Reti neurali

Note dal corso di Machine Learning

Corso di Laurea Magistrale in Informatica

a.a. 2010-2011

Prof. Giorgio Gambosi

Università degli Studi di Roma "Tor Vergata"

Queste note derivano da una selezione e rielaborazione dalle seguenti fonti:

- C.M. Bishop "Pattern recognition and machine learning", Springer
- J. Friedman, T. Hastie, R. Tibshirani "The elements of statistical learning", Springer
- C.M. Bishop "Neural networks for pattern recognition", Oxford University Press
- M.E. Tipping "Bayesian Inference: An Introduction to Principles and Practice in Machine Learning" , in "Advanced lectures in machine learning", LNAI 3176, Springer
- Note dal corso CS229 "Machine learning", Stanford University, Prof. A. Ng, 2008
- Note dal corso CS340 "Machine learning", University of British Columbia, Prof. K.P. Murphy, 2007

1

Reti neurali

Le reti neurali sono state inizialmente introdotte con l'obiettivo di modellare il comportamento di insiemi di neuroni all'interno del sistema nervoso. Anche se, nel tempo, ci si è resi conto che esse differiscono sotto diversi aspetti dai sistemi di neuroni, per cui non sembrano costituire un modello adeguato alla simulazione di tali sistemi organici, tuttavia sono venute a rappresentare un modello significativo di algoritmi di ottimizzazione e, in particolare, di apprendimento.

Le reti neurali si rifanno ad un *modello connessionistico* di calcolo, in cui la computazione è effettuata per mezzo dell'interazione tra unità elementari (i neuroni) che effettuano, singolarmente, operazioni molto semplici e si scambiano informazioni per mezzo dei collegamenti che li uniscono. Le ipotesi fondamentali che sono effettuate nelle reti neurali sono le seguenti:

1. L'elaborazione dell'informazione viene effettuata in molte semplici unità, dette neuroni

2. I neuroni si scambiano segnali per mezzo dei collegamenti definiti tra essi

3. Ogni collegamento ha un peso associato, che tipicamente amplifica (moltiplicandolo) il segnale trasmesso

4. Ogni neurone calcola il suo valore in output applicando una *funzione di attivazione*, tipicamente non lineare, al suo input, rappresentato dalla somma dei segnali (amplificati) provenienti dai collegamenti in ingresso

Una rete neurale è quindi caratterizzata da tre elementi: la sua topologia (insieme dei neuroni e collegamenti tra essi), la funzione di attivazione utilizzata dai neuroni (che può essere anche diversa per diversi neuroni), il metodo utilizzato per derivare i pesi associati ai collegamenti a partire da un training set di elementi osservati.

Dal punto di vista della topologia, una rete neurale può essere descritta sotto forma di un grafo orientato, in cui ogni nodo corrisponde a un neurone e ogni arco (orientato) a un collegamento: i nodi senza archi entranti sono detti *nodi di input* e corrispondono alle features i cui valori sono forniti alla rete, i nodi senza archi uscenti sono detti *nodi di output* e forniscono il valore calcolato dalla rete e, infine, i nodi aventi archi sia entranti che uscenti sono detti *nodi interni*.

Caratteristica fondamentale di una rete neurale è il numero di strati *layer* della rete, dove i nodi di input non sono considerati come layer. Un neurone in una rete neurale si trova al layer i-esimo se esiste una sequenza di i archi da un nodo di input al neurone stesso. Tipicamente, le reti neurali hanno una topologia stratificata, per cui per ogni neurone la sequenza di archi da qualunque nodo di input al neurone stesso ha la stessa lunghezza: quindi, in

questo caso, il neurone si trova al layer i-esimo se per ogni nodo di input esiste una sequenza di i archi dal nodo di input al neurone.

Spesso, le reti sono anche *feed-forward*, tali cioè da non prevedere cicli: ciò ci assicura che l'output della rete potrà essere calcolato direttamente, a partire dai valori dei coefficienti e degli input.

1.1 Perceptron

Il *perceptron* è un esempio di modello di discriminazione lineare molto significativo dal punto di vista storico, e alla base delle reti neurali successivamente introdotte. Negli anni '60 (è stato introdotto nel 1962), il perceptron veniva visto come un semplice modello approssimativo del funzionamento dei singoli neuroni all'interno del cervello. Pur nella sua semplicità, il modello mal si presta ad essere valutato dal punto di vista probabilistico.

Il perceptron corrisponde a un modello binario in cui il vettore $\mathbf{x}$ è classificato nel modo seguente

$$y(\mathbf{x}) = \mathcal{S}(\boldsymbol{\beta}^T \mathbf{x})$$

dove $\mathcal{S}$ è una variante della funzione segno

$$\mathcal{S}(i) = \begin{cases} -1 & \text{se } i < 0 \\ 1 & \text{se } i \geq 0 \end{cases}$$

Quindi, $y(\mathbf{x})$ può assumere i soli valori ± 1: identifichiamo $y(\mathbf{x}) = 1$ come $\mathbf{x} \in C_1$ e $y(\mathbf{x}) = -1$ come $\mathbf{x} \in C_2$. Indichiamo con $t(\mathbf{x})$ il valore *target* di $\mathbf{x}$, vale a dire il valore di $y(\mathbf{x})$ che permette di classificare correttamente l'elemento.

