y = \begin{cases} 1 & \text{se } a \geq 0 \\ 0 & \text{se } a < 0 \end{cases}$$



Se si scorpora il contributo dell'ingresso x_0 dal livello di attivazione, cioè $a = \sum_{i=1}^{n} w_i x_i$, il diagramma diventa il seguente:

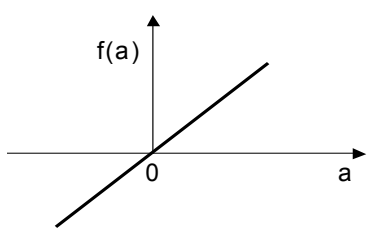


evidenziando così la notazione di soglia per il parametro θ:

$$y = \begin{cases} 1 & \text{se } \sum_{i=1}^{n} w_i x_i \geq \theta \\ 0 & \text{se } a < \theta \end{cases}$$

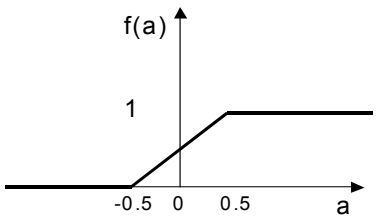
- **Funzione lineare**

f(a) = a



- **Funzione lineare a tratti**

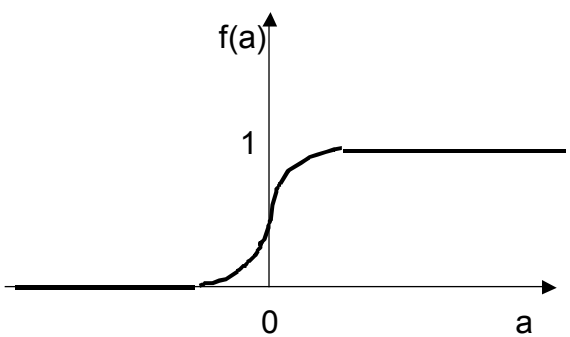
$$f(a)=\begin{cases} 0 & \text{se } a\le -0.5 \\ a+0.5 & \text{se } -0.5<a<0.5 \\ 1 & \text{se } a\ge 0.5 \end{cases}$$



- **Funzione sigmoide**

Assieme alla funzione di soglia, le funzioni sigmoidi sono tra le più usate. Un esempio di funzione sigmoide è la *funzione logistica*, definita come

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



Osserviamo che, mentre una funzione a soglia assume solo i valori 0 e 1, una funzione sigmoide assume tutti i valori da 0 a 1. Notiamo, inoltre, che la funzione sigmoide è derivabile, mentre la funzione a soglia non lo è (questo ci servirà nel seguito).

Le funzioni di attivazione viste finora assumono valori tra 0 e +1 (esclusa la funzione lineare). A volte è opportuno che la funzione di attivazione assuma valori tra –1 e +1. In particolare, la funzione a soglia è ridefinita così :

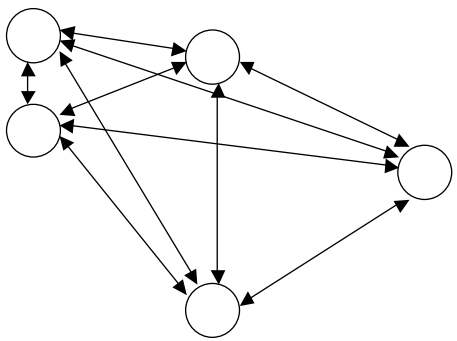
$$f(a)=\begin{cases} 1 & \text{se } a>0 \\ 0 & \text{se } a=0 \\ -1 & \text{se } a<0 \end{cases}$$

Tale funzione è nota come *funzione segno*.

ARCHITETTURA DI UNA RETE NEURALE

Si possono identificare più tipi di architettura di rete. Ne presentiamo due.

- **Reti completamente connesse (non stratificate)**



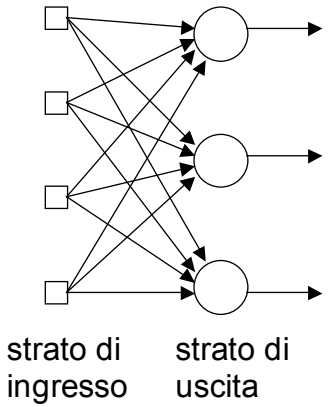
In una rete completamente connessa ogni neurone è connesso (in modo bidirezionale) con tutti gli altri.

Le connessioni tra i neuroni di una rete completamente connessa sono rappresentate mediante una matrice quadrata W, di dimensione pari al numero di neuroni, il cui generico elemento w_{ij} rappresenta il peso della connessione tra il neurone i ed il neurone j. Facciamo notare che alcuni autori usano, invece, la notazione w_{ji} per indicare il peso sulla connessione dal neurone i al neurone j.

- **Reti stratificate**

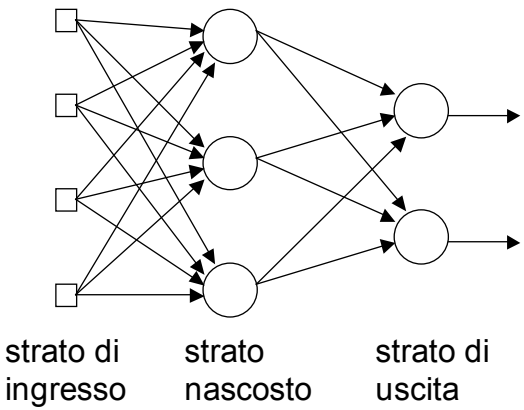
Nelle reti stratificate si individuano degli strati di neuroni tali che ogni neurone è connesso con tutti quelli dello strato successivo, ma non esistono connessioni tra i neuroni all'interno dello stesso strato, né tra neuroni di strati non adiacenti.

Il numero di strati ed il numero di neuroni per strato dipendono dallo specifico problema che si intende risolvere.



Dato che nello strato di ingresso non avviene alcuna computazione (i neuroni di ingresso devono semplicemente passare allo strato successivo i segnali ricevuti dall'ambiente esterno), la rete nella figura precedente viene di solito considerata come una rete con un solo strato. Inoltre, dato che i segnali viaggiano dallo strato di ingresso verso lo strato di uscita, si parla di rete *feedforward*.

Nella figura successiva viene mostrata una rete stratificata feedforward contenente uno *strato nascosto*, cioè uno strato i cui neuroni non comunicano direttamente con l'esterno. In generale, possono esserci uno o più strati nascosti. I neuroni nascosti permettono alla rete di costruire delle opportune rappresentazioni interne degli stimoli in ingresso in modo da facilitare il compito della rete.



Le connessioni tra i neuroni di una rete stratificata sono rappresentate mediante tante matrici quante sono le coppie di strati adiacenti. Ogni matrice contiene i pesi delle connessioni tra le coppie di neuroni di due strati adiacenti.

MODALITÀ DI ATTIVAZIONE DEI NEURONI

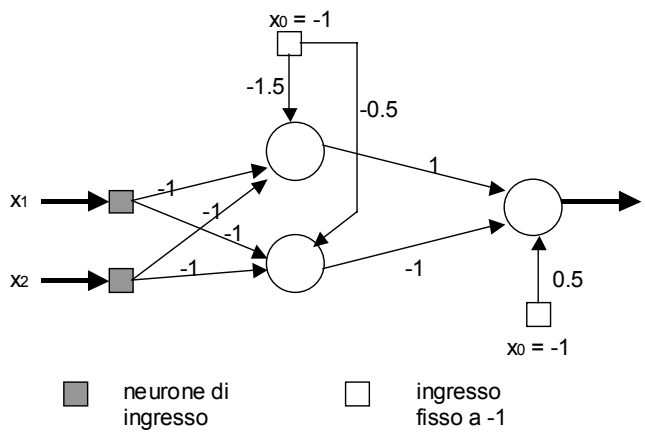
A seconda dei modelli di rete neurale, un solo neurone per volta può attivarsi ovvero tutti i neuroni possono attivarsi contemporaneamente. Nel primo caso si parla di *attivazione asincrona*, mentre nel secondo di *attivazione sincrona* o *parallela*. In particolare, nell'attivazione asincrona il neurone che può attivarsi è scelto in modo casuale.

Esempio

Costruiamo una rete neurale che calcola l'or esclusivo (XOR) che è così definito:

ingresso		
x1	x2	uscita
1	1	0
1	0	1
0	1	1
0	0	0

Consideriamo una rete con uno strato nascosto. Per calcolare l'uscita da ogni neurone dello strato nascosto e dello strato di uscita usiamo la funzione a soglia. (Ricordiamo che l'uscita di un neurone nello strato di ingresso coincide con il segnale in ingresso a tale neurone).



Diamo in ingresso alla rete la coppia [1 1]. Per il primo neurone nascosto con soglia -1.5 abbiamo:

$$a = (x_0 \times -1.5) + (x_1 \times -1) + (x_2 \times -1)$$
$$= (-1 \times -1.5) + (1 \times -1) + (1 \times -1) = -0.5$$

quindi l'uscita è 0. Per il secondo neurone nascosto con soglia -0.5 abbiamo:

$$a=(x_0\times-0.5)+(x_1\times-1)+(x_2\times-1)$$
$$=(-1\times-0.5)+(1\times-1)+(1\times-1)=-1.5$$

quindi l'uscita è 0. Per il neurone di uscita con soglia 0.5 abbiamo:

$$a=(x_0\times0.5)+(x_1\times1)+(x_2\times-1)$$
$$=(-1\times0.5)+(0\times1)+(0\times-1)=-0.5$$

quindi l'uscita è 0.

Facendo i calcoli con gli altri ingressi della tabella che definisce lo XOR, otteniamo le uscite indicate nella stessa tabella.

APPRENDIMENTO SUPERVISIONATO

Con riferimento all'esempio precedente, possiamo osservare che il corretto funzionamento della rete neurale dipende dall'architettura della rete (cioè dal numero di strati e dal numero di neuroni per strato), dalla funzione di attivazione dei neuroni e dai pesi. I primi due parametri sono fissati prima della fase di addestramento. Il compito dell'addestramento è quindi quello di aggiustare i pesi in modo che la rete produca le risposte desiderate.

Uno dei modi più usati per permettere ad una rete di imparare è l'*apprendimento supervisionato*, che prevede di presentare alla rete per ogni esempio di addestramento la corrispondente uscita desiderata.

Di solito i pesi vengono inizializzati con valori casuali all'inizio dell'addestramento. Poi si cominciano a presentare, uno alla volta, gli esempi costituenti l'insieme di addestramento (*training set*). Per ogni esempio presentato si calcola l'errore commesso dalla rete, cioè la differenza tra l'uscita desiderata e l'uscita effettiva della rete. L'errore è usato per aggiustare i pesi.

Il processo viene di solito ripetuto ripresentando alla rete, in ordine casuale, tutti gli esempi del training set finchè l'errore commesso su tutto il training set (oppure l'errore medio sul training set) risulta inferiore ad una soglia prestabilita.

Dopo l'addestramento la rete viene testata controllandone il comportamento su un insieme di dati, detto *test set*, costituito da esempi

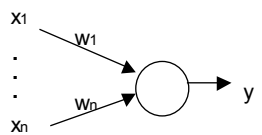
non utilizzati durante la fase di training. La fase di test ha quindi lo scopo di valutare la capacità di generalizzazione della rete neurale.

Diremo che la rete ha imparato, cioè è in grado di fornire risposte anche per ingressi che non le sono mai stati presentati durante la fase di addestramento.

Ovviamente le prestazioni di una rete neurale dipendono fortemente dall'insieme di esempi scelti per l'addestramento. Tali esempi devono quindi essere rappresentativi della realtà che la rete deve apprendere e in cui verrà utilizzata. L'addestramento è in effetti un processo ad hoc dipendente dallo specifico problema trattato.

Delta rule

Riferiamoci al neurone rappresentato di seguito:



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

$$\delta=t-y.$$

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

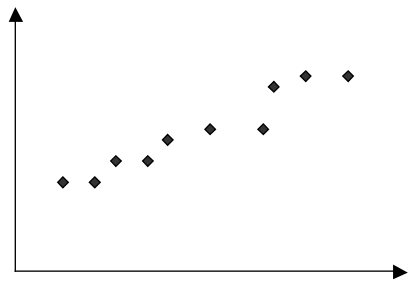
$$\Delta w_i=\eta\delta x_i$$

dove η è un numero reale compreso tra 0 e 1 detto *learning rate*. Il learning rate determina la velocità di apprendimento del neurone. La delta rule modifica in maniera proporzionale all'errore solo i pesi delle connessioni che hanno contribuito all'errore (cioè che hanno $x_i\neq0$). Al contrario, se $x_i=0$, w_i non viene modificato poiché non si sa se ha contribuito all'errore. Il nuovo valore dei pesi è quindi:

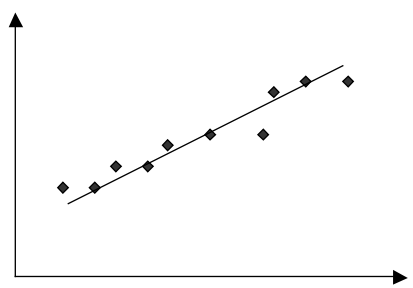
$$w_i=w_i+\Delta w_i$$

Esempio di addestramento

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



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



La retta disegnata può essere utile per ricavare (o prevedere) valori della variabile dipendente (rappresentata sull'asse delle ordinate) in corrispondenza di valori della variabile indipendente per cui non sono state fatte misurazioni.

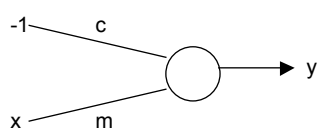
Considerando che l'equazione di una retta in forma esplicita è

$$y = mx + c$$

dove y ed x sono variabili, m la pendenza e c il punto in cui la retta incontra l'asse y, possiamo calcolare m e c con il metodo dei minimi quadrati risolvendo le equazioni ottenute uguagliando a 0 le derivate parziali (rispetto ad m e c, rispettivamente) della somma degli errori quadratici.

In alternativa al metodo dei minimi quadrati si può usare una rete neurale che approssima una retta. In ingresso alla rete vengono presentati i punti da approssimare con la linea retta e si lascia alla rete il compito di apprendere.

Possiamo usare una rete (rappresentata nella figura seguente) costituita da un neurone di ingresso ed un neurone di uscita con funzione di attivazione lineare. Dato che la rete deve stimare m e c, m e c costituiscono i pesi. Tali pesi saranno inizializzati in modo casuale. Gli esempi che costituiscono il training set sono le coppie (x, y) delle coordinate dei punti da approssimare con la linea retta; ovvero l'ascissa rappresenta l'ingresso e l'ordinata l'uscita desiderata. Il peso c è la soglia, quindi è associato ad un ingresso costantemente uguale a -1.



La rete viene addestrata usando la delta rule con un certo learning rate (ad esempio, $\eta = 0.1$). I punti del training set saranno presentati un certo numero di volte, finchè l'errore commesso dalla rete non scende al di sotto di una soglia prestabilita.

L'uscita della rete produrrà una retta del tutto simile a quella generata col metodo dei minimi quadrati.

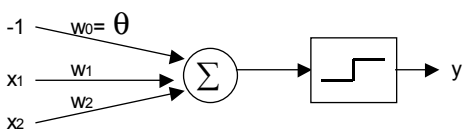
Diremo che la rete ha imparato perché dandole in ingresso l'ascissa di un punto non usato durante l'addestramento la rete produrrà in uscita l'ordinata corrispondente.

CLASSIFICAZIONE

In molte applicazioni si incontrano problemi di classificazione di un insieme di oggetti, cioè occorre associare ogni oggetto alla classe corretta.

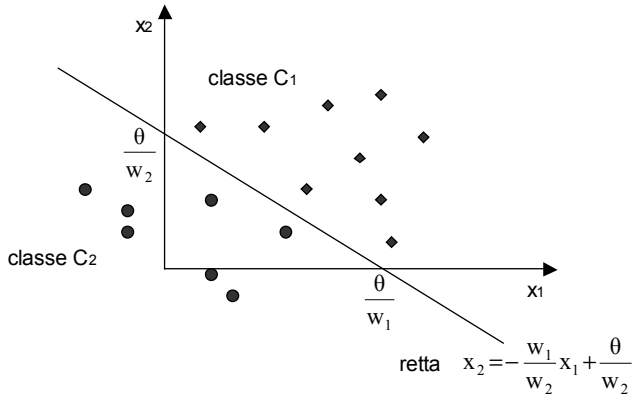
Supponiamo di voler classificare in due classi distinte oggetti rappresentati mediante punti nel piano. Se le due classi sono linearmente separabili, possiamo usare una rete neurale che approssima una retta di separazione tra le due classi. Un oggetto sarà quindi classificato rappresentandolo come punto nel piano ed assegnandolo a quella delle due classi individuata dal semipiano in cui cade il punto.

Tale classificazione può essere facilmente ottenuta addestrando una rete neurale con due ingressi ed un solo neurone di uscita con funzione di attivazione a soglia:



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

Consideriamo, ad esempio, la figura seguente:



Possiamo associare la classe C_1 all'insieme di stimoli per cui la rete risponde con y=1 e C_2 all'insieme di stimoli per cui la rete risponde con y=0, cioè:

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

Nel piano (x_1,x_2) degli ingressi della rete le classi C_1 e C_2 sono rappresentate da due semipiani separati dalla retta

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

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

1. si inizializzano i pesi w_i con valori casuali;
2. si presenta alla rete un ingresso x_k insieme al valore t_k desiderato in uscita;
3. si calcola la risposta y_k della rete e si aggiornano i pesi mediante la delta rule;

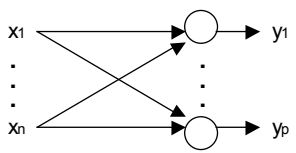
4. si ripete il ciclo dal passo 2, finchè la risposta della rete non risulti soddisfacente.

Con riferimento all'ultima figura, osserviamo che il processo di apprendimento, modificando i pesi w_1, w_2 e θ, non fa altro che modificare la posizione e la pendenza della retta di separazione tra le due classi. Il processo termina quando la retta separa correttamente le due classi.

Infine, osserviamo che la rete considerata nell'esempio è un perceptron con due ingressi perché gli oggetti da classificare sono rappresentati come punti in $\Re^2$. Tali punti sono separati da una retta. In generale, se gli oggetti da classificare sono vettori *n*-dimensionali, gli ingressi del perceptron sono *n* e le due classi sono separate da un iperpiano in $\Re^n$.