Una definizione naturale della funzione di errore, da minimizzare, è rappresentata dal numero di elementi classificati male: in questo caso, però, la funzione di errore sarebbe una funzione costante a tratti, in quanto dato un valore per $\boldsymbol{\beta}$ il corrispondente errore rimane evidentemente costante almeno fino a quando la variazione di $\boldsymbol{\beta}$ non permette di classificare in modo corretto un elemento attualmente mal classificato (o, al contrario di classificare in modo scorretto un elemento attualmente classificato correttamente).

Quanto detto nel seguito si applica anche al caso in cui si considera $\overline{x}$, con $x_0 = 1$ e $\overline{\boldsymbol{\beta}}$, con il coefficiente aggiuntivo β_0.

L'andamento di questa funzione non consente però di applicare metodi di discesa del gradiente per individuare punti di minimo, per tale motivo è preferibile utilizzare una diversa funzione di errore, che risulterà lineare a tratti. Per definire tale funzione, osserviamo che il nostro obiettivo è avere, per ogni $\mathbf{x}$, $\boldsymbol{\beta}^T \mathbf{x} > 0$ se $\mathbf{x} \in C_1$ e $\boldsymbol{\beta}^T \mathbf{x} < 0$ se $\mathbf{x} \in C_2$, e quindi, in generale $\boldsymbol{\beta}^T \mathbf{x} t(\mathbf{x}) > 0$: possiamo definire il contributo alla funzione di errore apportato dall'elemento $\mathbf{x}$ come

1. 0 se $\mathbf{x}$ è classificato in modo corretto
2. $-\boldsymbol{\beta}^T \mathbf{x} t(\mathbf{x}) > 0$ se $\mathbf{x}$ è classificato in modo sbagliato

Indicando con $\mathcal{M}$ l'insieme degli elementi mal classificati, la valutazione complessiva dell'errore è allora

$$E_p(\boldsymbol{\beta}) = -\sum_{\mathbf{x}_i \in \mathcal{M}} \boldsymbol{\beta}^T \mathbf{x}_i t(\mathbf{x}_i)$$

che, si noti, è per l'appunto lineare a tratti, e il cui gradiente è

$$\nabla_{\boldsymbol{\beta}} E_p = \begin{bmatrix} \frac{\partial E_p}{\partial \beta_1} \\ \frac{\partial E_p}{\partial \beta_2} \\ \vdots \\ \frac{\partial E_p}{\partial \beta_d} \end{bmatrix} = \begin{bmatrix} -\sum_{\mathbf{x}_i \in \mathcal{M}} x_{i1} t(\mathbf{x}_i) \\ -\sum_{\mathbf{x}_i \in \mathcal{M}} x_{i2} t(\mathbf{x}_i) \\ \vdots \\ -\sum_{\mathbf{x}_i \in \mathcal{M}} x_{in} t(\mathbf{x}_i) \end{bmatrix} = -\sum_{\mathbf{x}_i \in \mathcal{M}} \mathbf{x}_i t(\mathbf{x}_i)$$

Per trovare il minimo di questa funzione, possiamo applicare un metodo di discesa del gradiente.

$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} - \eta \nabla_{\boldsymbol{\beta}} E_p(\boldsymbol{\beta}^{(k)}) = \boldsymbol{\beta}^{(k)} + \eta \sum_{\mathbf{x}_i \in \mathcal{M}_k} \mathbf{x}_i t(\mathbf{x}_i)$$

dove $\mathcal{M}_k$ indica l'insieme dei punti mal classificati utilizzando l'iperpiano definito da $\boldsymbol{\beta}^{(k)}$.

In particolare, consideriamo una discesa del gradiente on line, in cui il valore della funzione viene ad ogni passo aggiornato tenendo conto di un solo elemento del training set. Questo comporta che, ad ogni iterazione, viene effettuato l'aggiornamento

$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} + \eta \mathbf{x}_i t(\mathbf{x}_i)$$

dove $\mathbf{x}_i \in \mathcal{M}_k$ e $\eta > 0$ è denominato *fattore di scala* e determina, evidentemente, l'impatto di un elemento mal classificato sulla funzione di errore.

L'algoritmo opera quindi nel modo seguente:

Inizializza $\boldsymbol{\beta}^0$
$k := 0$ **repeat**
  $k := k + 1$
  $i := (k \bmod n) + 1$
  $y := \mathcal{S}(\boldsymbol{\beta}^T \mathbf{x}_i) t(\mathbf{x}_i)$
  if $y > 0$ **then** $\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)}$
  else $\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} + \eta \mathbf{x}_i t(\mathbf{x}_i)$
until tutti gli elementi sono ben classificati

Si noti che, ad ogni iterazione, se $\mathbf{x}_i$ è ben classificato allora $\boldsymbol{\beta}^{(k)}$ rimane inalterato, mentre se $\mathbf{x}_i$ è mal classificato, abbiamo che, considerando il contributo di $\mathbf{x}_i$ all'errore,

$$-(\boldsymbol{\beta}^{(k+1)})^T \mathbf{x}_i t(\mathbf{x}_i) = -(\boldsymbol{\beta}^{(k)})^T \mathbf{x}_i t(\mathbf{x}_i) - \eta (\mathbf{x}_i t(\mathbf{x}_i))^T \mathbf{x}_i t(\mathbf{x}_i) < -(\boldsymbol{\beta}^{(k)})^T \mathbf{x}_i t(\mathbf{x}_i)$$

Si ricordi che $(\mathbf{x}_i t(\mathbf{x}_i))^T \mathbf{x}_i t(\mathbf{x}_i) > 0$ per ogni $\mathbf{x}_i \neq \mathbf{0}$.

e quindi il contributo di $\mathbf{x}_i$ all'errore diminuisce. Si noti però che questo di per sé non garantisce che l'errore totale diminuisca, perché ad esempio qualche altro elemento $\mathbf{x}_j$ potrebbe diventare ora mal classificato.

È possibile però mostrare che l'algoritmo converge, in un numero finito di passi alla soluzione corretta, se questa esiste, vale a dire se gli elementi delle due classi sono separabili mediante un iperpiano (*linearmente separabili*).

Infatti, indicando con $\hat{\boldsymbol{\beta}}$ una soluzione, vale a dire un vettore $\boldsymbol{\beta}$ che separa C_1 e C_2, abbiamo che, se $\mathbf{x}_{k+1}$ è l'elemento considerato alla $(k+1)$-esima iterazione e $\mathbf{x}_{k+1}$ è mal classificato, allora

$$\boldsymbol{\beta}^{(k+1)} - \alpha\hat{\boldsymbol{\beta}} = (\boldsymbol{\beta}^{(k)} - \alpha\hat{\boldsymbol{\beta}}) + \eta \mathbf{x}_{k+1} t(\mathbf{x}_{k+1})$$

dove $\alpha > 0$. Quindi,

$$\left|\left|\boldsymbol{\beta}^{(k+1)} - \alpha\hat{\boldsymbol{\beta}}\right|\right|^2 = \left|\left|\boldsymbol{\beta}^{(k)} - \alpha\hat{\boldsymbol{\beta}}\right|\right|^2 + \eta^2 \left|\left|\mathbf{x}_{k+1}\right|\right|^2 + 2\eta(\boldsymbol{\beta}^{(k)} - \alpha\hat{\boldsymbol{\beta}})^T \mathbf{x}_{k+1} t(\mathbf{x}_{k+1})$$

Dato che $\mathbf{x}_{k+1}$ è stato classificato male, $(\boldsymbol{\beta}^{(k)})^T \mathbf{x}_{k+1} t(\mathbf{x}_{k+1}) < 0$, quindi

$$\left|\left|\boldsymbol{\beta}^{(k+1)} - \alpha\hat{\boldsymbol{\beta}}\right|\right|^2 < \left|\left|\boldsymbol{\beta}^{(k)} - \alpha\hat{\boldsymbol{\beta}}\right|\right|^2 + \eta^2 \left|\left|\mathbf{x}_{k+1}\right|\right|^2 - 2\eta\alpha\hat{\boldsymbol{\beta}}^T \mathbf{x}_{k+1} t(\mathbf{x}_{k+1})$$

Indichiamo ora con γ il valore del minimo prodotto scalare tra $\hat{\boldsymbol{\beta}}$ e un qualche elemento $\mathbf{x}_i$, con segno determinato dalla classe di $\mathbf{x}_i$

$$\gamma = \min_i (\hat{\boldsymbol{\beta}}^T \mathbf{x}_i t(\mathbf{x}_i)) = \min_i |\hat{\boldsymbol{\beta}}^T \mathbf{x}_i| > 0$$

inoltre, sia δ la lunghezza del vettore di lunghezza massima tra gli $\mathbf{x}_i$

$$\delta^2 = \max_i ||\mathbf{x}_i||^2$$

Allora,

$$\left|\left|\boldsymbol{\beta}^{(k+1)} - \alpha\hat{\boldsymbol{\beta}}\right|\right|^2 < \left|\left|\boldsymbol{\beta}^{(k)} - \alpha\hat{\boldsymbol{\beta}}\right|\right|^2 + \eta^2\delta^2 - 2\eta\alpha\gamma$$

Ponendo

$$\alpha = \frac{\eta\delta^2}{\gamma}$$

otteniamo

$$\left|\left|\boldsymbol{\beta}^{(k+1)} - \alpha\hat{\boldsymbol{\beta}}\right|\right|^2 < \left|\left|\boldsymbol{\beta}^{(k)} - \alpha\hat{\boldsymbol{\beta}}\right|\right|^2 - \eta^2\delta^2$$

Quindi, il quadrato della distanza tra $\boldsymbol{\beta}^{(k+1)}$ e $\hat{\boldsymbol{\beta}}$ diminuisce di un fattore $\eta^2\delta^2$ ad ogni iterazione, e quindi

$$\left|\left|\boldsymbol{\beta}^{(k+1)} - \alpha\hat{\boldsymbol{\beta}}\right|\right|^2 < \left|\left|\boldsymbol{\beta}^{(0)} - \alpha\hat{\boldsymbol{\beta}}\right|\right|^2 - (k+1)\eta^2\delta^2$$

e, dato che

$$\left|\left|\boldsymbol{\beta}^{(0)} - \alpha\hat{\boldsymbol{\beta}}\right|\right|^2 - (k+1)\eta^2\delta^2 = 0$$

per

$$\overline{k} = \frac{\left|\left|\boldsymbol{\beta}^{(0)} - \alpha\hat{\boldsymbol{\beta}}\right|\right|^2}{\eta^2\delta^2} - 1$$

ne deriva che dopo al più $\overline{k}$ aggiornamenti di $\boldsymbol{\beta}$, è stato determinato un iperpiano di separazione.