Aggiornamento dei pesi e convergenza della delta rule

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



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

$$E_k = \frac{1}{2}\sum_{j=1}^{p}(t_j - o_j)^2 \qquad (1)$$

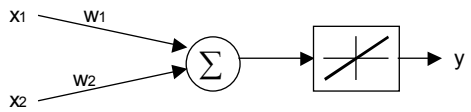
(il fattore ½ è stato introdotto per semplificare le notazioni)

per cui l'errore globale commesso dalla rete su tutto il training set costituito da *m* esempi è $E = \sum_{k=1}^{m} E_k$.

Per illustrare il razionale della delta rule, verrà preso in esame un caso semplice di comportamento lineare; anche se analiticamente tale situazione

può essere affrontata con metodi diretti, faremo riferimento alla procedura iterativa della delta rule.

Consideriamo allora un singolo neurone, rappresentato nella figura seguente, con due ingressi, soglia = 0 e funzione di attivazione lineare (cioè l'uscita coincide con l'ingresso: f(a)=a).



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

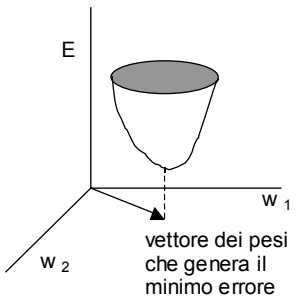
$$E = \frac{1}{2}(t-a)^2$$

Considerando che $a = x_1 w_1 + x_2 w_2$ e svolgendo i calcoli otteniamo:

$$E = \frac{1}{2}[t^2 - 2ta + a^2] = \\ = \frac{1}{2}[t^2 - 2t(x_1 w_1 + x_2 w_2) + x_1^2 w_1^2 + 2x_1 w_1 x_2 w_2 + x_2^2 w_2^2] \qquad (2)$$

Abbiamo ricavato che l'errore E, nel caso di neuroni lineari, è un paraboloide nello spazio dei pesi. In generale, per altri tipi di neuroni, sarà un'ipersuperficie nello spazio dei pesi.

Prima di iniziare l'addestramento, i pesi sono inizializzati a valori casuali quindi il punto che rappresenta lo stato iniziale della rete può trovarsi ovunque sulla superficie dell'errore (in generale non coinciderà con il punto di minimo di tale superficie). Durante l'addestramento i pesi dovranno essere modificati in modo da far muovere lo stato della rete lungo una direzione, che la delta rule individuerà essere quella di massima pendenza, della superficie dell'errore in modo da minimizzare l'errore globale.



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

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

dove t_j è l'uscita desiderata dal neurone j, o_j l'uscita effettiva, x_i il segnale proveniente dal neurone i, η il learning rate e Δw_{ij} la variazione del peso sulla connessione da i a j.

Vogliamo dimostrare che, aggiornando i pesi mediante la delta rule, l'apprendimento converge verso una configurazione dei pesi che minimizza l'errore quadratico globale.

Osserviamo che per dimostrare la convergenza della delta rule basterebbe dimostrare che tale regola è riconducibile alla forma

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

la quale varierebbe i pesi in modo da favorire la diminuzione dell'errore. Infatti:

- se E cresce all'aumentare di w_{ij} (cioè $\frac{\partial E}{\partial w_{ij}} > 0$) allora w_{ij} viene diminuito per contrastare la crescita di E ($\Delta w_{ij} < 0$)
- se E diminuisce all'aumentare di w_{ij} (cioè $\frac{\partial E}{\partial w_{ij}} < 0$) allora w_{ij} viene aumentato per favorire la diminuzione di E ($\Delta w_{ij} > 0$).

Per semplicità, consideriamo un neurone lineare con uscita definita da

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

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

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

Dalla (1) e dalla definizione di δ_j otteniamo

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

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

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

A questo punto, inserendo il learning rate, otteniamo proprio la delta rule.

Con riferimento alla figura che mostra la superficie dell'errore, osserviamo che il processo di apprendimento può essere interpretato come una discesa su tale superficie lungo la linea di massima pendenza, individuata appunto dal gradiente $-\nabla E = \left(-\frac{\partial E}{\partial w_{ij}} \right)$.

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

Osservazione

La delta rule, che abbiamo ricavato riferendoci per semplicità ai neuroni lineari, è in realtà valida per qualsiasi tipo di neurone. Si parla, in effetti, di delta rule nel caso di neuroni lineari e di *delta rule generalizzata* per altri tipi di neuroni, tenendo però conto della eventuale forma della funzione dell'errore da minimizzare.

Problemi lineari e non lineari

Un problema di classificazione in cui si devono separare in due classi i punti appartenenti ad un certo insieme si dice *lineare* se una linea (in due dimensioni) o un iperpiano (in *n* dimensioni) possono separare correttamente tutti i punti. In caso contrario, il problema si dice *non lineare*.

Abbiamo visto che un problema lineare può essere risolto usando un perceptron. Infatti, un perceptron con *n* ingressi è in grado di rappresentare un iperpiano *n*-dimensionale. Quindi, un perceptron è in grado di risolvere problemi linearmente separabili in cui gli ingressi devono essere catalogati in due differenti classi separabili tramite una retta (perceptron a due ingressi), un piano (perceptron a tre ingressi), o un iperpiano (perceptron a *n* ingressi).

Un tipico problema non lineare è l'or esclusivo (XOR). Tale operatore, infatti, produce 1 in uscita solo quando uno solo degli ingressi vale 1, altrimenti dà 0. Non esiste alcuna retta che separi i punti (0,1) e (1,0) dai punti (0,0) e (1,1). Per risolvere questo problema si hanno due possibilità:

1) si ricorre a particolari funzioni di uscita non lineari,

2) si usano reti con più strati.

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

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

Ovviamente, questo approccio può essere difficile da perseguire qualora la funzione non lineare richiesta sia difficile da individuare.

Nel secondo approccio, si usa una rete con uno o più strati nascosti in modo da modellare due o più rette per separare i dati. Tale rete è detta *multilayer perceptron*.

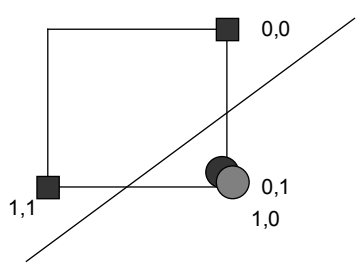
Nel caso dello XOR la rete può essere quella già vista precedentemente e contenente due neuroni nascosti che rappresentano due rette e un neurone di uscita che combina le informazioni prodotte dalle due rette.

Con riferimento alla rete già vista per risolvere lo XOR, consideriamo la seguente tabella che riporta gli ingressi ai neuroni nascosti e le relative uscite.

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

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

La figura seguente rappresenta l'uscita dallo strato nascosto.

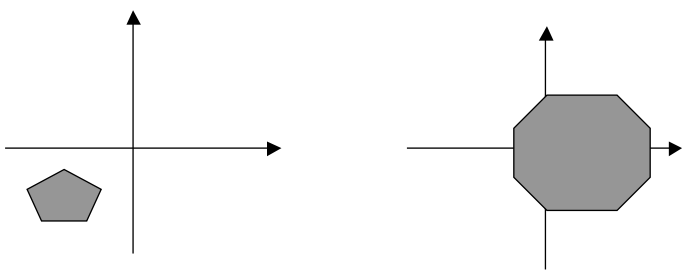


Anche il secondo livello di pesi (che connettono lo strato nascosto con lo strato di uscita) modella una retta. Affinché la rete produca gli output desiderati, occorre quindi che le configurazioni degli input al secondo livello di pesi siano separabili da una retta, come confermato dalla figura precedente.

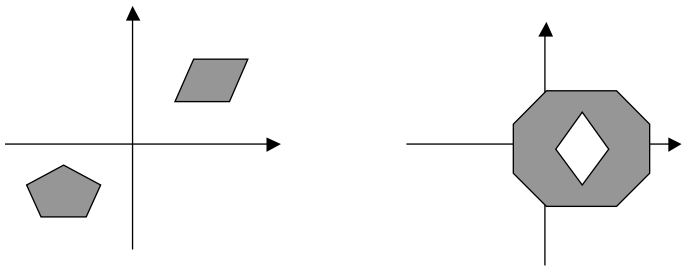
I pesi che definiscono la retta di separazione disegnata nella figura precedente sono quelli già visti quando abbiamo parlato per la prima volta del problema dello XOR.

Osservazioni

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



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



La maggior parte dei problemi è risolvibile adottando reti con al massimo due strati nascosti.

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

BACKPROPAGATION

La rete multistrato che abbiamo adottato precedentemente per risolvere il problema dello XOR è stata costruita definendo i pesi appropriati. Ovviamente, ciò che ci interessa trovare è un algoritmo di addestramento supervisionato che permetta alla rete multistrato di trovare da sola i pesi.

Il problema incontrato nell'addestramento delle reti multistrato è il seguente: volendo adottare un meccanismo di aggiornamento dei pesi

simile alla delta rule (in cui l'errore è calcolato come differenza tra l'uscita desiderata e l'uscita effettiva di ciascun neurone) si riesce ad aggiornare solo i pesi relativi ai neuroni di uscita, ma non quelli relativi ai neuroni degli strati nascosti. Infatti, mentre per lo strato di uscita si conosce l'uscita desiderata (tale uscita viene data come secondo elemento delle coppie che costituiscono gli esempi del training set), niente si sa dell'uscita desiderata dai neuroni nascosti.

Questo problema è stato risolto dopo molti anni di disinteresse per le reti neurali (in quanto non si riusciva ad addestrarle) solo nel 1986, quando fu introdotto l'algoritmo di *backpropagation*. Tale algoritmo prevede di calcolare l'errore commesso da un neurone dell'ultimo strato nascosto propagando all'indietro l'errore calcolato sui neuroni di uscita collegati a tale neurone. Lo stesso procedimento è poi ripetuto per tutti i neuroni del penultimo strato nascosto, e così via.

L'algoritmo di backpropagation prevede che, per ogni esempio del training set, i segnali viaggino dall'ingresso verso l'uscita al fine di calcolare la risposta della rete. Dopo di che c'è una seconda fase durante la quale i segnali di errore vengono propagati all'indietro, sulle stesse connessioni su cui nella prima fase hanno viaggiato gli ingressi, ma in senso contrario, dall'uscita verso l'ingresso. Durante questa seconda fase vengono modificati i pesi.

I pesi sono inizializzati con valori casuali. Come funzione di uscita non lineare dei neuroni della rete si adotta in genere la funzione sigmoide (l'algoritmo richiede che la funzione sia derivabile). Tale funzione produce valori tra 0 e 1.

L'algoritmo di backpropagation usa una generalizzazione della delta rule.

Nel seguito useremo *net* per indicare la somma pesata degli ingressi di un neurone.

Possiamo esprimere la derivata dell'errore come segue:

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

Poniamo

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

(osserviamo che questa definizione coincide con la delta rule data nelle pagine precedenti. Lì, infatti, avevamo $\delta_j = -\frac{\partial E}{\partial o_j}$ perché consideravamo neuroni lineari, cioè $o_j = net_j$).

Riscriviamo l'equazione precedente:

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

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

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

abbiamo

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

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

$$o_j = f(net_j)$$

e quindi

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

da cui

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

Essendo

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

si ha

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

per cui, tornando alla formula di partenza risulta

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

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

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

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

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

dove *s* è l'indice dei neuroni dello strato che trasmette all'indietro l'errore.

Come detto, una funzione spesso usata è la funzione logistica

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

La derivata di tale funzione è:

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

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

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

Osservazione

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

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

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

Algoritmo di backpropagation

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

1. si inizializzano i pesi con valori casuali (in genere, con valori non troppo elevati);
2. si presenta un ingresso X_k e si calcolano le uscite $o_j = \frac{1}{1 + \exp(-net_j)}$ di tutti i neuroni della rete;

3. dato T_k, si calcolano l'errore δ_j e la variazione dei pesi Δw_{ij} per ogni neurone dello strato di uscita:

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

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

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

5. per tutti gli strati aggiornare i pesi:

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

6. ripetere il processo dal punto 2 finchè non si siano presentati tutti gli *m* esempi del training set;

7. si calcola l'errore medio sul training set (oppure l'errore globale); se l'errore è al di sotto di una soglia prefissata, l'algoritmo termina (si è raggiunta la convergenza), altrimenti si ripete un intero ciclo di presentazione del training set.

Un ciclo di presentazione degli esempi del training set è detto *epoca*.

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

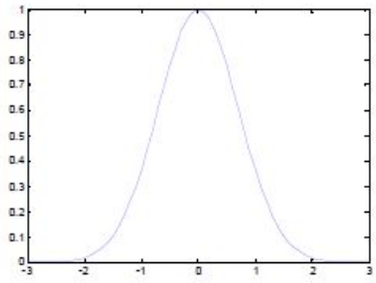
2 Artificial Neural Networks pt.2

2.1 Radial functions

Le *radial functions* sono una speciale classe di funzioni: la loro risposta decrementa (o aumenta) monotonicamente con la distanza dal punto centrale.

Una tipica radial function è la Gaussiana la quale, nel caso di un input scalare, è:

$$h(x) = exp\left(-\frac{(x-t)^2}{\sigma^2}\right)$$



I suoi parametri sono il *centro* t e il suo *raggio* (o *spread*) σ.
Quindi una radial function $h(x) : \mathbb{R}^n \to \mathbb{R}$ è una funzione definita come:

$$h(\mathbf{x}) = \varphi\left(\|\mathbf{x} - \mathbf{t}\|\right)$$

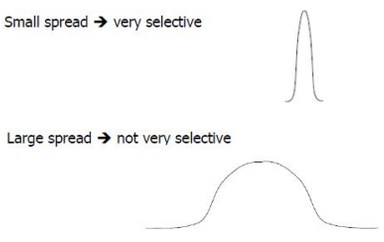
dove $\varphi(r) : \mathbb{R} \to \mathbb{R}, \mathbf{x}$ è l'input, $\mathbf{t}$ è il centro $\|\cdot\|$ è la norma Euclidea

Esempio: Gaussiana

$$h(\mathbf{x}) = exp\left(-\frac{(\mathbf{x}-\mathbf{t})^2}{\sigma^2}\right)$$

Dove σ è il parametro (**spread**) che indica la **selettività** della funzione.

$$Spread = \frac{1}{Selectivity}$$

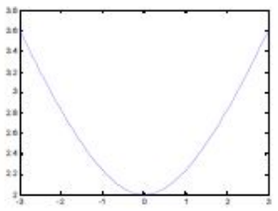


I tipi di RBF includono:

- Multiquadrics

$$\varphi = \sqrt{(r^2 + c^2)}$$

$$c > 0 \ (shapeparameter), \quad r = \|x - t\|$$



- Inverse multiquadircs

$$\varphi = \frac{1}{\sqrt{(r^2 + c^2)}}$$

- Gaussian

$$\varphi = exp\left(-\frac{r^2}{c^2}\right)$$

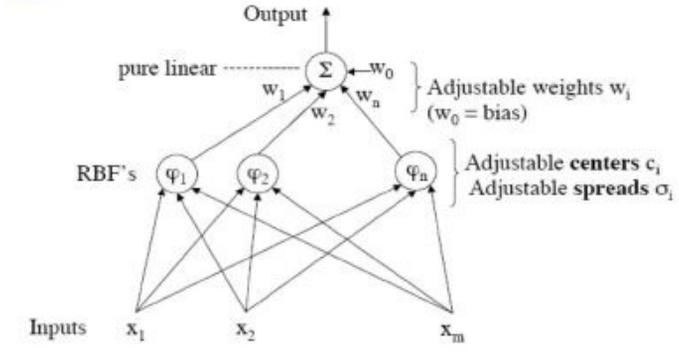
2.2 Rete RBF

Una rete RBF nella sua forma più semplice è una rete two-layered:

- L'input layer all'hidden layer è **non lineare**
- L'hidden layer all'output layer è **lineare**

Tipicamente (ma non sempre) # hidden units > # input units.

L'idea è quella che se un problema non lineare è posto in uno spazio ad alta dimensionalità in maniera non lineare allora è più probabile che esso sia linearmente separabile.

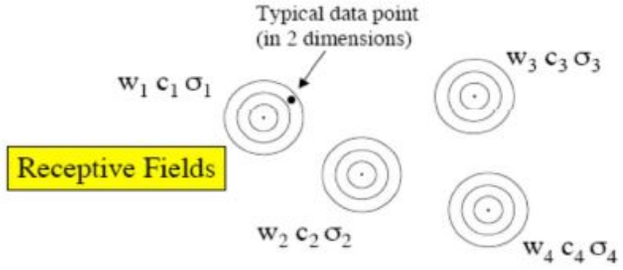


I pesi connessi ad una hidden unit definiscono il centro della radial basis function per tale hidden unit.

L'input della rete ad una hidden unit è la norma Euclidea.

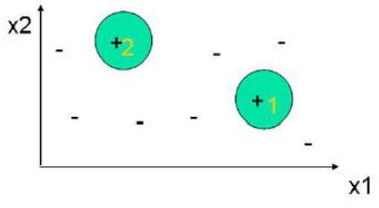
$$net_j = \|\mathbf{x} - \mathbf{w}_j\| = \left(\sum_{i=1}^{m} (x_i - w_{ij})^2 \right)^{\frac{1}{2}}$$

$$Output = \sum w_j \varphi_i(\mathbf{x}) \quad where\ \mathbf{x}\ is\ the\ input\ vector$$



Una funzione è approssimata come una combinazione lineare di funzioni radial-basis. Una RBF risponde solo ad una piccola regione dell'input space dove la funzione stessa è centrata. Una rete RBF può essere usata non solo per il function fitting ma anche per la classificazione.