Ponendo $\boldsymbol{\beta}^{(0)} = \mathbf{0}$, abbiamo che

$$\overline{k} = \frac{\alpha^2}{\eta^2\delta^2}\left|\left|\hat{\boldsymbol{\beta}}\right|\right|^2 - 1 = \frac{\delta^2}{\gamma^2}\left|\left|\hat{\boldsymbol{\beta}}\right|\right|^2 - 1 = \frac{\max_i ||\mathbf{x}_i||^2}{(\min_i (\hat{\boldsymbol{\beta}}^T \mathbf{x}_i))^2}\left|\left|\hat{\boldsymbol{\beta}}\right|\right|^2 - 1$$

dal che possiamo notare come la ricerca dell'iperpiano di separazione sia più lunga se $\min_i (\hat{\boldsymbol{\beta}}^T \mathbf{x}_i)$ è piccolo, e quindi se esiste un elemento $\mathbf{x}_i$ ortogonale (o quasi) a $\hat{\boldsymbol{\beta}}$.

Occorre ribadire che il perceptron, nella sua semplicità, permette di ottenere classificatori binari soltanto nel caso in cui gli elementi del training set siano linearmente separabili.

▶ 1.2 Reti a più livelli

I modelli visti fino ad ora sono caratterizzati dalla presenza di un solo livello di coefficienti da apprendere: in tutti i casi, la struttura del modello è del tipo $y = f(\boldsymbol{\beta}^T\boldsymbol{\phi}(\mathbf{x}))$, il risultato dell'applicazione dei coefficienti ai valori in input è cioè utilizzato direttamente per generare l'ouput, per mezzo di una funzione non parametrizzata (l'identità per la regressione, una funzione non lineare per la classificazione).

Classi più generali di modelli possono essere ottenute prevedendo sequenze di trasformazioni successive, corrispondenti a reti aventi diversi livelli (*layer*) di funzioni parametrizzate per mezzo di coefficienti.

Nel seguito, considereremo soltanto reti a due livelli, che risultano sufficientemente potenti da approssimare qualunque funzione continua. La sola restrizione che applichiamo è che la rete sia *feed-forward*.

In altri termini, in una rete neurale viene introdotta la possibilità che le funzioni base siano parametrizzate per mezzo di coefficienti adattabili, e che possono essere appresi: la rete neurale è quindi espressa mediante una serie di trasformazioni funzionali.

Dato un vettore d-dimensionale $\mathbf{x}$ di input, una rete neurale prevede $M > 0$ combinazioni lineari degli elementi $x_1, \ldots, x_d$ di $\mathbf{x}$

$$a_j^{(1)} = \sum_{i=1}^{d} \beta_{ji}^{(1)} x_i + \beta_{j0}^{(1)} = (\boldsymbol{\beta}_j^{(1)})^T \mathbf{x} + \beta_{j0}^{(1)} = (\overline{\boldsymbol{\beta}}_j^{(1)})^T \overline{\mathbf{x}}$$

dove M è un parametro predefinito. Su ogni valore $a_j^{(1)}$, denominato *attivazione*, è quindi applicata una *funzione di attivazione* non lineare $h_1()$

$$z_j^{(1)} = h_1(a_j^{(1)})$$

dove $h_1()$ è tipicamente una approssimazione della funzione soglia, come la sigmoide

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

o la funzione tangente iperbolica

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1}{1 + e^{-2x}} - \frac{1}{1 + e^{2x}} = \sigma(2x) - \sigma(-2x)$$

La computazione di

$$z_j^{(1)} = h_1((\overline{\boldsymbol{\beta}}_j^{(1)})^T \overline{\mathbf{x}})$$

viene considerata come effettuata dalla j-esima *unità interna* (*hidden unit*) della rete. Ai valori calcolati dalle unità interne sono quindi applicate $K > 0$ combinazioni lineari

$$a_k^{(2)} = \sum_{i=1}^{M} \beta_{ki}^{(2)} a_i^{(1)} + \beta_{k0}^{(2)} = (\boldsymbol{\beta}_k^{(2)})^T \mathbf{z}^{(1)} + \beta_{k0}^{(2)} = (\overline{\boldsymbol{\beta}}_k^{(2)})^T \overline{\mathbf{z}}^{(1)}$$

dove $\mathbf{z} = (z_1^{(1)}, \ldots, z_M^{(1)})^T = (h_1(a_1^{(1)}), \ldots, h_1(a_M^{(1)}))^T$. Ad ogni valore di attivazione $a_k^{(2)}$ è quindi applicata una funzione di attivazione $h_2()$, che fornisce il valore di output $y_k = h_2(a_k^{(2)})$: la funzione $h_2()$ è tipicamente scelta sulla base del tipo di problema considerato, e corrisponde alla funzione *canonica* per la distribuzione esponenziale assunta per il modello. Così, nel caso della regressione lineare $h_2()$ (distribuzione normale) è l'identità, mentre nel caso della classificazione binaria (distribuzione di Bernoulli) $h_2()$ è la sigmoide e, infine, nel caso di classificazione a più classi (distribuzione multinomiale), $h_2()$ è una softmax.

Allo stesso modo che nel caso delle unità interne, possiamo assumere che il calcolo di

$$y_k = h_2((\overline{\boldsymbol{\beta}}_k^{(2)})^T \overline{\mathbf{z}}^{(1)})$$

sia effettuato da una *unità di output* (o *output unit*). La figura 1 mostra la struttura risultante per la rete.

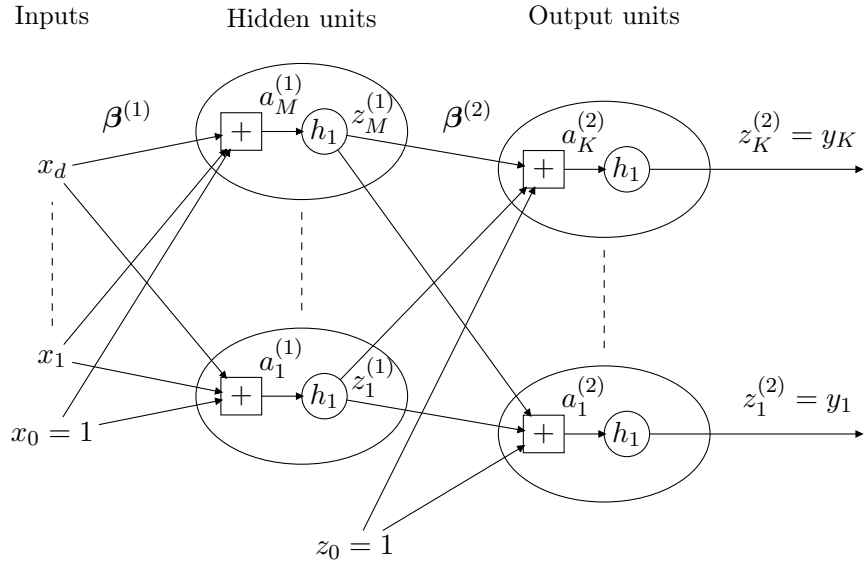

Figura 1. Struttura di una rete neurale a due livelli.

In definitiva, componendo le varie espressioni, vediamo come la rete neurale restituisce, a partire da un vettore d-dimensionale $\mathbf{x}$, un vettore K-dimensionale $\mathbf{y}$ tale che

$$y_k = h_2\left(\sum_{j=1}^{M} \beta_{kj}^{(2)} h_1\left(\sum_{i=1}^{d} \beta_{ji}^{(1)} x_i + \beta_{j0}^{(1)}\right) + \beta_{k0}^{(2)}\right)$$

e, come si può verificare, la rete neurale si può vedere come un modello lineare generalizzato in cui le funzioni base sono parametrizzate dai coefficienti $\boldsymbol{\beta}^{(1)}$.

Ad esempio, assumendo che sia $h_1()$ che $h_2()$ siano sigmoidi, avremo che la rete neurale determina y_k per mezzo di una logistic regression

$$y_k = \sigma\left((\boldsymbol{\beta}_k^{(2)})^T \boldsymbol{\phi}\right)$$

dove

$$\boldsymbol{\phi} = \begin{pmatrix} \sigma(\sum_{i=1}^{d} \beta_{1i}^{(1)} x_i + \beta_{10}^{(1)}) \\ \sigma(\sum_{i=1}^{d} \beta_{2i}^{(1)} x_i + \beta_{20}^{(1)}) \\ \vdots \\ \sigma(\sum_{i=1}^{d} \beta_{Mi}^{(1)} x_i + \beta_{M0}^{(1)}) \end{pmatrix} = \begin{pmatrix} \sigma((\overline{\boldsymbol{\beta}}_1^{(1)})^T \overline{\mathbf{x}}) \\ \sigma((\overline{\boldsymbol{\beta}}_2^{(1)})^T \overline{\mathbf{x}}) \\ \vdots \\ \sigma((\overline{\boldsymbol{\beta}}_M^{(1)})^T \overline{\mathbf{x}}) \end{pmatrix}$$

Le reti neurali, pur nella loro semplice struttura, sono *approssimatori universali*: è possibile mostrare che qualunque funzione continua può essere approssimata per mezzo di una opportuna rete neurale a due livelli con funzioni di attivazione sigmoidali, dotata di un opportuna quantità di unità interne. Ciò può essere informalmente verificato considerando il caso di due variabili di input x_1, x_2 e una variabile di output y.