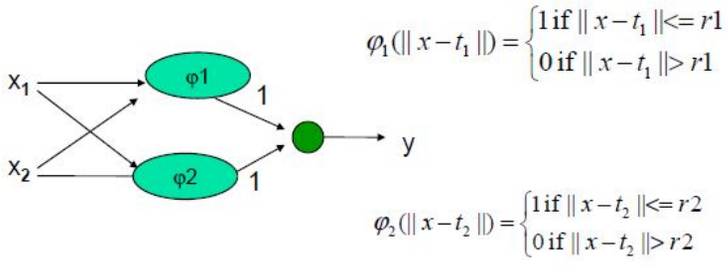
Esempio 1

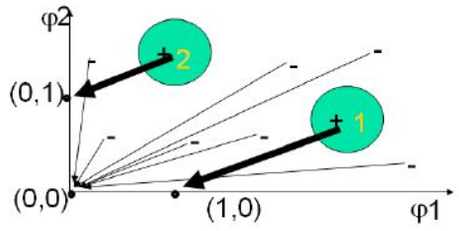


Gli esempi all'interno dei cerchi 1 e 2 sono di classe +, mentre quelli fuori dai cerchi sono di classe −.

Possiamo separare le due classi usando una rete RBF?

Una soluzione: Scegliamo come centri t_1, t_2 i centri delle due circonferenze. Siano r_1,r_2 i raggi delle due circonferenze, e $x = (x_1, x_2)$, un esempio è:



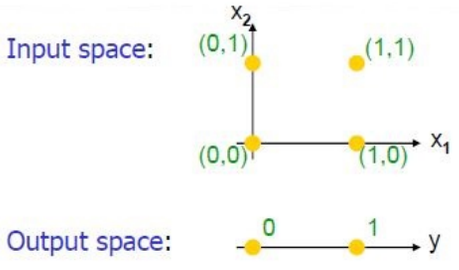


Geometricamente: gli esempi sono mappati dall'input space $< x1, x2 >$ al feature space $< \varphi_1, \varphi_2 >$:

- Gli esempi nel cerchio 1 sono mappati nel punto (1,0)
- Gli esempi del cerchio 2 sono mappati in (0,1)
- Gli esempi fuori da entrambi i cerchi sono mappati in (0,0)

Le due classi diventano linearmente separabili nel feature space $< \varphi_1, \varphi_2 >$.

Esempio 2: lo XOR problem



Dobbiamo creare un classificatore RBF di pattern con delle funzioni di attivazione Gaussiane in modo che:

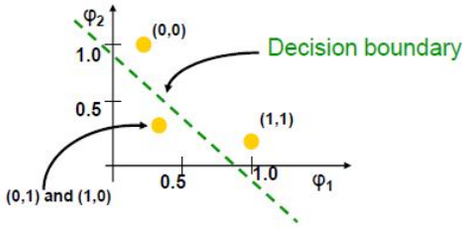
- (0,0) e (1,1) sono mappati in 0, classe C1
- (1,0) e (0,1) sono mappati in 1, classe C2

Soluzionen: Scegliamo i centri

$$t_1 = (1,1) \quad t_2 = (0,0)$$

$$\varphi_1(\| x - t_1 \|) = e^{-\|x-t_1\|^2}$$
$$\varphi_2(\| x - t_2 \|) = e^{-\|x-t_2\|^2}$$

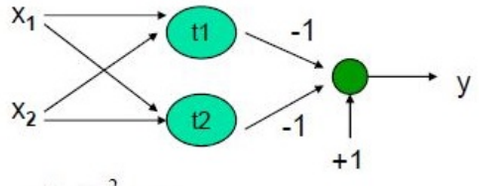
Input x	$\varphi_1(x)$	$\varphi_2(x)$
(1,1)	1	0.1353
(0,1)	0.3678	0.3678
(1,0)	0.3678	0.3678
(0,0)	0.1353	1



Quando sono mappati nel feature space (hidden layer), C1 e C2 diventano linearmente separabili.

Quindi un classificatore lineare con $\varphi_1(x)$ e $\varphi_2(x)$ come input può essere usato per risolvere lo XOR problem.

$$\varphi_1(\| x - t_1 \|) = e^{-\|x-t_1\|^2} \quad \text{with } t_1 = (1,1) \text{ and } t_2 = (0,0)$$
$$\varphi_2(\| x - t_2 \|) = e^{-\|x-t_2\|^2}$$



$$y = -e^{-\|x-t_1\|^2} - e^{-\|x-t_2\|^2} + 1$$

Se y>0 allora è classe 1, altrimenti classe 0.

2.3 Determinazione dei parametri

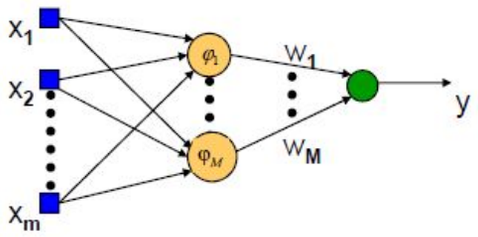
Che parametri dobbiamo usare per il learning di una rete RBF con una architettura data e funzioni di attivazione Gaussiane?

- I **centri** delle funzioni di attivazione RBF

- Gli **spreads** delle funzioni di attivazione RBF Gaussiane
- I **pesi** dall'hidden all'output layer

2.4 Strategie di learning

I differenti layer di una rete RBF eseguono tasks differenti, e quindi è ragionevole separare l'ottimizzazione dell'hidden e dell'output layer della rete usando tecniche differenti. Ci sono molteplici strategie di learning a seconda di come i centri delle funzioni radial-basis sono specificati. Essenzialmente, possiamo identificare tre approcci:



2.4.1 Algoritmo di learning 1: Centri fissi selezionati casualmente

La locazione dei **centri** può essere scelta randomicamente dal training data set. Gli **spread** vengono scelti tramite *normalizzazione*. Più precisamente, la deviazione standard delle funzioni Gaussiane è fissata tramite la normalizzazione in accordo allo spread dei centri

$$\sigma = \frac{d}{\sqrt{M}}$$

dove M è il numero di centri e d è la distanza massima tra i centri scelti. Tale scelta per la standard deviation assicura soltanto che le funzioni Gaussiane non siano troppo aguzze o troppo piatte; entrambi questi estremi devono essere evitati.

Quindi la funzione di attivazione per gli hidden neurons diventa:

$$\varphi\left(\|\mathbf{x} - \mathbf{t}_i\|\right) = exp\left(-\frac{M}{d^2}\|\mathbf{x} - \mathbf{t}_i\|^2\right), \qquad i = 1, ..., M$$

I **pesi** lineari nell'output layer sono calcolati tramite il *metodo dello pseudo-inverso*:

- Per un example (x_i, d_i) consideriamo l'ouptut della rete

$$y(x_i) = w_1\varphi_1(\|x_i - t_1\|) + ... + w_M\varphi_M(\|x_i - t_M\|)$$

- Noi vorremmo $y(x_i) = d_i$ per ogni example, che è

$$w_1\varphi_1(\|x_i - t_1\|) + ... + w_M\varphi_M(\|x_i - t_M\|) = d_i$$

Questo può essere riscritto in forma matriciale per un example

$$[\varphi_1(\|x_i - t_1\|)...\varphi_M(\|x_i - t_M\|)][w_1...w_M]^T = d_i$$

e

$$\begin{bmatrix} \varphi_1(\| x_1 - t_1 \|)...\varphi_M(\| x_1 - t_M \|) \\ ... \\ \varphi_1(\| x_N - t_1 \|)...\varphi_M(\| x_N - t_M \|) \end{bmatrix} [w_1...w_M]^T = [d_1...d_N]^T$$

Per tutti gli N examples allo stesso istante.
sia

$$\mathbf{\Phi} = \begin{bmatrix} \varphi_1(\| x_1 - t_1 \|) & ... & \varphi_M(\| x_1 - t_M \|) \\ & ... & \\ \varphi_1(\| x_N - t_1 \|) & ... & \varphi_M(\| x_N - t_M \|) \end{bmatrix}$$

Allora possiamo scrivere

$$\mathbf{\Phi} \begin{bmatrix} w_1 \\ ... \\ w_M \end{bmatrix} = \begin{bmatrix} d_1 \\ ... \\ d_N \end{bmatrix}$$

Se Φ^+ è la pseudo-inversa della matrice Φ otteniamo i pesi usando la seguente formula:

$$[w_1...w_M]^T = \Phi^+[d_1...d_n]^T$$

2.5 Algoritmo di learning 2: Selezione auto-organizzata dei centri

Viene utilizzato un algoritmo di clustering per trovare i **centri**. Ciò permette di piazzare i centri delle funzioni radial-basis solo in quelle regioni dell'input space dove sono presenti dati significativi. Gli **spreads** sono scelti tramite la normalizzazione.

Viene applicata una supervised learning rule (ad esempio *l'algoritmo least mean square (LMS)*) per calcolare i **pesi** lineari dell'output layer. Gli output delle hidden units nella rete RBF servono come input dell'algoritmo LMS.

2.6 Algoritmo di learning 3: Selezione supervisionata dei centri

Nel terzo approccio, le formule di aggiornamento sono usata per addetrare contemporaneamente centri, pesi e spreads iterativamente usando il gradient descent.

Nella pratica, l'errore quadratico $E = \frac{1}{2}\sum_{j=1}^{N} e_j^2$ è minimizzato, dove N è il numero dei training examples e e_j è il segnale dell'errore, definito come

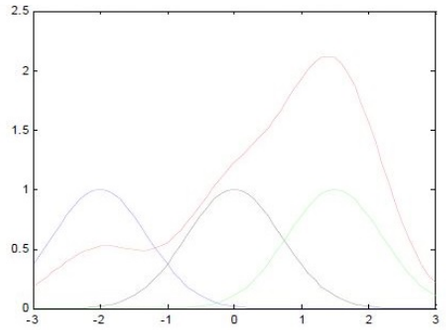
$$e_j = d_j - y(x_j) = d_j - \sum_{i=1}^{M} w_i \varphi_i \left(\|x_j - t_i\| \right)$$

Il requisito è quello di trovare i parametri liberi in modo che minimizzino E. Il risultato di tale minimizzazione sono gli aggiornamenti per:

- Centri: $\Delta t_j = -\eta_{t_j} \frac{\partial E}{\partial t_j}$
- Spread: $\Delta \sigma_j = -\eta_{\rho_j} \frac{\partial E}{\partial \sigma_j}$
- Pesi: $\Delta \omega_{ij} = -\eta_{ij} \frac{\partial E}{\partial \omega_{ij}}$

```
p=[-3:.1:3];
f1=radbas(p);
f2=radbas(p-1.5);
f3=radbas(p+2);
f=f1+f2*2+f3*0.5;
```

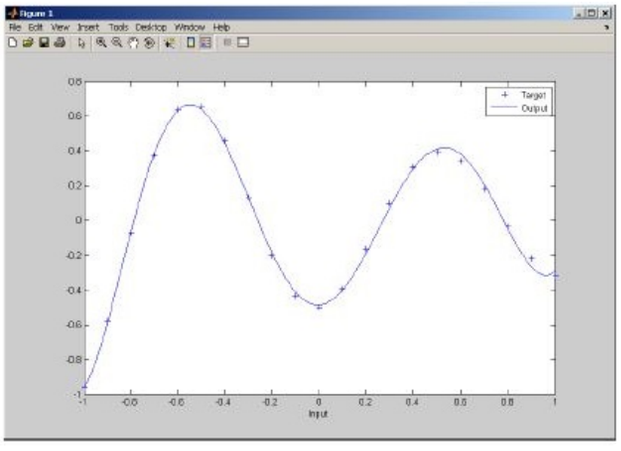
$radbas(n) = e^{-n^2}$



An RBF network that approximates a function defined by 21 data points (-1:.1:1)

First case

spread constant = 1

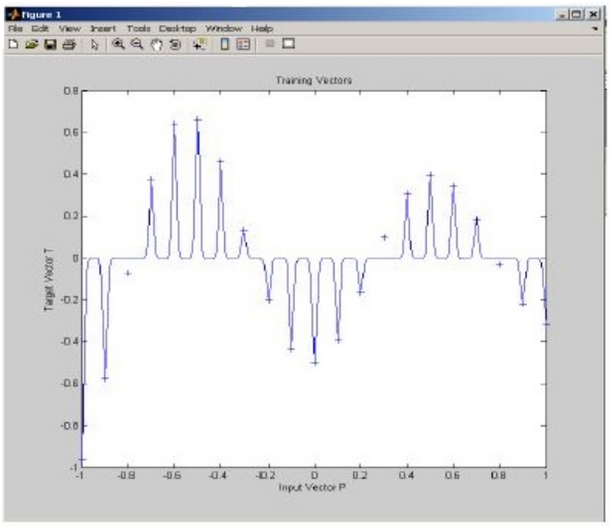


Second case

spread constant = .01 ➔ no two RBF neurons have a strong output for any given input

In other words, the solution does not generalize from the input/output vectors

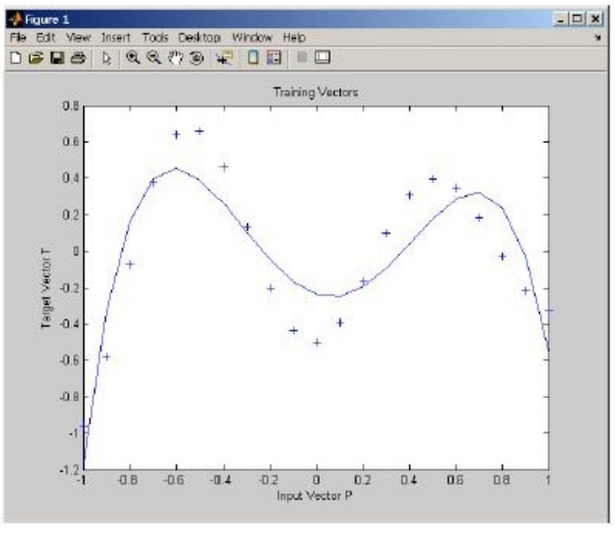
➔ **overfitting**



Third case

spread constant =100 ➔ the RBF neurons will output large values (≈1) for all the inputs used to design the network

All the neurons always output 1, so they cannot be used to generate different responses



Morale della storia: scegliere uno spread più largo della distanza tra vettori di input adiacenti, in modo da avere una buona generalizzazione, ma più piccolo della distanza attraverso l'intero input space.

2.7 Confronto tra reti RBF e Multilayer Perceptrons

Le reti RBF e i multilayer perceptrons (MLPs) sono esempi di reti layered feedforward non lineari. Entrambe sono **approssimatori universali**: esiste sempre una rete RBF capace di imitare una specifica MLP, o viceversa. Comunque sia, queste due reti differiscono tra di loro in alcuni importanti aspetti:

Architettura

- Una rete RBF (nella sua forma più elementare) ha un singolo hidden layer;
- Un MLP può avere uno o più hidden layers;

Modello dei neuroni

- I nodi computazionali dell'hidden layer di una rete RBF sono differenti e servono ad uno scopo diverso rispetto a quelli dell'ouput layer;
- Tipicamente i nodi computazionali di un MLP, che siano locati in un hidden o in un output layer, condividono un modello di neurone comune;
- L'hidden layer di una rete RBF è non loneare, mentre l'output layer è tipicamente lineare;
- Gli hidden e gli output layers di un MLP usato come classificatore sono tipicamente tutti non lineari; quando il MLP è usato per risolvere problemi di regressione non lineari, la scelta tipica è quella di un layer lineare per l'output;

Funzione di attivazione

- L'argomento della funzione di attivazione di ogni hidden unit in una rete RBF è la *norma Euclidea (distanza)* tra il vettore di input e il centro di tale unità;
- L'argomento della funzione di attivazione di ogni hidden unit in un MLP è il *prodotto scalare* tra il vettore di input e il vettore dei pesi sinaptici di tale unità;

Approssimazione

- Le reti RBF usando non-linearità localizzate esponenzialmente decrescenti (ad esempio, funzioni Gaussiane) creano delle approssimazioni *locali* per un mapping input-output non lineare, con il risultato che queste reti sono capaci di apprendere velocemente ed avere una ridotta sensitività all'ordine di presentazione dei training data. In molti casi, però, abbiamo che per dover rappresentare un mapping ad un determinato grado di "levigatezza"m il numero di funzioni radial basis necessarie per includere l'input space adeguatamente potrebbe essere molto alto;
- I MLPs costruiscono delle approssimazioni *globali* di mapping non lineari input/output. Di conseguenza, sono capaci di fare la generalizzazione in regioni dell'input space dove abbiamo pochi training data;

3 Classificazione

La classificazione è il processo di assegnare degli oggetti a delle classi. Una **classe** contiene oggetti simili, mentre oggetti da classi differenti sono dissimili. Un **oggetto** è descritto come un set di attributi o caratteristiche chiamate *features*. Le *features* possono essere qualitative (categoriche) o quantitative (numeriche). In quest'ultimo caso, i valori delle features per un dato oggetto sono disposto come un vettore di dimensione n.

$$\mathbf{x} = [x_1, ..., x_n]^T \in \mathbb{R}^n$$

Lo spazio reale $\mathbb{R}^n$ è chiamato *feature space*, ogni asse corrisponde ad una feature.

3.1 Classificatore

Un **classificatore** è una qualsiasi funzione:

$$D : \mathbb{R}^n \to \Omega$$

con $\mathbb{R}^n$ il feature space e Ω il set delle class labels.

Un classifier viene creato da un labeled data set, e **assegna le class labels agli oggetti**, cioè **predice** a quale classe deve appartenere un nuovo oggetto. Un classificatore può essere:

- **Discreto**, se ha come output una class label;
- **Continuo**, se ha come output un valore reale (ad esempio una stima della probabilità di appartenenza di un oggetto ad una determinata classe), al quale possono essere applicate soglie differenti per prevedere l'appartenenza ad una classe.

Il confronto tra classifier solitamente è basato sull'**accuratezza della classificazione**:

$$accuracy = \frac{correctly\ classified\ samples}{testing\ samples}$$

Alternativamente, sull'**error rate**: $error\ rate = 1 - accuracy$

L'uso dell'accuratezza presuppone che le propbabilità a priori di appartenenza alla classe siano **costanti** e **relativamente bilanciate**. Presuppone inoltre **gli stessi costi dell'errore**. Analizziamo adesso sia la distribuzione delle classi che i costi dell'errore.

Nei problemi reali, spesso vi è un'alta concentrazione di casi normali o non interessanti, ed un piccolo numero di casi inusuali o interessanti (ad esempio individuzione delle frodi, test medici diagnostici per malattie rare ecc...): la **distribuzione delle classi** è molto **asimmetrica**.

Con una distribuzione assimetrica, l'accuratezza non funziona. Ad esempio, si consideri un dominio dove le classi si presentano con una proporzione 999:1. Una semplice regola "Classifica sempre come la classe più probabile" offre un'**accuratezza del 99%**.