Sappiamo che, per ogni valore di x_1, la funzione $y(x_1, x_2)$ può essere arbitrariamente approssimata per mezzo di una serie di Fourier, vale a dire di una serie di funzioni armoniche (ad esempio $\cos x$) pesate da coefficienti opportuni. Nel nostro caso i coefficienti dipenderanno da x_1, mentre le funzioni saranno definite su x_2.

$$y(x_1, x_2) \simeq \sum_s a_s(x_1) \cos(s x_2)$$

Gli stessi coefficienti a_s possono essere espressi come serie di Fourier

$$a_s \simeq \sum_t a_{st} \cos(t x_1)$$

dal che deriva

$$y(x_1, x_2) \simeq \sum_s \sum_t a_{st} \cos(tx_1) \cos(sx_2)$$

Ricordando che $\cos\alpha \cos\beta = \frac{1}{2}(\cos(\alpha+\beta) + \cos(\alpha-\beta))$ abbiamo che $y(x_1, x_2)$ può essere approssimato da una somma di termini del tipo $\cos\gamma_{st}$ e $\cos\overline{\gamma}_{st}$, dove $\gamma_{st} = s\alpha + t\beta$ e $\overline{\gamma}_{st} = s\alpha - t\beta$.

Ognuna delle funzioni $\cos\gamma_{st}, \cos\overline{\gamma}_{st}$ può essere approssimata a livello arbitrario da un numero opportuno di funzioni soglia (a loro volta approssimate da sigmoidi). Ciò può essere verificato osservando la figura 2, in cui è mostrato come una funzione continua $f(z)$ venga approssimata da un insieme di funzioni soglia opportunamente numeroso,

$$f(z) \simeq f(z_0) + \sum_{i=1}^{N} (f(z_{i+1}) - f(z_i)) H(z - z_i)$$

dove $H(z)$ è la funzione soglia (di Heaviside) tale che $H(z') = 0$ per $z' < 0$ e $H(z') = 1$ per $z' \geq 0$.

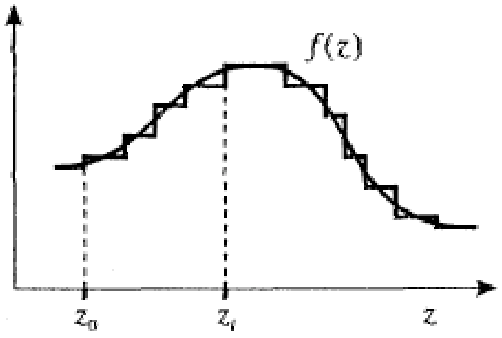

Figura 2. Approssimazione di una funzione continua per mezzo di funzioni soglia..

1.3 Massima verosimiglianza e reti neurali

La fase di addestramento di una rete neurale, che assumiamo sia a due livelli e comprenda M unità interne, corrisponde all'apprendimento, a partire da un training set $\mathcal{T} = \{(\mathbf{x}_1, \mathbf{t}_1), (\mathbf{x}_2, \mathbf{t}_2), \ldots, (\mathbf{x}_n, \mathbf{t}_n)\}$ dei valori dei coefficienti $\boldsymbol{\beta} = \boldsymbol{\beta}^{(1)} \cup \boldsymbol{\beta}^{(2)}$.

Come al solito, l'apprendimento viene effettuato minimizzando una funzione di errore dipendente dalle ipotesi probabilistiche poste, in funzione del problema considerato: tale funzione di errore, nel criterio di maximum likelihood, è tale che la sua minimizzazione equivale alla massimizzazione della verosimiglianza. Se ad esempio consideriamo il caso della regressione e assumiamo $K = 1$ (vale a dire un solo valore in output), una comune ipotesi probabilistica è che per ogni elemento $(\mathbf{x}_i, t_i)$ del training set il valore $y_i = h(\mathbf{x}_i)$ fornito dalla rete sia distribuito come una gaussiana di media pari al valore corretto t_i e varianza σ^2 da determinare. Ciò equivale ad assumere che, dato $y_i = y(\mathbf{x}_i, \mathbf{B})$, il valore sconosciuto t_i sia distribuito come una gaussiana, con medesima varianza σ^2 e media y_i. Quindi, la probabilità è data da

$\mathbf{B}$ è l'insieme dei coefficienti $\beta_{ij}^{(k)}$.

$$p(t_i|\mathbf{x}_i, \mathbf{B}, \sigma^2) = \mathit{Normal}(t_i|y(\mathbf{x}_i, \mathbf{B}), \sigma^2)$$

sull'intero training set abbiamo poi, assumendo l'indipendenza degli elementi, che la verosimiglianza risulta

$$L(\mathbf{t}|\mathbf{X}, \mathbf{B}, \sigma^2) = \prod_{i=1}^{n} p(t_i|\mathbf{x}_i, \mathbf{B}, \sigma^2)$$

X è l'insieme dei vettori di input $\mathbf{x}_i$, $\mathbf{t}$ è il vettore dei valori in output..