Consideriamo adesso un **problema a due classi** (classificazione binaria), ad esempio un test diagnostico che cerca di determinare se una persona ha una particolare malattia:

p = classe dei positivi (e.g., pazienti con il cancro)

n = classe dei negativi (e.g., pazienti senza cancro)

Sia

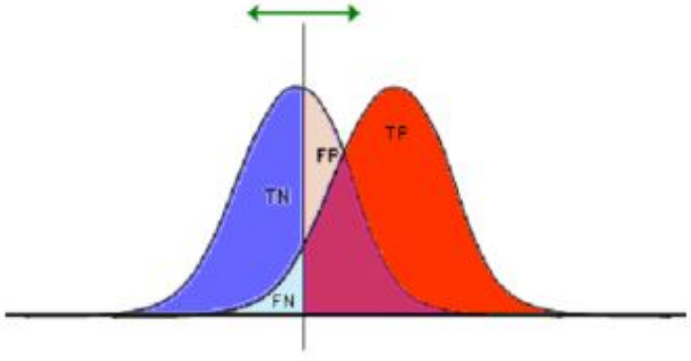
Y = classe predetta positiva

N = classe predetta negativa

Possiamo avere 4 risultati possibili :

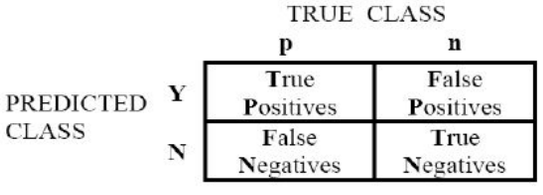
- **True Positive (TP)**: oggetto positivo classificato come positivo (*hit*);
- **True Negative (TN)**: oggetto negativo classificato come negativo (*correct rejection*);
- **False Positive (FP)**: oggetto negativo classificato come positivo (*false alarm*);
- **False Negative (FN)**: oggetto positivo classificato come negativo (*miss*);

I casi FP e FN sono causati dalla sovrapposizione delle distribuzioni delle classi positivo e negativo:



3.2 Matrice di confusione

Definiamo un esperimento con **P** casi positivi e **N** casi negativi. I quattro risultati possono essere rappresentati in una **matrice di confusione** 2x2.



$$accuracy = \frac{TP+TN}{P+N}$$

$$TP\,rate = \frac{TP}{TP+FN} = \frac{TP}{P}$$

$$FP\,rate = \frac{FP}{TN+FP} = \frac{FP}{N}$$

True positive rate (o *hit rate* o *recall*) di un classificatore:

$$TPR = \frac{positives\ correctly\ classified}{total\ positives}$$

TPR coincide con la **sensitività**.
False positive rate (o *false alarm rate*) di un classificatore:

$$FPR = \frac{negatives\ incorrectly\ classified}{total\ negatives}$$

FPR coincide con il complemento a 1 della **specificità**.
Funzione del costo dell'errore: **c(classification, class)**

- $c(Y, n)$ = costo di un errore falso positivo;
- $c(N, p)$ = costo di un errore falso negativo;

La valutazione tramite l'accuratezza della classificazione assume **costi di errore** uguali:

$$c(Y, n) = c(N, p)$$

Nel mondo reale, ciò raramente è vero, dato che le classificazioni conducono ad azioni, le quali hanno delle conseguenze. Le azioni possono essere diverse come rifiutare un accredito o informare un paziente di una diagnosi di cancro. Eseguire un'azione sbagliata potrebbe essere molto costoso. Il costo degli errori difficilmente è equivalente, ad esempio non riconoscere un caso di cancro è molto peggiore di classificare un paziente sano come malato. In tali casi, la massimizzazione dell'accuratezza dovrebbe essere rimpiazzata con la minimizzazione dei costi. I problemi di costi dell'errore non uguali e distribuzioni sbilanciate delle classi sono correlati.

3.3 Grafici ROC

Possono essere fatti dei confronti tra classificatori più generale tramite l'"analisi **Receiver Operating Characteristics (ROC)**, una metodologia classica derivante dalla teoria di rilevamento dei segnali. I grafici ROC sono una tecnica utile per visualizzare e selezionare i calssificatori basandosi sulle loro performance. Sono specialmente utili in presenza di distribuzioni asimmetriche delle classe e costi di classificazione non bilanciati.

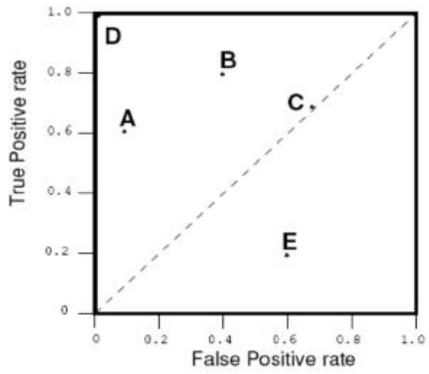
3.4 Spazio ROC

I grafici ROC sono grafici a due dimensioni:

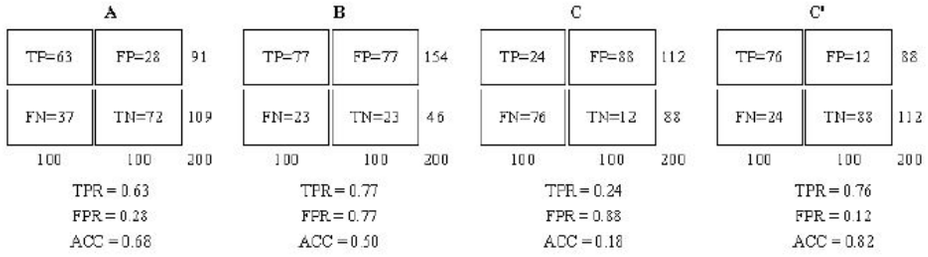
TPR viene posto sull'asse **Y**

FPR viene posto sull'asse **X**

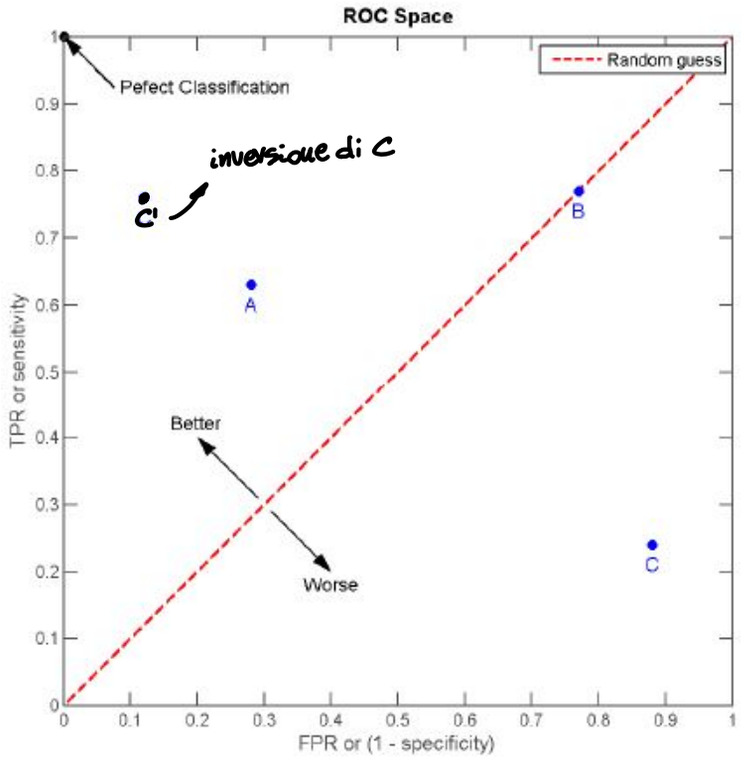
Un grafico ROC mostra il compromesso tra i benefici (true positives) e i costi (false positives).



Un classificatore **binario** (cioè un'istanza della matrice di confusione) produce una coppia (FP rate, TP rate) che corrisponde ad un **punto singolo** nello spazio ROC. La figura illustra un grafico ROC che mostra 5 diversi classificatori (segnati da A a E). Consideriamo 4 predizioni ottenuti da 100 istanze positive e 100 istanze negative:



Plottiamo quindi i quattro risultati sopra mostrati nello spazio ROC:



A mostra un maggior potere predittivo tra **A**, **B**, e **C**.

L'accuratezza di **B** è del 50%, di conseguenza il risultato di B giace sulla random guess line (la linea diagonale). **C** mostra il peggior potere predittivo **A**, **B**, e **C**, ma quando **C** è specchiato rispetto al punto centrale (0.5, 0.5), il metodo risultante **C'** è anche migliore di **A** (il metodo specchiato semplicemente inverte le previsioni di qualsiasi metodo prodotto dalla matrice di confusione **C**).

Anche se il metodo **C** originale ha un potere predittivo negativo, invertendo semplicemente le sue decisoni otteniamo un nuovo metodo **C'** il quale ha un potere predittivo positivo. Quando il metodo **C** predice **p** o **n**, il metodo **C'** prevederebbe**n** o **p**, rispettivamente. In questo modo, **C'** risulta essere il migliore.

Più un risultato della confusion matrix è vicino all'angolo in alto a sinistra migliore è la predizione, ma la distanza dalla random guess line in entrambe le direzioni è il miglior indicatore di quanto potere predittivo ha un metodo. Se il risultato è sotto la linea (ovvero il metodo è peggiore di un tentativo casuale), tutte le previsioni di un metodo devono essere invertite in modo da utilizzare il suo potere, quindi muovendo il risultato sopra la random guess line. Alcuni punti dello spazio ROC sono importanti:

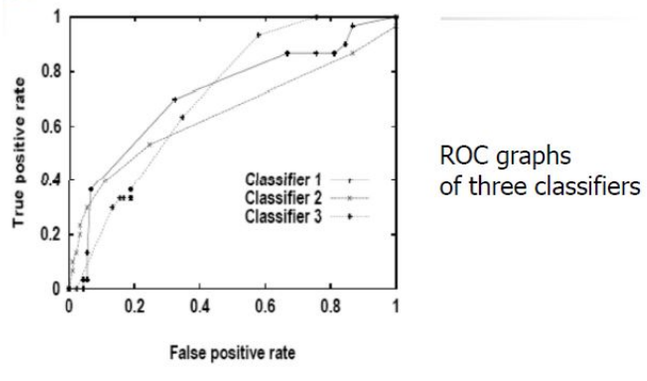
- Il punto in basso a sinistra (0,0) rappresenta la strategia "never alarming", ovvero non rilasciare mai una classificazione positiva: non ci sono errori false positive ma anche nessun true positive;
- Il punto in alto a destra (1,1) rappresenta la strategia oppsta di "never alarming", ovvero rilasciare sempre classificazioni positive;
- il punto (0,1) rappresenta la classificazione perfetta;
- La linea diagonale y=x rappresenta la strategia di tentare casualmente la classe (ogni classificatore che appare nella parte inferiore destra performa peggio che il tentativo casuale).

Ufficiosamente, un punto nello spazio ROC è migliore di un altro se esso è locato più a nord-ovest (TPR maggiore, minor FPR o entrambi). Un grafico ROC permette un confronto grafico informale di un set di classificatori.

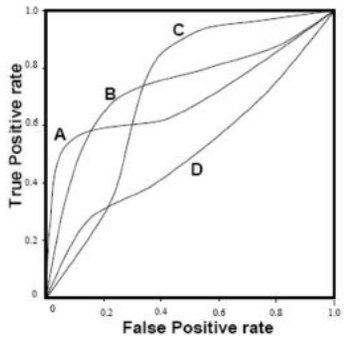
3.4.1 Curve nello spazio ROC

Un classificatore **continuo** è rappresentato da un set di coppie (FPR, TPR), ognuna di esse pertinente ad una specifica scelta della soglia, la quale discrimina gli output negativi da quelli positivi: se l'output del classificatore è al di sopra della soglia, il classificatore produce Y, altrimenti N (ogni soglia definisce un classificatore).

Queste coppie danno origine alla così chiamata **curva ROC**.



La curva ROC mostra il comportamento del classificatore **senza alcun riferimento alla distribuione della classe o il costo dell'errore di classificazione**, separando quindi le performance di classificazione da questi fattori. Sfortunatamente, mentre un grafico ROC è una tecnica grafica utile, può aiutare a scegliere il miglior classificatore globale solo se vi è un classificatore che distacca gli altri su tutto lo spazio ROC. Ad esempio, la curva A è migliore della curva D perchè è superiore ad essa in tutti i punti:



3.5 Area al di sotto della curva (AUC)

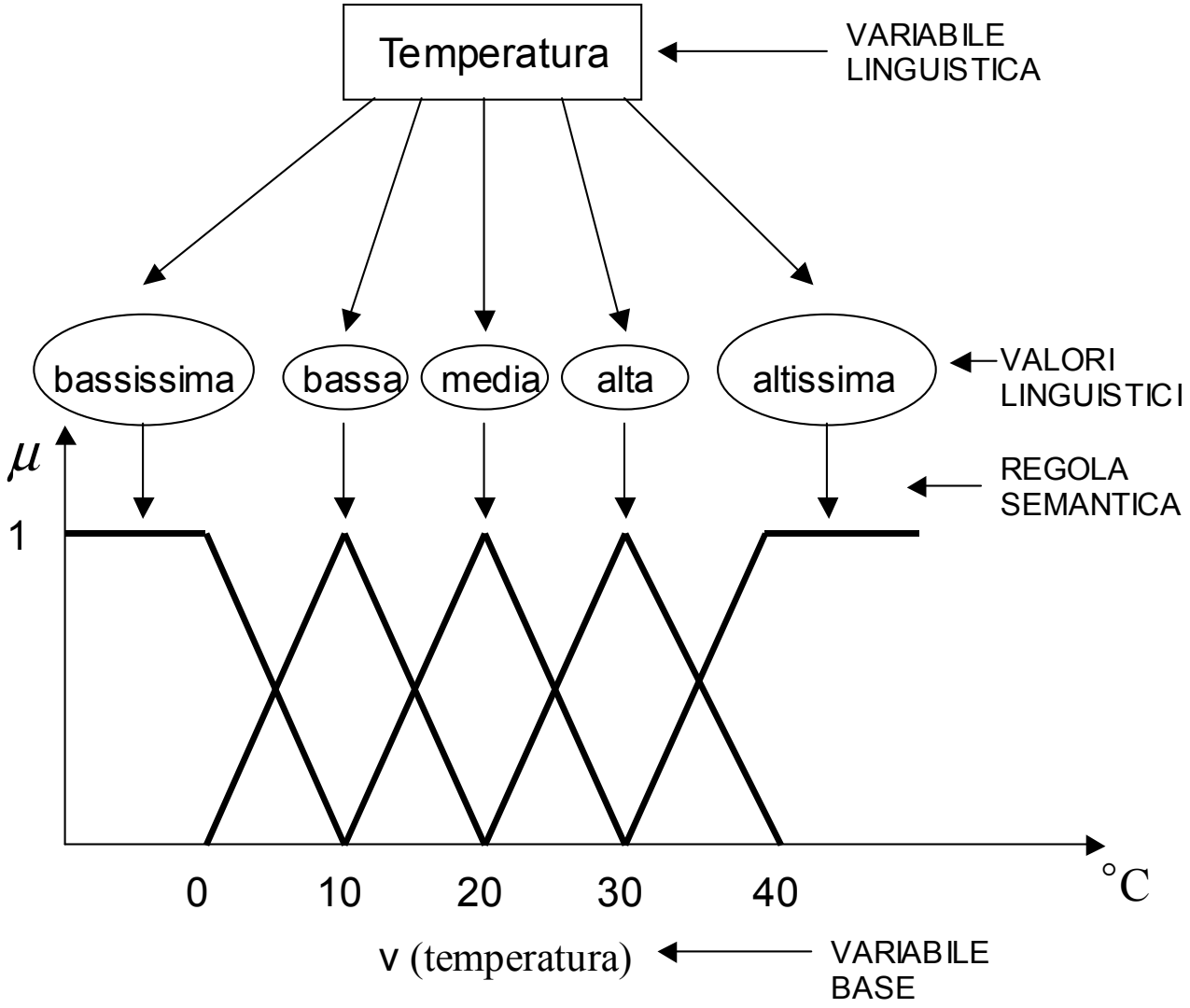
Se nessun classificatore è chiaramente migliore degli altri su tutto lo spazio ROC, un metodo spesso usato consiste nello scegliere il classificatore con l'**are al di sotto della curva (AUC)** con il valore più alto.

Assumento che non si sia particolarmente interesati in uni specifico compromesso tra TPR e FPR (ovvero un particolare punto sulla curva ROC), l'AUC è utile in quanto aggrega le performance sull'intero range di compromessi.

L'interpretazione dell'AUC è facile: più alto è il valore, meglio è. Le performance perfette corrispondono a 1, mentre performance casuali corrispondono ad un valore di 0.5.

VARIABILI LINGUISTICHE

Una variabile linguistica è una variabile i cui valori sono termini linguistici.



Una variabile linguistica è definita in termini di una *variabile base* (variabile in senso classico), i cui valori sono numeri reali all'interno di un dato intervallo.

I *termini linguistici*, che rappresentano valori approssimati della variabile base, sono interpretati come *numeri fuzzy*.

Una variabile linguistica è caratterizzata da una quintupla

$$(v, T, X, g, m)$$

dove v = nome della variabile

T = insieme dei termini linguistici di v;

X = universo di definizione della variabile base;

g = regola sintattica (una grammatica) per generare i termini linguistici (ad esempio, "molto alta", "non molto bassa");

m = regola semantica che assegna a ciascun termine linguistico $t \in T$ il suo significato, m(t), che è un insieme fuzzy su X (universo di definizione della variabile base), cioè $m: T \rightarrow F(X)$.

Una variabile linguistica è definita attraverso i suoi *termini primari* (rappresentati nella figura precedente). I termini primari sono etichette di insiemi fuzzy: 'bassissima', 'bassa', 'media', 'alta' e 'altissima'.

MODIFICATORI LINGUISTICI

I termini atomici (quali 'medio', 'alto', 'buono', 'bello') del linguaggio naturale possono essere modificati con aggettivi o avverbi come 'molto', 'quasi', 'più o meno', ….

Interpretando i termini atomici come insiemi fuzzy, l'effetto di un modificatore linguistico è quello di modificare la funzione di appartenenza di un termine atomico.