Considerando, come al solito, la log-verosimiglianza abbiamo

$$\begin{aligned} l(\mathbf{t}|\mathbf{X},\mathbf{B},\sigma^2) &= \ln L(\mathbf{t}|\mathbf{X},\mathbf{B},\sigma^2) \\ &= \ln \prod_{i=1}^{n} \left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y(\mathbf{x}_i,\mathbf{B}) - t_i)^2}{2\sigma^2} \right) \right) \\ &= \sum_{i=1}^{n} \ln \left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y(\mathbf{x}_i,\mathbf{B}) - t_i)^2}{2\sigma^2} \right) \right) \\ &= \sum_{i=1}^{n} \left(\frac{1}{2}\ln(2\pi\sigma^2) - \frac{(y(\mathbf{x}_i,\mathbf{B}) - t_i)^2}{2\sigma^2} \right) \\ &= \frac{n}{2}\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y(\mathbf{x}_i,\mathbf{B}) - t_i)^2 \end{aligned}$$

Chiaramente, massimizzare tale funzione rispetto a $\mathbf{B}$ equivale a minimizzare la funzione di errore

$$E(\mathbf{B}) = \frac{1}{2}\sum_{i=1}^{n}(y(\mathbf{x}_i,\mathbf{B}) - t_i)^2$$

nella quale però, a differenza che nel caso della regressione lineare, il termine $y(\mathbf{x})$ non è lineare. Ciò comporta che la funzione non abbia, in generale, un unico minimo, ma diversi minimi locali.

Osserviamo che, trovati valori di massima verosiglianza $\mathbf{B}_{ML}$ (o quantomeno dei valori corrispondenti ad un massimo locale della verosimiglianza), il valore della varianza è derivabile osservando che

$$\begin{aligned} \frac{\partial l(\mathbf{t}|\mathbf{X},\mathbf{B},\sigma^2)}{\partial\sigma^2} &= \frac{\partial}{\partial\sigma^2}\left(\frac{n}{2}\ln(2\pi\sigma^2)\right) - \frac{\partial}{\partial\sigma^2}\left(\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y(\mathbf{x}_i,\mathbf{B}_{ML}) - t_i)^2\right) \\ &= \frac{n}{2\sigma^2} - \frac{1}{2(\sigma^2)^2}\sum_{i=1}^{n}(y(\mathbf{x}_i,\mathbf{B}_{ML}) - t_i)^2 \end{aligned}$$

che, posto pari a 0, ci fornisce

$$\sigma^2_{ML} = \frac{1}{n}\sum_{i=1}^{n}(y(\mathbf{x}_i,\mathbf{B}_{ML}) - t_i)^2$$

Nel caso $K > 1$ di più output, quanto sopra può essere generalizzato modellando la distribuzione di $\mathbf{t}_i$ come una gaussiana multivariata di media $\mathbf{y}_i = \mathbf{y}(\mathbf{x}_i,\mathbf{B})$, la cui verosimiglianza sarà

$$L(\mathbf{t}_i|\mathbf{x}_i,\mathbf{B},\sigma^2) = \mathit{Normal}(\mathbf{t}_i|\mathbf{y}(\mathbf{x}_i,\mathbf{B}),\sigma^2\mathbf{I}) = \prod_{j=1}^{K}\mathit{Normal}(t_{ij}|y_j(\mathbf{x}_i,\mathbf{B}),\sigma^2)$$

La verosimiglianza rispetto al training set è allora

$$L(\mathbf{T}|\mathbf{X},\mathbf{B},\sigma^2) = \prod_{i=1}^{n} p(\mathbf{t}_i|\mathbf{x}_i,\mathbf{B},\sigma^2) = \prod_{i=1}^{n}\prod_{j=1}^{K}\mathit{Normal}(t_{ij}|y_j(\mathbf{x}_i,\mathbf{B}),\sigma^2)$$

e la log-verosimiglianza

$$\begin{aligned} l(\mathbf{T}|\mathbf{X},\mathbf{B},\sigma^2) &= \ln L(\mathbf{T}|\mathbf{X},\mathbf{B},\sigma^2) \\ &= \ln \prod_{i=1}^{n}\prod_{j=1}^{K}\left(\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y_j(\mathbf{x}_i,\mathbf{B})-t_{ij})^2}{2\sigma^2}\right)\right) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{K}\ln\left(\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y_j(\mathbf{x}_i,\mathbf{B})-t_{ij})^2}{2\sigma^2}\right)\right) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{K}\left(\frac{1}{2}\ln(2\pi\sigma^2)-\frac{(y_j(\mathbf{x}_i,\mathbf{B})-t_{ij})^2}{2\sigma^2}\right) \\ &= \frac{nK}{2}\ln(2\pi\sigma^2)-\frac{1}{2\sigma^2}\sum_{i=1}^{n}\sum_{j=1}^{K}(y_j(\mathbf{x}_i,\mathbf{B})-t_{ij})^2 \end{aligned}$$

La funzione di errore da minimizzare è allora in questo caso

$$E(\mathbf{B}) = \sum_{i=1}^{n}\sum_{j=1}^{K}(y_j(\mathbf{x}_i,\mathbf{B})-t_{ij})^2$$

Applicando le medesime considerazioni per quanto riguarda la valutazione sia di $\mathbf{B}_{ML}$ che di σ^2_{ML}, che risulta pari a

$$\sigma^2_{ML} = \frac{1}{nK}\sum_{i=1}^{n}||\mathbf{y}(\mathbf{x}_i,\mathbf{B}_{ML})-\mathbf{t}_i||^2$$