Sia α un termine atomico

$$\alpha = \int_Y \mu_\alpha(y) / y .$$

Si ha, ad esempio:

$$\text{"molto"}\alpha = \alpha^2 = \int_Y [\mu_\alpha(y)]^2 / y$$

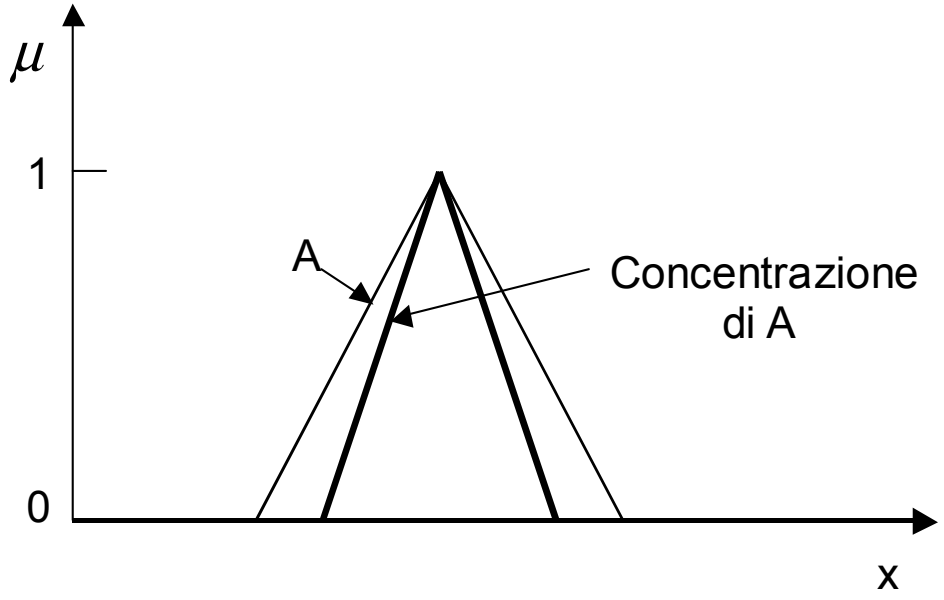
$$\text{"leggermente"}\alpha = \sqrt{\alpha} = \int_Y [\mu_\alpha(y)]^{0.5} / y$$

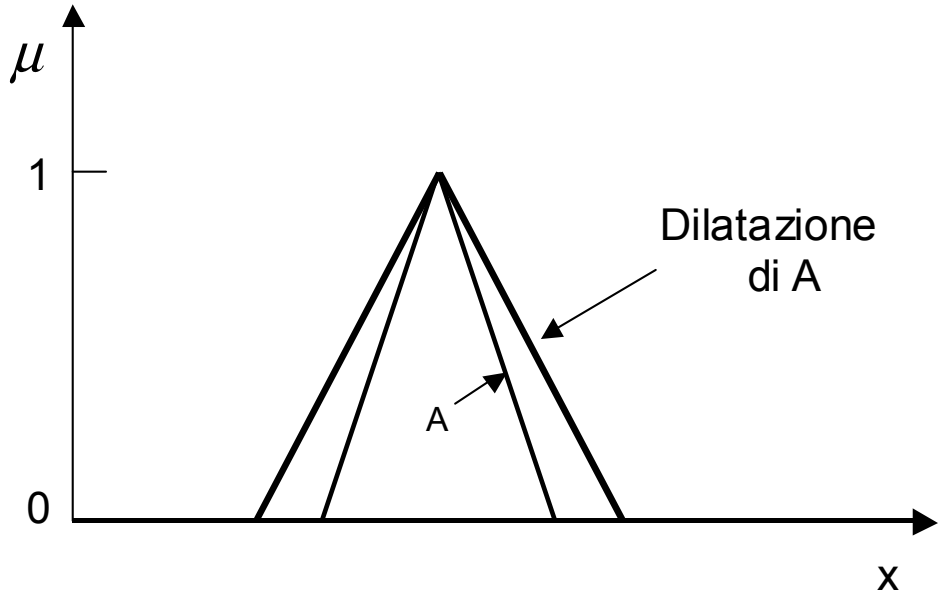
…

Le espressioni come la prima sono dette *concentrazioni*, perché riducono il grado di appartenenza di tutti gli elementi che sono "parzialmente" nell'insieme (*riduzione dell'incertezza*).

Più basso è il valore di appartenenza di un elemento ad un insieme fuzzy, più è ridotta la sua appartenenza mediante una concentrazione.

Le espressioni come la seconda sono dette *dilatazioni*, perché incrementano l'appartenenza di elementi che sono "parzialmente" nell'insieme (*aumento dell'incertezza*).


μ
1
0
A
Concentrazione
di A
x

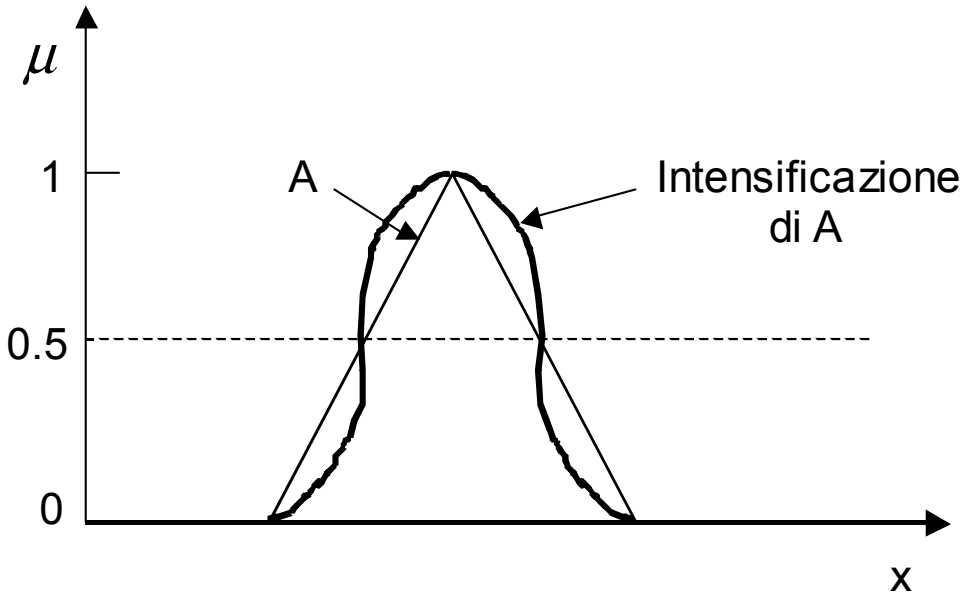

μ
1
0
Dilatazione
di A
A
x

INTENSIFICAZIONE

Un'altra operazione sugli insiemi fuzzy linguistici è l'*intensificazione*. Questa operazione incrementa il grado di appartenenza degli elementi il cui valore di appartenenza originale è > 0.5 e diminuisce il grado di appartenenza degli elementi il cui valore di appartenenza originale è < 0.5. L'intensificazione può essere espressa in vari modi, uno dei quali, proposto da Zadeh, è

$$\text{"intensifica"}\alpha = \begin{cases} 2\mu_{\alpha}^{2}(y) & \text{per } 0 \le \mu_{\alpha}(y) \le 0.5 \\ 1-2[1-\mu_{\alpha}(y)]^{2} & \text{per } 0.5 \le \mu_{\alpha}(y) \le 1 \end{cases}.$$

L'intensificazione aumenta il contrasto tra gli elementi dell'insieme con $\mu > 0.5$ e quelli con $\mu < 0.5$.



TERMINI COMPOSTI

Il linguaggio naturale comprende, oltre ai *termini atomici* (‘alto’, ‘bello’, …), anche i *termini composti*, ottenuti combinando termini atomici, connettivi logici e modificatori linguistici.

LOGICA FUZZY

La logica fuzzy costituisce il fondamento teorico del ragionamento approssimato.

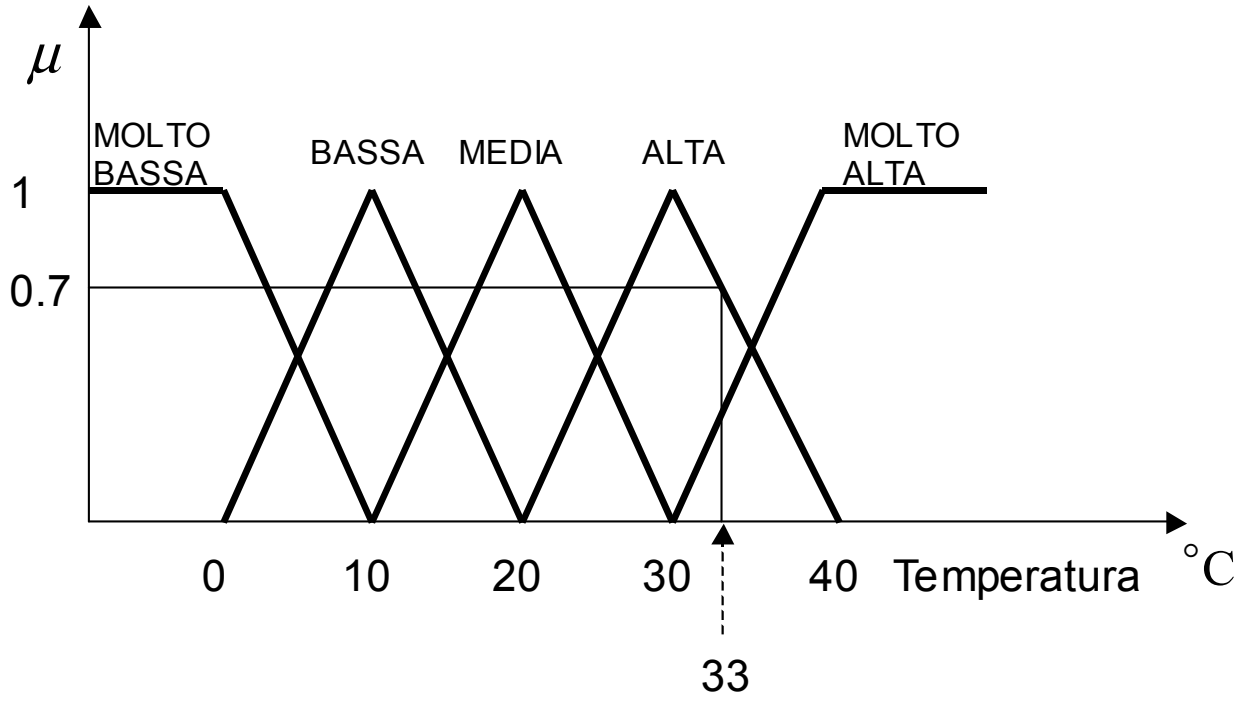
PROPOSIZIONI FUZZY

$$P: X \text{ is } A$$

dove X è una variabile che assume valori su un universo U ed A è un insieme fuzzy su U. Ad esempio, A può rappresentare un predicato fuzzy, come "alto", "medio", …

Il grado di verità di P è

$$T(P) = A(X)$$



Il grado di verità, T(P), dipende dal valore effettivo della temperatura e dalla definizione del predicato A ("temperatura alta").

Se X = 33, allora A(33) = 0.7 e T(P) = 0.7.

PROPOSIZIONI FUZZY CONDIZIONALI

Le proposizioni fuzzy condizionali sono relazioni tra variabili linguistiche:

$$p : \text{if } X \text{ is } A \text{ then } Y \text{ is } B$$

dove X e Y sono variabili definite, rispettivamente, su U e V, A e B sono insiemi fuzzy su U e V, rispettivamente.

Una proposizione condizionale è rappresentata da una relazione R, che è un insieme fuzzy su $U \times V$ cosí definito:

$$R(x, y) = I[A(x), B(y)]$$

dove I denota implicazione fuzzy e A(X) e B(Y) sono i valori di verità delle proposizioni $X \text{ is } A$ e $Y \text{ is } B$, rispettivamente.

Quindi il grado di appartenenza R(x,y) della coppia (x,y) alla relazione R rappresenta il valore di verità della proposizione fuzzy condizionale

$$\text{if } X \text{ is } A \text{ then } Y \text{ is } B.$$

Siano P e Q due proposizioni fuzzy. Analogamente al caso crisp abbiamo:

- negazione: $T(\overline{P}) = 1 - T(P)$
- disgiunzione: $P \vee Q : x \text{ is } A \text{ or } B$ $\quad T(P \vee Q) = \max(T(P), T(Q))$
- congiunzione: $P \wedge Q : x \text{ is } A \text{ and } B$ $\quad T(P \wedge Q) = \min(T(P), T(Q))$
- implicazione: $(P \rightarrow Q)$ Esistono più modi per definire l'implicazione fuzzy

IMPLICAZIONE FUZZY

L'implicazione fuzzy è una funzione

$$I:[0,1]\times[0,1]\rightarrow[0,1]$$

che, date due proposizioni fuzzy P e Q, definisce il valore di verità della proposizione condizionale IF P THEN Q.

Tale funzione dovrebbe essere un'estensione dell'implicazione classica, cioè $I(0,0)=I(0,1)=I(1,1)=1$ e $I(1,0)=0$.

Nella logica classica l'implicazione può essere definita in vari modi le cui estensioni alla logica fuzzy danno luogo a diversi operatori di implicazione fuzzy.

Ad esempio, l'implicazione $(P\rightarrow Q)$ può essere rappresentata sotto forma di regola

$$\text{IF } X \text{ is } A \text{ THEN } Y \text{ is } B$$

ed è equivalente alla relazione fuzzy

$$R=(A\times B)\cup(\overline{A}\times Y)$$

per cui, risolvendo il prodotto con la funzione *min* e l'unione con la funzione *max*, una possibile espressione per l'operatore di implicazione fuzzy è la seguente:

$$\mu_R(x,y)=\max\{\min[\mu_A(x),\mu_B(y)],1-\mu_A(x)\} \qquad \text{(Zadeh)}$$

ALTRE ESPRESSIONI PER L'OPERATORE DI IMPLICAZIONE FUZZY

$\mu_R(x,y) = \max\{1-\mu_A(x), \mu_B(y)\}$ Kleene-Dienes

$\mu_R(x,y) = \min\{1, [1-\mu_A(x) + \mu_B(y)]\}$ Lukasiewicz

$\mu_R(x,y) = \min\{\mu_A(x), \mu_B(y)\}$ Mamdani

…..

Osserviamo che l'espressione di Mamdani, benché spesso usata, non è un'estensione dell'implicazione classica, in quanto I(0,1)=min(0,1)=0.

La scelta di un operatore di implicazione dipende dal contesto.

Esempio

Siano $A = \frac{0.1}{x_1} + \frac{0.8}{x_2} + \frac{1}{x_3}$ e $B = \frac{0.5}{y_1} + \frac{1}{y_2}$.

Consideriamo l'implicazione di Lukasiewicz:

$$I(x,y) = \min(1, 1-x+y)$$

Allora

$$A \rightarrow B = \frac{1}{x_1, y_1} + \frac{1}{x_1, y_2} + \frac{0.7}{x_2, y_1} + \frac{1}{x_2, y_2} + \frac{0.5}{x_3, y_1} + \frac{1}{x_3, y_2}$$

Questo significa, ad esempio, che

$T(A \rightarrow B) = 1$ quando $x = x_1$ e $y = y_1$

$T(A \rightarrow B) = 0.7$ quando $x = x_2$ e $y = y_1$

…

RAGIONAMENTO APPROSSIMATO

Sia data la regola

$$R: \text{IF } X \text{ is } A \text{ THEN } Y \text{ is } B$$

Dato un nuovo antecedente A' si può derivare un nuovo conseguente B' mediante la composizione fuzzy

$$B' = A' \circ R$$

Si può usare la composizione *max-min*, la composizione *max-product*, etc.

In generale,

$$B'(y) = \sup_{x \in U} T[A'(x), I(A(x), B(y))]$$

dove T è una t-norma e I un operatore di implicazione fuzzy.

Questa regola è detta *modus ponens generalizzato* o *regola composizionale di inferenza*.

Ogni scelta di T e di I genera una regola diversa.

In modo analogo si può ottenere anche il *modus tollens generalizzato*.

Esempio

Siano date le variabili X e Y definite su $U=\{x_1, x_2, x_3\}$ e $V=\{y_1, y_2\}$, rispettivamente. Sia data la proposizione $\text{IF X is A THEN Y is B}$.

Siano $A=\frac{0.5}{x_1}+\frac{1}{x_2}+\frac{0.6}{x_3}$ $\quad B=\frac{1}{y_1}+\frac{0.4}{y_2}$.

Sia dato un fatto espresso dalla proposizione X is A' con

$$A'=\frac{0.6}{x_1}+\frac{0.9}{x_2}+\frac{0.7}{x_3}.$$

Usando l'implicazione di Lukasiewicz I = min(1, 1-x+y), abbiamo

$$R=\frac{1}{x_1,y_1}+\frac{0.9}{x_1,y_2}+\frac{1}{x_2,y_1}+\frac{0.4}{x_2,y_2}+\frac{1}{x_3,y_1}+\frac{0.8}{x_3,y_2}.$$

Applicando il modus ponens generalizzato con T=min otteniamo

$$B'(y_1)=\sup_{x\in U}\min[A'(x),R(x,y_1)]$$

$$=\max[\min(0.6,1),\min(0.9,1),\min(0.7,1)]$$

$$=0.9$$

$$B'(y_2)=\sup_{x\in U}\min[A'(x),R(x,y_2)]$$

$$=\max[\min(0.6,0.9),\min(0.9,0.4),\min(0.7,0.8)]$$

$$=0.7$$

Si può concludere Y is B' dove $B'=\frac{0.9}{y_1}+\frac{0.7}{y_2}$.

SISTEMI FUZZY CON PIÙ REGOLE

Regola 1: IF X is A_1 THEN Y is B_1

Regola 2: IF X is A_2 THEN Y is B_2

...

Regola n: IF X is A_n THEN Y is B_n

Fatto: X is A'

Conclusione: Y is B'

Ogni regola produce una conclusione.

Esistono due possibili strategie per combinare le n conclusioni:

FITA (First Infer Then Aggregate)

FATI (First Aggregate Then Infer)

STRATEGIA FITA

Prima si applica la regola composizionale di inferenza alle singole regole, poi si combinano le conclusioni inferite dalle regole attraverso un operatore di aggregazione.

Sia data una base di *n* regole $I(A_i(x), B_i(y)),\ i=1,...,n,$ ed un fatto $A'(x)$. La conclusione ottenuta con la strategia FITA è

$$B^{(1)} = h(B'_1, ..., B'_n)$$

dove $B'_i(y) = A'(x) \circ I(A_i(x), B_i(y)) = \sup_{x \in U} T[A'(x), I(A_i(x), B_i(y))]$

e *h* è un *operatore di aggregazione* (di solito, il minimo o il massimo).

Quindi, l'approccio FITA include nell'ordine:

- implicazione
- composizione
- aggregazione

STRATEGIA FATI

Prima si aggregano tutte le regole generando una relazione fuzzy globale R che è l'aggregazione di tutte le relazioni di implicazione fuzzy corrispondenti alle singole regole, poi si applica la regola composizionale di inferenza al fatto e alla relazione R.

Sia data una base di *n* regole $I(A_i(x), B_i(y)),\ i=1,...,n,$ ed un fatto $A'(x)$. La conclusione ottenuta con la strategia FATI è

$$B^{(2)} = A'(x) \circ R(x,y) = \sup_{x \in U} T(A'(x), R(x,y))$$

$$\text{dove } R(x,y) = h(I(A_1(x), B_1(y)),..., I(A_n(x), B_n(y)))$$

e *h* è un operatore di aggregazione.

Quindi, l'approccio FATI include nell'ordine:

- implicazione
- aggregazione
- composizione

Risulta $B^{(2)} \subseteq B^{(1)}$.

REGOLE SISO
(Single-Input Single-Output)

Sono regole con un solo antecedente:

$$\text{IF } X \text{ is } A \text{ THEN } Y \text{ is } B$$

REGOLE MISO
(Multi-Input Single-Output)

Sono regole con più antecedenti:

$$\text{IF } X_1 \text{ is } A_1 \text{ AND } X_2 \text{ is } A_2 \text{ ... AND } X_m \text{ is } A_m \text{ THEN } Y \text{ is } B_1$$

Il connettivo AND è interpretato come una congiunzione fuzzy, cioè una relazione fuzzy $R(X_1,...,X_m)$ definita sul prodotto cartesiano $X_1 \times ... \times X_m$ come segue:

$$R(X_1,...,X_m) = i(A_1(x_1),...,A_m(X_m))$$

dove *i* è una t-norma.

Quindi, dato un fatto

$$X_1 \text{ is } A'_1 \text{ AND ... AND } X_m \text{ is } A'_m$$

si ha:

$$B'(y) = \sup_{x_1,...,x_m} T\left[i(A'_1(X_1),...,A'_m(X_m)), I(i(A_1(X_1),...,A_m(X_m)), B(y)) \right]$$

dove T ed i sono t-norme (in generale, può essere $T \neq i$).

Di solito, essendo richiesta l'idempotenza, i = minimo.

TECNICHE GRAFICHE DI INFERENZA

Un sistema fuzzy con *m* ingressi ed una singola uscita è descritto da una collezione di *n* regole IF-THEN:

Regola 1: IF X_1 is A_{11} AND ... AND X_m is A_{1m} THEN Y is B_1

Regola 2: IF X_1 is A_{21} AND ... AND X_m is A_{2m} THEN Y is B_2

.....

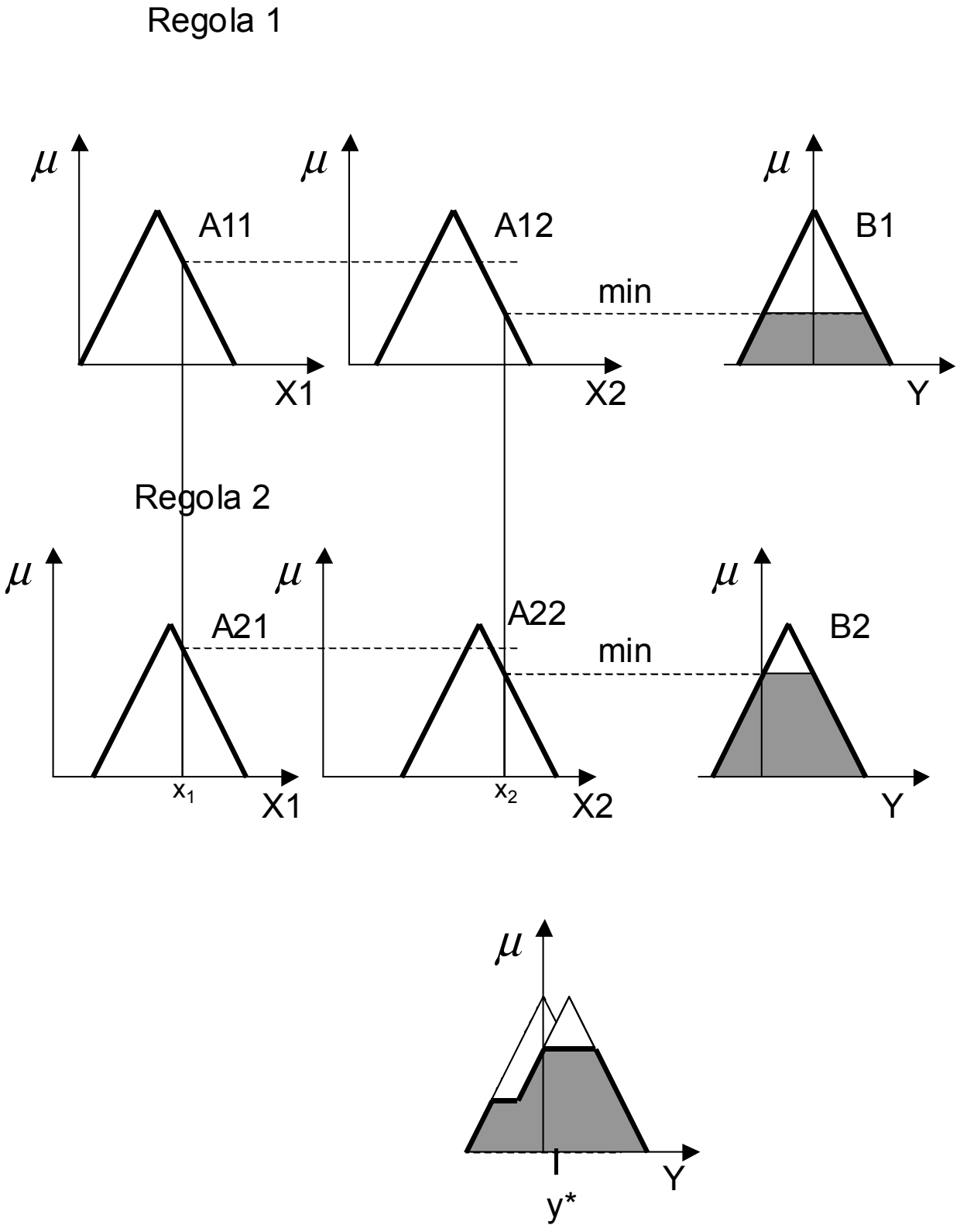
Regola n: IF X_1 is A_{n1} AND ... AND X_m is A_{nm} THEN Y is B_n

Consideriamo un sistema con due ingressi ed una uscita; il sistema è descritto dalle seguenti due regole fuzzy:

Regola 1: IF X_1 is A_{11} AND X_2 is A_{12} THEN Y is B_1

Regola 2: IF X_1 is A_{21} AND X_2 is A_{22} THEN Y is B_2

Supponiamo che i due ingressi crisp al sistema siano x_1 e x_2. Nella pagina successiva mostriamo graficamente il processo di inferenza fuzzy. Occorre innanzi tutto '*fuzzificare*' gli input: tipicamente, si considerano x_1 ed x_2 come fuzzy singleton. Usiamo poi il minimo per risolvere l'operatore AND nell'antecedente di ogni regola, e realizziamo anche l'operatore di implicazione con il minimo (questo equivale a troncare la funzione di appartenenza dell'insieme conseguente con il minimo dei valori di appartenenza degli input agli insiemi fuzzy antecedenti). Infine usiamo il massimo come operatore di aggregazione degli insiemi fuzzy prodotti da ciascuna regola.


Regola 1
μ
A11
X1
μ
A12
X2
min
μ
B1
Y
Regola 2
μ
A21
x1
X1
μ
A22
x2
X2
min
μ
B2
Y
μ
Y
y*

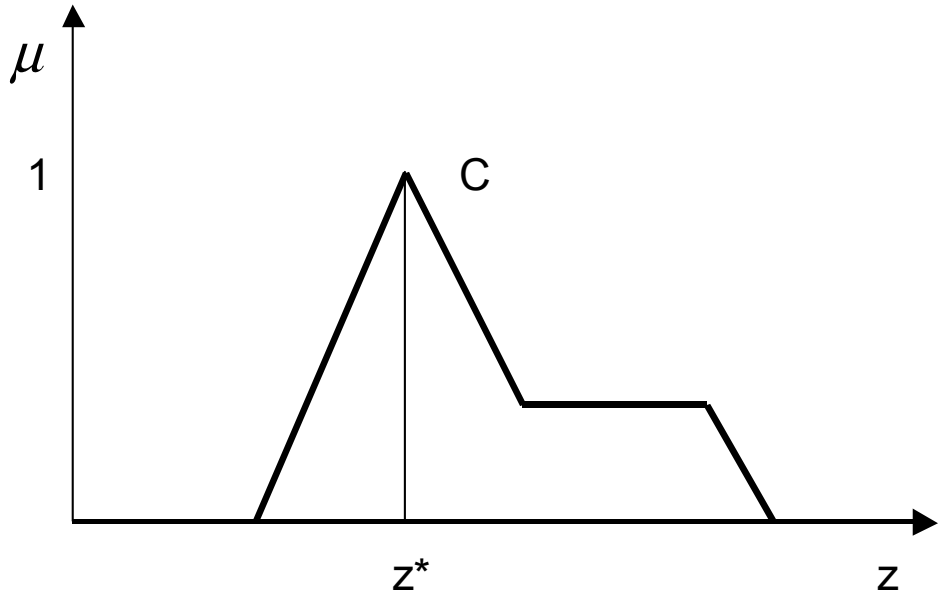
DEFUZZIFICAZIONE

Sia C l'insieme fuzzy di uscita. In generale, C risulterà dall'unione di varie funzioni triangolari, trapezoidali, etc. Magari non tutte queste funzioni saranno normali. Occorre '*defuzzificare'* C per produrre un numero crisp.

METODI DI DEFUZZIFICAZIONE

- *Metodo della massima appartenenza* (o *metodo dell'altezza*): è usato solo per funzioni di uscita con un picco:

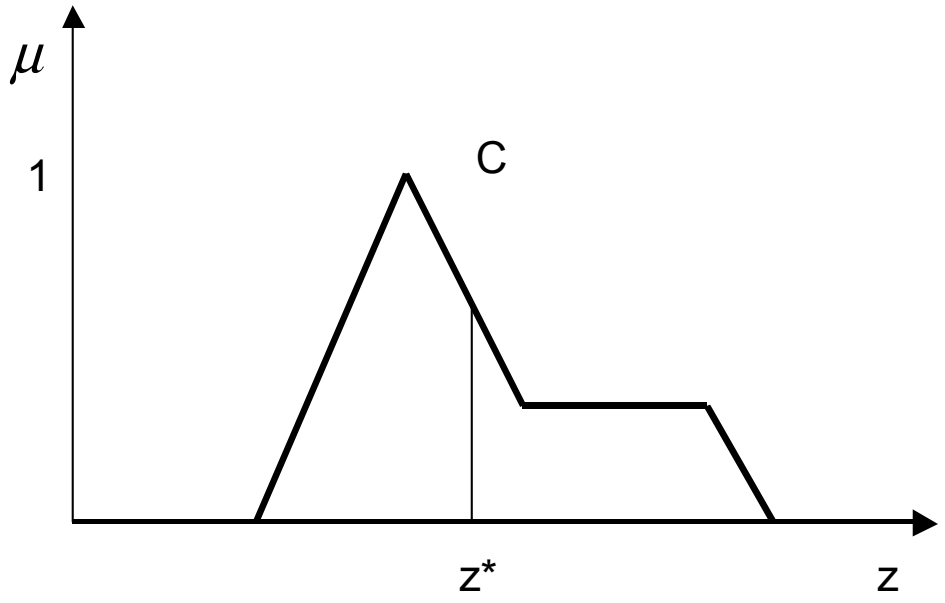
$$\mu_C(z^*) \geq \mu_C(z) \quad \forall z \in Z$$



- *Metodo del centroide* (o *centro di gravità*): è il più usato:

$$z^* = \frac{\int \mu_C(z) \cdot z \, dz}{\int \mu_C(z) \, dz}$$

dove $\int$ denota integrazione algebrica.

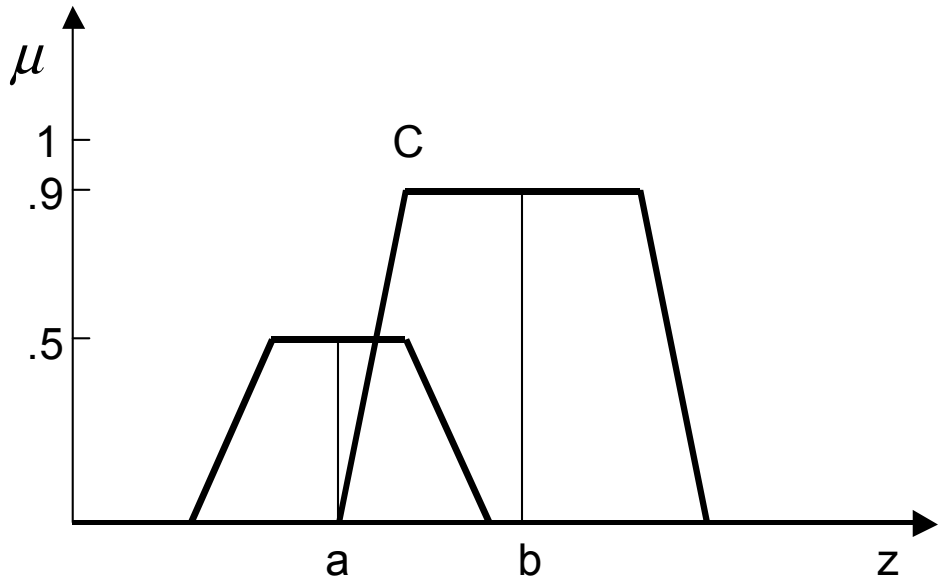


- *Metodo della media pesata*: è usato solo per funzioni di appartenenza simmetriche.

$$z^* = \frac{\sum \mu_C(\bar{z}) \cdot \bar{z}}{\sum \mu_C(\bar{z})}$$

dove $\sum$ denota somma algebrica. Ogni funzione membro nell'uscita è pesata con il suo valore di appartenenza massimo.

Esempio:



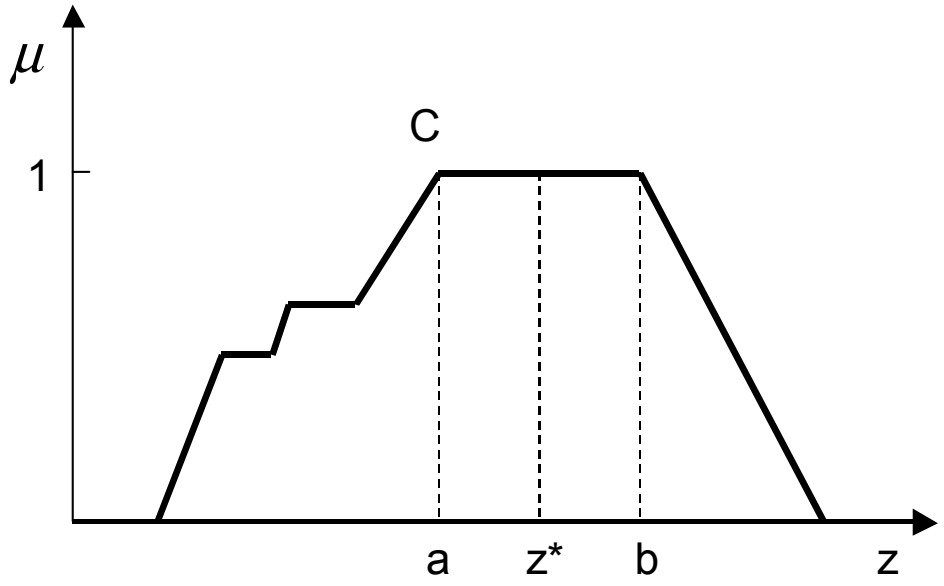
$$z^* = \frac{(.5a) + (.9b)}{.5 + .9}$$

dove a e b sono le medie delle rispettive funzioni.

- *Metodo della media dei massimi*: è simile al primo metodo eccetto il fatto che il valore massimo può essere assunto in più di un punto. Si ha

$$z^* = \frac{a + b}{2}$$

dove *a* e *b* sono definiti come segue:



$$a = \inf \{z | \mu_C(z) = h(C)\}$$
$$b = \sup \{z | \mu_C(z) = h(C)\}$$

con h = altezza di C.

SCELTA DEL METODO DI DEFUZZIFICAZIONE

La scelta del metodo dipende dal contesto e dal problema.

Possono comunque essere tenuti presenti alcuni criteri per misurare la bontà di un metodo, quali

1) *continuità*: un piccolo cambiamento nell'input di un processo fuzzy non dovrebbe produrre un grande cambiamento nell'output.

2) *Non ambiguità*: un metodo di defuzzificazione dovrebbe produrre un solo valore per z^*.

3) *Plausibilità*: per essere plausibile, z^* dovrebbe trovarsi approssimativamente nel mezzo del supporto di C ed avere un alto grado di appartenenza a C.

4) *Semplicità computazionale*: per esempio, il metodo dell'altezza e il metodo della media dei massimi sono più semplici del metodo del centroide.

8 An Introduction to Fuzzy Logic and Fuzzy Systems pt. 2

8.1 Tipi di regole fuzzy

Nella seguente parte vengono descritte alcune fuzzy rules.

8.1.1 Mamdani fuzzy rule

$$\textit{if } X_1 \textit{ is } A_{i1} \textit{ and } \ \ldots \textit{ and } X_m \textit{ is } A_{im} \textit{ then } Y \textit{ is } B_i$$

Il risultato della regola è un fuzzy set. Il principale vantaggio è la sua alta interpretabilità. Gli svantaggi sono una bassa precisione e costi computazionali alti.

8.1.2 Takagi-Sugeno-Kang (TSK) fuzzy rule

$$\textit{if } X_1 \textit{ is } A_{i1} \textit{ and } \ \ldots \textit{ and } X_m \textit{ is } A_{im} \textit{ then}$$

$$Y \textit{ is } a_{i0} + a_{i1}X_1 + \ldots + a_{im}X_m$$

dove $a_{i0}, a_{i1}, \ldots, a_{im}$ sono numeri reali.

Il risultato della regola è una funzione, tipicamente **lineare**, delle variabili di input. Quindi, ogni regola può essere considerata come un *modello locale lineare*.

Talvolta possono essere usate delle funzioni lineari di secondo (o più) grado.

L'output del sistema è la media ponderata di ogni output della regola.

- Vantaggio principale: precisione più alta dei sistemi Mamdani;
- Svantaggio: bassa interpretabilità.

8.1.3 Singleton fuzzy rule

Il risultato della regola è un *valore costante*.

Una fuzzy singleton rule può essere considerata come un caso particolara sia della Mamdani che della TSK fuzzy rule. Infatti, un valore costante è equivalente sia per un singleton fuzzy set che per una funzione lineare nella quale i coefficenti delle variabili di input sono 0.

La rappresentazione del singleton costituisce un compromesso tra l'interpretabilità offerta dai sistemi Mamdani con le loro labels utili e l'accuratezza offerta dai TSK con le loro funzioni lineari.

Inoltre, grazie alla rappresentazione discreta della variabile di utput, la defuzzificazione richiede meno calcoli che negli altri due casi.

8.2 Previsione dei turisti pt.2

TSK Convertiamo il sistema in un sistema TSK rimpiazzando i 3 risultati linguistici - *Low, Medium e High* - con delle funzioni lineari delle variabili di input adeguatamaente calcolate. I tre risultati delle regole quindi diventano:

Rule 1: **if** ... **then** Tourists = $f_1(T,S)=2T+0.8S-40$

Rule 2: **if** ... **then** Tourists = $f_2(T,S)=T+S-23$

Rule 3: **if** ... **then** Tourists = $f_3(T,S)=0.5T+0.3S$

Con questi risultati, lo step di implicazione possiede le seguenti previsioni:

	Rule 1	Rule 2	Rule 3
Truth level μ_{Rule}	0.2	0.67	0.33
Estimated tourists	$f_1(19,60)=46\%$	$f_2(19,60)=56\%$	$f_3(19,60)=27.5\%$

Lo step di aggregazione non modifica questi valori. Quindi, il defuzzificatore produce:

$$output=\frac{\sum_{i=1}^{3} f_i(19,60)\cdot\mu_{Rulei}}{\sum_{i=1}^{3}\mu_{Rulei}}$$
$$=\frac{0.2\times46\%+0.67\times56\%+0.33\times27.5\%}{0.2+0.67+0.33}$$
$$=46.4\%$$

Singleton Il sistema Mamdani è facilmente convertito in un sistema singleton rimpiazzando i valori fuzzy Low, Medium e High con i loro corrispondenti center of areas: 0%, 50% e 100%. Con questi nuovi risultati, lo step di implicazione possiete:

	Rule 1	Rule 2	Rule 3
Truth level μ_{Rule}	0.2	0.67	0.33
Estimated tourists	High=100%	Medium=50%	Low=0%

Il defuzzificatore produce::

$$output = \frac{\sum_{i=1}^{3} f_i(19,60) \cdot \mu_{Rule\,i}}{\sum_{i=1}^{3} \mu_{Rule\,i}}$$
$$= \frac{0.2 \times 100\% + 0.67 \times 50\% + 0.33 \times 0\%}{0.2 + 0.67 + 0.33}$$
$$= 44.4\%$$

8.3 Un nuovo scenario di modellazione

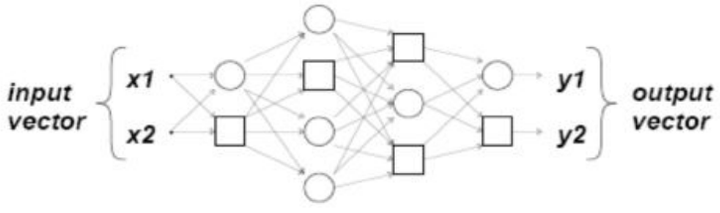
Fino ad ora abbiamo solo considerato delle funzioni di appartenenza che sono state fissate, e in qulche modo scelte arbitrariamente. Inoltra, abbiamo solo applicato le inferenza fuzzy a sistemi di modellazione le cui regole strutturali sono essenzialmente predeterminate dall'interpretazione dell'utente delle caratteristiche delle variabili del modello.

Abbiamo una collezione di dati di input/output che vorremmo usare per la modellazione. Non abbiamo necessariamente un modello strutturale predeterminato basato sulle caratteristiche delle variabili nel nostro sistema.

Invece che scegliere i parametri associati con una data funzione di appartenenza, questi parametri possono essere scelti in modo da personalizzare le funzioni di appartenenza ai dati di input/output.

8.3.1 Reti adattive

Una **rete adattiva** è una rete multi-layer feedforward nella quale ogni nodo segue una particolare funzione (**funzione di nodo**) sul segnale in ingresso.



Un nodo **quadrato** (nodo **adattivo**) ha dei parametri. Un nodo **circolare** (nodo **fisso**) non ha parametri.

I collegamenti indicano solamente la direzione del flusso dei segnali tra i nodi, e non vi è associato nessun peso.

Per ottenere un desiderato mapping input/output, i parametri sono aggiornati per in accordo al training data fornito e ad una procedura di learning gradient-descent.

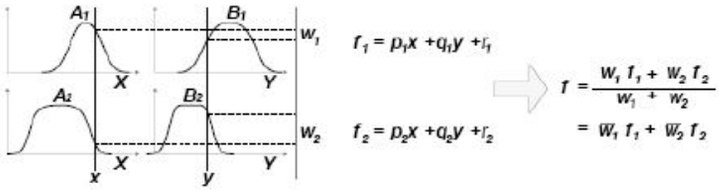
8.3.2 Adaptive-Network-based Fuzzy Inference Systems (ANFIS)

ANFIS Si riferice ad una classe di reti adattive che sono funzionalmente equivalenti ai sistemi di inferenza fuzzy.

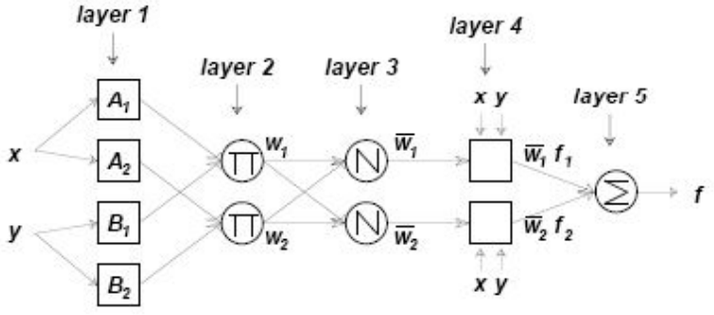
Assumiamo che il sistema di inferenza fuzzy che stiamo considerando abbia due input X e Y e un output f. Supponiamo che la regola base contenga due fuzzy con regola if-then di tipo TSK:

$$\text{Rule 1: } \textit{if } X \textit{ is } A_1 \textit{ and } Y \textit{ is } B_1 \textit{ then } f_1 = p_1 x + q_1 y + r_1$$
$$\text{Rule 2: } \textit{if } X \textit{ is } A_2 \textit{ and } Y \textit{ is } B_2 \textit{ then } f_2 = p_2 x + q_2 y + r_2$$

- Fuzzy reasoning



- ANFIS architecture



- **Layer 1**: Ogni nodo i in questo layer è un nodo quadrato con funzione di nodo

$$O_i^1 = \mu_{A_i}(x)$$

  I parametri della funzione di appartenenza sono chiamati *premise parameters*;

- **Layer 2**: Ogni nodo in questo layer rappresenta la *firing strength* di una regola. Ogni nodo in questo layer è un nodo circolare etichettato

con Π, che moltiplica il segnale in ingresso e inoltra il prodotto (o altre t-norm), ad esempio

$$w_1 = \mu_{A_i}(x) \times \mu_{B_i}(y) \quad i = 1, 2$$

- **Layer 3**: Ogni nodo i in questo layer è un nodo circolare etichettato con N. L'i-esimo nodo calcola il rapporto tra l'i-esima firing strength della regola e la somma di tutte le firing strength delle regole

$$\bar{w_1} = \frac{w_i}{w_1 + w_2} \quad i = 1, 2$$

  Gli output di questo layer sono chiamati *normalized firing strengths*

- **Layer 4**: Ogni nodo i in questo layer è un nodo quadrato con funzione di nodo

$$O_i^4 = \bar{w}_i f_i = \bar{w}_i(p_i x + q_i y + r_i)$$

  dove p_i, q_i, r_i sono i *paramentri di risultato*;

- **Layer 5**: Il singolo nodo nel layer calcola l'output totale come

$$O_1^5 = \textit{overall output} = \sum_i \bar{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i}$$

Quindi abbiamo costruito una rete adattiva che è funzionalmente equivalente ad un sistema di tipo TSK di inferenza fuzzy.

8.4 Algoritmo di learning ibrido

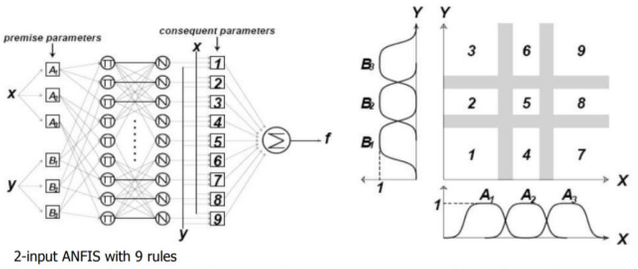
Al posto di una procedura di learning con gradient descent, possiamo usare un algoritmo di learning più veloce, che combina il metodo del gradiente e il least squares estimate (LSE).

Ogni epoch dell'algoritmo di learning ibrido consiste in due step:

- Nello step di **forward**, dati i valori dei parametri premise, forniamo i dati di input e i segnali funzionali in modo che vengano inoltrati fino al quarto layer per calcolare ogni output dei nodi, e i successivi parametri sono identificati dal **least squares estimate**; Dopo aver indentificato questi parametri, i segnali funzionali contuinano ad essere inoltrati fino a che l'errore viene calcolato.

- Nello step di **backward**, i rate dell'errore viene propagato in direzione inversa e i parametri premise vengono aggiornati dal **gradient descent**.

-	forward pass	backward pass
premise parameters	fixed	gradient descent
consequent parameters	least squares estimate	fixed
signals	node outputs	error rates

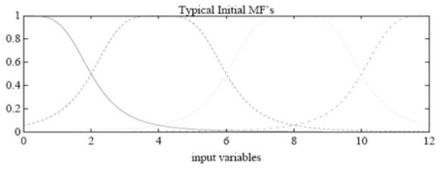


2-input ANFIS with 9 rules

3 membership functions for each input ➔ the input space is partitioned into 9 fuzzy subspaces, each governed by a rule: the antecedent of a rule defines a subspace while the consequent specifies the output within this subspace

Typical initial membership function setting

- The number of membership functions is chosen empirically and/or by trial and error. Typically the initial values of MFs are equally spaced. Moreover the fuzzy system provides smooth transition and sufficient overlapping from one linguistic label to another



Modeling a two-input nonlinear function

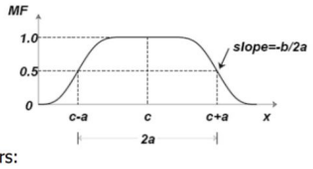
- Use ANFIS to model a nonlinear *sinc* equation

$$z = \mathrm{sinc}(x,y) = \frac{\sin(x)}{x} \times \frac{\sin(y)}{y}$$

- From the grid points of the range $[-10,10]\times[-10,10]$, 121 training data pairs are obtained first

  - As membership function we use the **bell function**

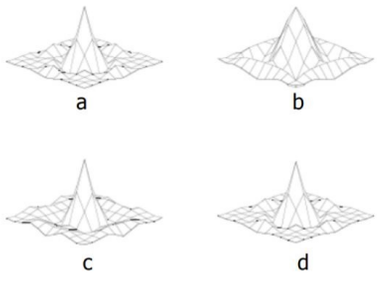
$$\mu_A(x) = \frac{1}{1+\left[\left(\frac{x-c}{a}\right)^2\right]^b}$$



The bell function contains 3 parameters:

- *c* is the center of the function
- *a* is the half width
- *b* (together with *a*) controls the slope at the crossover points (where MF value = 0.5)

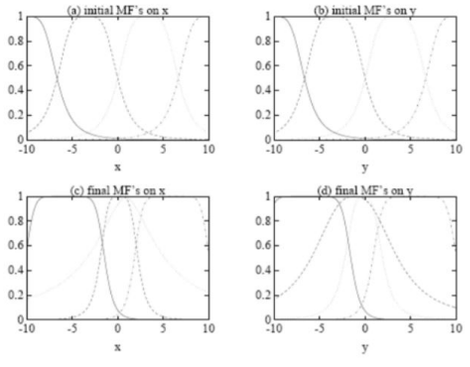
The used ANFIS contains 16 rules, with 4 membership functions for each input variable ➔ # fitting parameters = 72 (24 premise parameters and 48 consequent parameters)



- Training data (a) and reconstructed surfaces at 0.5 (b), 99.5 (c) and 249.5 (d) epochs

  (Note that since the error is computed after the forward pass, the epoch numbers always end with ".5". Further, the surface after 0.5 epochs is due to the identification of consequent parameters only)

Initial and final membership functions

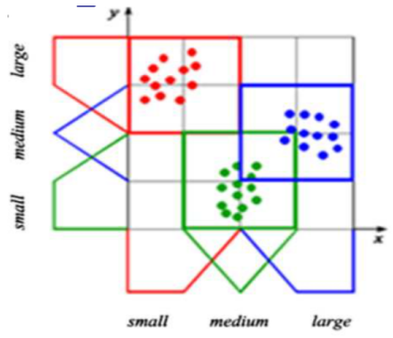


9 Generazione di fuzzy rule

Le regole base di un sistema fuzzy possono essere create dopo una conoscenza accurata dell'argomento. Talvolta questultima potrebbe mancare, o essere parziale. In questi casi è possibile creare una base di regole dal nulla sfruttando un data set appropriato. In generale, una base di regole può includere sia la conoscenza disponibile e nuova conoscenza estratta dai dati.

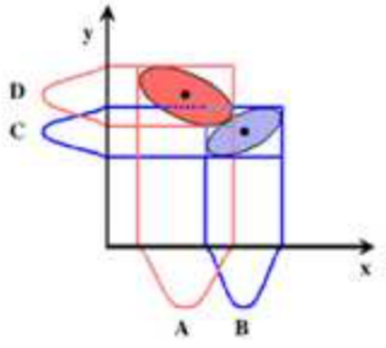
9.1 Approccio structure-oriented

Forniamo dei fuzzy set predefiniti per tutte le variabile, strutturando quindi il data space tramite una griglia fuzzy multidimensionale (hyperboxes sovrapposte); quindi generiamo una regola fuzzy per ogni hyperbox che contiene dei dati.



9.2 Approccio Cluster-oriented

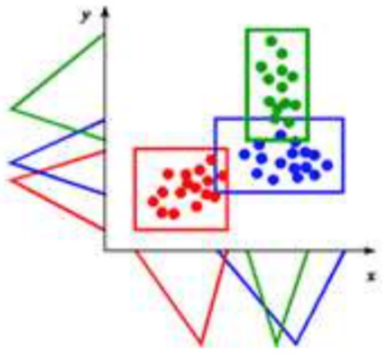
Cerchiamo dei cluster nei dati (il processo di learning non è supervisionato); quindi generiamo una regola fuzzy per ogni cluster proiettando il cluster sulle variabili. Ogni cluster corrisponde ad una regola fuzzy che mette in relazione una regione dell'input space ad una regione dell'output (o una classe do output per la classificazione dei pattern). In questo modo, l'algoritmo di learning crea delle regole e delle funzioni di appartenenza nello stesso momento.



Osserviamo che ci può essere perdita di informazione derivata dalla proiezione: una regola rappresenta un'approssimazione del cluster.

9.3 Approccio hyperbox-oriented

Proviamo a coprire il data space sovrapponendo delle hyperboxes (il processo di learning è supervisionato); quindi creiamo una regola fuzzy per ogni hyperbox proiettando l'hyperbox sulle variabili. Anche in questo caso, le regole fuzzy e i fuzzy set sono creati nello stesso momento.



Osserviamo che non abbiamo perdita di informazione quando le hyperbox sono rappresentate come regole fuzzy.

Se l'obbiettivo principale è quello di creare delle basi di regole fuzzy **interpretabili**, l'approccio structure-oriented è preferibile grazie alla sua alta interpretabilità, mentre gli approcci cluster-oriented e hyperbox-oriented sono probabilmente meno utili, soprattutto perchè:

- Ogni regola fuzzy usa fuzzy sets individuali (mentre nell'approccio structure-oriented tutte le regole condividono gli stessi fuzzy set);
- I fuzzy set ottenuti dalla proiezione sono difficili da interpretare linguisticamente.

Esempi di approcci structure-oriented sono:

- **Algoritmo di Wang e Mendel**;
- **Alberi di decisione fuzzy**: Dopo aver partizionato tutte le variabili in fuzzy sets, lo data space è ricorsivamente partizionato in maniera data-driven e la partizione è rappresentata come un albero. Un albero di decisione può essere trasformato in una base di regole, seguendo ogni percorso dalla radice fino alla foglia.

Introduzione agli Algoritmi Genetici

Prof. Beatrice Lazzerini

Dipartimento di Ingegneria della Informazione
Via Diotisalvi, 2
56122 PISA

ALGORITMI GENETICI (GA)

Sono usati per risolvere problemi di ricerca e ottimizzazione.

Riproducono il processo evolutivo della specie umana.

Considerano una popolazione di cromosomi (individui) che rappresentano soluzioni possibili per un certo problema.

La *qualità* di un individuo (cioè quanto è buona la soluzione per il problema) è misurata mediante una *funzione di fitness*.

In un certo senso, la funzione di fitness indica l'adattabilità all'ambiente: gli individui che meglio si adattano ('fit') hanno piú probabilità di riprodursi e di trasmettere i propri geni alle generazioni future.

Un GA è una procedura di ricerca iterativa il cui scopo è l'ottimizzazione della funzione di fitness.

Partendo da una popolazione iniziale, un GA produce nuove generazioni che contengono (di solito) individui migliori delle precedenti: l'algoritmo evolve verso l'ottimo globale della funzione di fitness.

In realtà, non è garantito che un GA trovi una soluzione ottima globale; un GA è in grado di trovare soluzioni *buone* in tempi *ragionevoli*.

Nel modello tradizionale, i cromosomi sono stringhe di bit di lunghezza fissa e tutte le generazioni hanno la stessa dimensione (numero di individui).

Ogni cromosoma rappresenta un punto nello spazio di ricerca.

I più importanti operatori di ricerca sono la *ricombinazione* (o *crossover*) e la *mutazione*.

Il crossover combina i geni tipicamente di due individui per produrre individui figli che ereditano caratteristiche da entrambi i genitori.

La mutazione reintroduce nella popolazione materiale genetico perduto.

La ricerca genetica realizza un compromesso tra '*exploitation*' della soluzione disponibile migliore ed '*exploration*' dello spazio di ricerca.

Exploitation ed exploration corrispondono, rispettivamente, a *ricerca locale* e *ricerca globale*: exploitation eccessiva può portare l'algoritmo a convergere ad una soluzione non accettabile (la ricerca resta intrappolata in un ottimo locale), exploration eccessiva può non sfruttare appropriatamente la conoscenza già disponibile rendendo il processo di ricerca molto lento (un esempio è la ricerca casuale).

RAPPRESENTAZIONE DEL PROBLEMA

Funzione di codifica

$$C : S \rightarrow X,$$

dove S è lo spazio delle soluzioni del problema ed X è lo spazio dei cromosomi (spazio di ricerca).

Dato

$$c = C(s),\ s \in S,$$

la soluzione *s* può essere interpretata come il fenotipo associato al genotipo (o cromosoma) c.

genoma di un organismo = informazione genetica completa

genotipo = insieme dei geni contenuti in un genoma (cioè, il programma genetico di un individuo ereditato dai genitori e codificato nel DNA di quell'organismo)

fenotipo = realizzazione di un genotipo in un individuo concreto, cioè l'insieme di tutte le caratteristiche osservabili di un organismo.

Ricombinazione e mutazione avvengono a livello di genoma.

FUNZIONE DI FITNESS

Per un problema di ottimizzazione la funzione di fitness può coincidere con la funzione obiettivo (o una sua trasformazione).

Osserviamo che i GA sono procedure di massimizzazione, quindi valori di fitness piú alti sono associati ad individui migliori (problemi di minimizzazione vengono di solito riformulati).

A volte la fitness di un cromosoma è misurata in maniera implicita, valutando la qualitá della corrispondente soluzione rispetto al problema.

Ad esempio, se abbiamo a disposizione un insieme di esempi, la fitness puó essere calcolata in funzione dell'errore della soluzione (differenza tra la soluzione attuale e l'uscita desiderata).

ELEMENTI DI BASE DI UN GA

La nuova generazione P(t+1) è ottenuta dalla popolazione P(t) per mezzo dei seguenti passi:

- *Valutazione*: si valuta la qualitá di ogni individuo (tramite la funzione di fitness).
- *Selezione per riproduzione*: gli individui migliori sono selezionati per la riproduzione. Sono inseriti in una popolazione intermedia P1. Gli individui in P1 entreranno nel *mating pool* con una certa probabilità (probabilità di crossover).
- *Crossover*: si applica l'operatore di crossover agli individui nel mating pool. Si ottiene una nuova popolazione intermedia P2.
- *Mutazione*: l'operatore di mutazione è applicato con una certa probabilità (probabilità di mutazione) agli individui di P2. Viene prodotta una popolazione P3.
- *Selezione per rimpiazzamento e sopravvivenza*: la nuova generazione P(t+1) contiene gli individui della popolazione P3 ma puó includere anche altri individui. Sono possibili piú algoritmi di selezione. Ad esempio, un sottoinsieme di individui di P(t) che non sono stati selezionati per la riproduzione, oppure gli individui migliore di P(t).

GA CANONICO

I cromosomi sono stringhe binarie di lunghezza fissa. Il valore di ogni gene è 0 o 1.

Il numero di cromosomi in ogni generazione è costante.

Il modello GA canonico parte con una arbitraria popolazione iniziale. La nuova generazione P(t+1), ottenuta applicando crossover e mutazione, rimpiazza completamente la generazione precedente.

ALGORITMO GA CANONICO

```
t = 0;
inizializza (casualmente) una popolazione di cromosomi P(t);
valuta P(t) usando una funzione di fitness;
while (condizione di terminazione non soddisfatta) do
    begin
      seleziona individui da P(t) ed inseriscili in P1;
      seleziona individui da P1 ed inseriscili nel mating pool (MP);
      applica il crossover agli individui di MP formando P2;
      applica la mutazione agli individui di P2 formando P3;
      forma P(t+1) selezionando per il rimpiazzamento individui da P3 e
      P(t);
      t = t+1;
    end
```

CONDIZIONE DI TERMINAZIONE

Un'alta percentuale degli individui di una generazione ha la stessa fitness dell' individuo migliore.

Sono state generate *n* generazioni, con *n* fissato.

SOLUZIONE AL PROBLEMA

miglior individuo dell'ultima generazione

miglior individuo su tutte le generazioni

Esempio

Supponiamo di voler massimizzare una funzione di 3 variabili f(x,y,z).

Assumiamo che ogni cromosoma rappresenti un punto <x,y,z> e che sia costituito da 3 geni ognuno dei quali codifica una delle tre variabili. Se ogni variabile è rappresentata con un numero binario di 10 bit, avremo cromosomi lunghi 30 bit. La funzione di fitness sará il valore delle funzione f nel punto rappresentato dal cromosoma.

STRATEGIA DI SELEZIONE PER LA RIPRODUZIONE

La selezione per la riproduzione ha un duplice scopo:

- favorire la riproduzione di individui con fitness alta
- preservare la diversitá della popolazione in modo da esplorare tutte le regioni dello spazio di ricerca

PROPORTIONAL SELECTION

È la strategia di selezione piú usata.

Supponiamo che la popolazione attuale contenga n cromosomi:

$$P(t) = \{x^1, x^2, ..., x^n\}$$

fitness totale F della popolazione

$$F = \sum_{i=1}^{n} f(x^i)$$

probabilità di selezione del cromosoma x^i

$$p_i = \frac{f(x^i)}{F}$$

Algoritmo

Una ruota girevole è divisa in *n* settori ciascuno corrispondente ad un cromosoma. La dimensione dell'i-esimo settore è proporzionale alla probabilità di selezione p_i del cromosoma x^i.

La ruota viene fatta girare *n* volte. Ogni volta si sceglie un cromosoma che viene copiato nella popolazione intermedia P1.

STRATEGIA DI SELEZIONE PER IL RIMPIAZZAMENTO

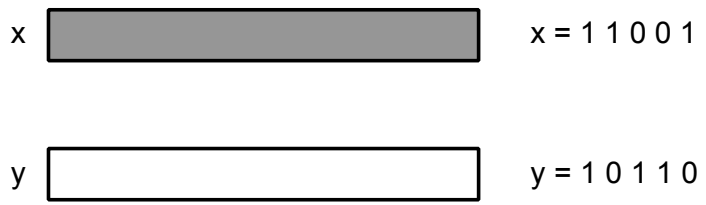
La selezione per il rimpiazzamento ha lo scopo di scegliere quali fra gli individui padri e gli individui figli costituiranno la nuova popolazione.

Nel GA canonico si assume che la generazione P(t+1) sia costituita dai figli (creati per crossover/mutazione) e da quegli individui nella popolazione intermedia P1 che non sono stati scelti per l'accoppiamento (mating). Il numero di individui in ogni generazione è costante.

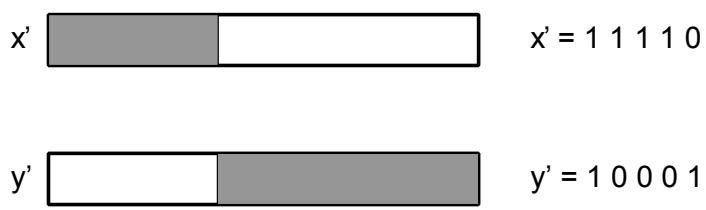
Un'altra possibilità è copiare un numero fissato di individui (i migliori) da P(t) in P(t+1) senza modificarli. Ovviamente, questi individui parteciperanno anche alla selezione per la riproduzione.

ONE-POINT CROSSOVER

Si genera un numero casuale $k \in \{1,2,...,L-1\}$, con L lunghezza dei cromosomi.



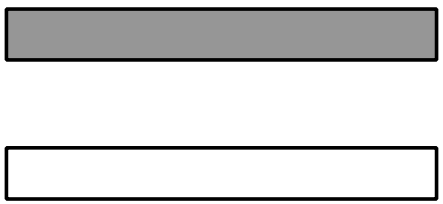
Se k = 2 otteniamo:



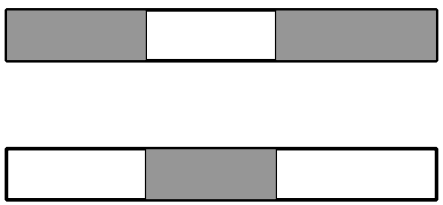
In generale, si considerano due padri che generano due figli.

TWO-POINT CROSSOVER

Si generano casualmente due punti di crossover.



Otteniamo:



Si può generalizzare ottenendo **N-POINT CROSSOVER**.

UNIFORM CROSSOVER

Si usa un parametro globale che indica, per ogni gene di un discendente, la probabilità che tale gene provenga dal primo o dal secondo genitore.

Si può ad esempio per ogni posizione del primo discendente scegliere con una certa probabilità qual è il genitore che fornisce il gene. Il secondo discendente prenderà il gene dall'altro genitore.

Oppure si può calcolare ogni gene di ogni discendente indipendentemente dagli altri permettendo in tal modo ad un genitore di fornire lo stesso gene ad entrambi i figli.

PROBABILITÁ DI CROSSOVER

La probabilità di crossover generalmente assume valori dell'ordine di 10^{-1}.

MATING

Una procedura di mating determina quali cromosomi si accoppiano per generare discendenti.

Tipicamente, i genitori sono scelti a caso con uguale probabilità tra gli individui nel mating pool.

MUTAZIONE

La mutazione modifica un singolo gene di un cromosoma.

Grazie alla mutazione, tutti i possibili valori di ogni gene sono considerati nel processo di ricerca.

PROBABILITÁ DI MUTAZIONE

Valori tipici sono $p_m \in [0.001, 0.01]$.

OPERATORI DI MUTAZIONE

Per ogni gene di ogni cromosoma, si genera un numero casuale *q* con distribuzione uniforme in [0,1]:

- se $q < p_m$ il gene è selezionato per la mutazione;
- se si usa un operatore di *mutazione forte*, il gene viene mutato ($0 \rightarrow 1, 1 \rightarrow 0$); se si usa un operatore di *mutazione debole*, il nuovo valore è scelto casualmente tra 0 e 1 (di fatto, il gene potrebbe alla fine risultare immutato).

CODIFICA REALE

I geni sono codificati come numeri reali.

Cromosoma = vettore a componenti reali.

OPERATORI DI RICOMBINAZIONE PER LA CODIFICA REALE

CROSSOVER DISCRETO (è analogo al crossover uniforme nella codifica binaria):

per ogni posizione *i* del primo discendente si sceglie (con una probabilità *p* fissata) il padre che fornisce quel gene. Il gene corrispondente nel secondo figlio sarà fornito dall'altro genitore. Se $p = 0.5$ il ruolo dei due genitori è simmetrico.

CONVEX CROSSOVER: i discendenti (eventualmente un unico discendente) sono espressi come combinazione convessa di due o più genitori.

Ad esempio, i cromosomi *x* ed *y* possono produrre i discendenti *u* e *v* i cui geni sono

$$u_i = \alpha\ x_i + (1-\alpha)\ y_i$$

$$v_i = \alpha\ y_i + (1-\alpha)\ x_i$$

con $\alpha \in [0,1]$.

DIAGONAL MULTI-PARENT CROSSOVER: crea *p* discendenti da *p* genitori.

Si scelgono *n* punti di crossover. Il primo discendente eredita l'i-esimo segmento dal padre *i*, i = 1, …, *p*. Gli altri discendenti sono costruiti operando una rotazione dei segmenti dei genitori.

OPERATORI DI MUTAZIONE PER LA CODIFICA REALE

L'operatore di mutazione può agire su un unico gene oppure su tutti i geni all'interno di un cromosoma.

Nel primo caso, il gene su cui agire è scelto in modo casuale.

Ogni singolo gene all'interno di un cromosoma può essere rimpiazzato (con una certa probabilità) da un numero reale scelto casualmente all'interno dell'insieme dei valori di quel gene.

Di solito, la probabilità di mutazione è la stessa per tutti i geni.

10 Metodo Wang-Mendel

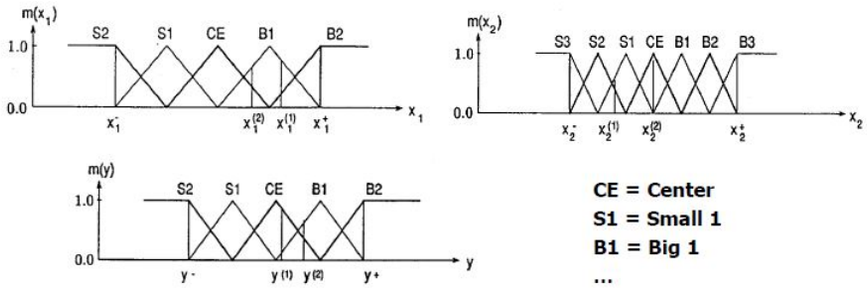
10.1 Generare regole fuzzy dai dati numerici

Supponiamo di avere un dato set di n coppie di input-output:

$$(x_1^{(1)}, x_2^{(1)}; y^{(1)}), (x_1^{(2)}, x_2^{(2)}; y^{(2)}), ..., (x_1^{(n)}, x_2^{(n)}; y^{(n)})$$

dove x_1 e x_2 sono input, e y è l'output.

1. **Dividere gli spazi di input e output in regioni fuzzy** Tipicamente vengono usate funzioni di appartenenza triangolari o trapezoidali equidistanti sovrapposti.



2. **Generare le regole fuzzy dalle coppie di dati fornite**

   - Determinare il grado di appartenenza di ogni variabile ad ogni fuzzy set;
   - Assegnare ad una data variabile la regione con massimo grado;
   - Ricavare una regola per ogni coppia input-output.

$$(x_1^{(1)}, x_2^{(1)}; y^{(1)}) \Rightarrow [x_1^{(1)}(0.8 \text{ in } B1, \max), x_2^{(1)}(0.7 \text{ in } S1, \max); y^{(1)}(0.9 \text{ in } CE, \max)] \Rightarrow \text{Rule 1:}$$

IF x_1 is $B1$ and x_2 is $S1$, THEN y is CE

$$(x_1^{(2)}, x_2^{(2)}; y^{(2)}) \Rightarrow [x_1^{(2)}(0.6 \text{ in } B1, \max), x_2^{(2)}(1 \text{ in } CE, \max); y^{(2)}(0.7 \text{ in } B1, \max)] \Rightarrow \text{Rule 2:}$$

IF x_1 is $B1$ and x_2 is CE, THEN y is $B1$

3. **Assegnare un grado ad ogni regola** Dato che ogni coppia di dati genera una regola ci possono essere delle regole conflittuali, ovvero regole con lo stessa parte IF ma parti THEN differenti. Per risolvere questo problema assegniamo un grado (peso) ad ogni regola (rappresentante l'importanza della regola) e accettiamo solo la regola di massimo grado da un gruppo di regole confittuali.

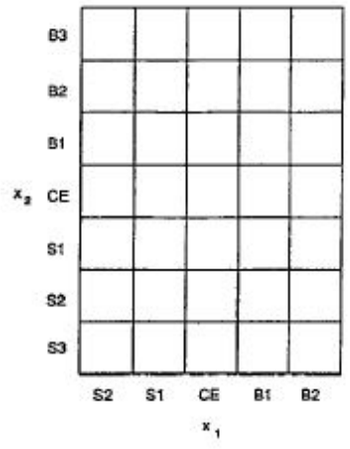
Ad esempio, *product strategy*: il grado di una regola è il prodotto dei gradi dei suoi componenti e, possibilmente, il grado della coppia di dati che genera tale regola:

$$D(Rule1) = m_{B1}(x_1)m_{S1}(x_2)M_{CE}(y)m^{(1)}$$

dove $m^{(1)}$ può rappresentare, ad esempio, la considerazione di un esperto riguardo l'utilità della coppia di dati.

4. **Creare una base di regole fuzzy combinate** Una base di regole fuzzy combinate viene costruita dalle regole generate dai dati numerici e dalle regole linguistiche (una regola linguistica ha un grado assegnato da un esperto).

   Nella figura vediamo la forma di una base di regole fuzzy:



   - Se vi è più di una regola in un riquadro, usiamo la regola del massimo grado;
   - Se una regola è un "and", riempie solo un riquadro;
   - Se la regola è un "or", riempie tutti i riquadri nella riga o colonna corrispondente alle regioni della parte IF (ad esempio, la regola "IF X1 is S1 or X2 is CE THEN Y is B2" riempie tutti i 7 riquadri della colonna S1 e i 5 riquadri della riga CE con B2).

5. **Determinare un mapping basato sulla base di regole fuzzy combinate** Questo step determina un mapping da un input space ad un output space basato sulla base di regole fuzzy combinate usando una procedura di defuzzificamento. Wand e Mendel progettarono il loro algoritmo per creare dei sistemi fuzzy per l'approssimazione di funzione. Provarono inoltre che questo algoritmo può creare delle regole fuzzy che possono approssimare ogni funzione continua su di un set compatto di accuratezza arbitraria.