Per motivi che saranno utili nel seguito, osserviamo ora come, per ogni singolo elemento $(\mathbf{x},\mathbf{t})$ del training set la derivata della funzione di errore $E(\mathbf{B})$ rispetto alla combinazione lineare $a_k^{(2)} = (\overline{\boldsymbol{\beta}}_k^{(2)})^T\mathbf{z}^{(1)}$ effettuata dalla k-esima unità di output sia data da

$$\frac{\partial E(\mathbf{B})}{\partial a_k^{(2)}} = \frac{\partial}{\partial a_k^{(2)}}\left(\frac{1}{2}\sum_{j=1}^{K}(y_j-t_j)^2\right)$$

e, dato che nel caso della regressione lineare $y_j = a_j^{(2)}$ per ogni $j = 1,\ldots,K$,

$$\frac{\partial E(\mathbf{B})}{\partial a_k^{(2)}} = \frac{\partial}{\partial a_k^{(2)}}\left(\frac{1}{2}\sum_{i=1}^{K}(a_j^{(2)}-t_j)^2\right) = a_k^{(2)}-t_k = y_k-t_k$$

Nel caso di classificazione binaria, l'unità di output ha funzione di attivazione $h_2(z) = \sigma(z)$. Posto $y(\mathbf{x},\mathbf{B}) = p(C_1|\mathbf{x})$, la probabilità condizionata del target è espressa dalla distribuzione di Bernoulli

$$p(t|\mathbf{x},\mathbf{B}) = y(\mathbf{x},\mathbf{B})^t(1-y(\mathbf{x},\mathbf{B}))^{1-t}$$

e, sull'intero training set, la verosimiglianza sarà

$$L(\mathbf{t}|\mathbf{X},\mathbf{B}) = \prod_{i=1}^{n}y(\mathbf{x}_i,\mathbf{B})^{t_i}(1-y(\mathbf{x}_i,\mathbf{B}))^{1-t_i}$$

e la log-verosimiglianza, denominata *cross-entropy* è

$$l(\mathbf{t}|\mathbf{X},\mathbf{B}) = \sum_{i=1}^{n}(t_i\ln y_i+(1-t_i)\ln(1-y_i))$$

La funzione di errore è evidentemente $E(\mathbf{B}) = -l(\mathbf{t}|\mathbf{X}, \mathbf{B})$ e, per quanto riguarda la derivata rispetto a $a_k^{(2)}$ abbiamo che

$$\frac{\partial E(\mathbf{B})}{\partial a_k^{(2)}} = -t_k \frac{1}{y_k} \frac{\partial y_k}{\partial a_k^{(2)}} + (1 - t_k) \frac{1}{1 - y_k} \frac{\partial y_k}{\partial a_k^{(2)}}$$

e, dato che $y_k = \sigma(a_k^{(2)})$ allora

$$\frac{\partial y_k}{\partial a_k^{(2)}} = \frac{\partial \sigma(a_k^{(2)})}{\partial a_k^{(2)}} = \sigma(a_k^{(2)})(1 - \sigma(a_k^{(2)})) = y_k(1 - y_k)$$

dal che concludiamo che

$$\begin{aligned} \frac{\partial E(\mathbf{B})}{\partial a_k^{(2)}} &= -t_k \frac{1}{y_k} y_k(1 - y_k) + (1 - t_k) \frac{1}{1 - y_k} y_k(1 - y_k) \\ &= -t_k(1 - y_k) + (1 - t_k) y_k \\ &= y_k - t_k \end{aligned}$$

ottenendo quindi la stessa proprietà che nel caso della regressione.

È possibile mostrare che tale proprietà vale anche nel caso in cui $h_2()$ è una softmax, in quanto il problema è quello della classificazione su più classi.

La funzione di errore in questo caso, come visto in precedenza, è

$$E(\mathbf{B}) = -\sum_{j=1}^{K} t_j \log s_j$$

dove

$$s_j = \frac{\exp(a_j^{(2)})}{\sum_{i=1}^{K} \exp(a_i^{(2)})}$$

Allora,

$$\begin{aligned} \frac{\partial E(\mathbf{B})}{\partial a_k^{(2)}} &= -\frac{\partial}{\partial a_k^{(2)}} \sum_{j=1}^{K} t_j \log s_j = -\sum_{j=1}^{K} \frac{t_j}{s_j} \frac{\partial s_j}{\partial a_k^{(2)}} \\ &= \sum_{j \neq k} t_j s_k - t_k(1 - s_k) = s_k \sum_{j=1}^{K} t_j - t_k \\ &= s_k - t_k = y_k - t_k \end{aligned}$$

in quanto $\sum_{i=1}^{K} t_i = 1$ e $y_k = s_k$.

1.4 Back-propagation

Data una rete neurale le cui funzioni di attivazione sono derivabili, allora le attivazioni delle unità di output risultano derivabili sia rispetto alle variabili in input che rispetto ai coefficienti. Se consideriamo una funzione di errore, come quelle viste sopra, derivabile rispetto alle variabili di output, allora essa risulta derivabile anche rispetto ai coefficienti.

Risulta quindi possibile valutare le derivate della funzione di errore rispetto ai coefficienti, in modo tale da individuare i valori dei coefficienti stessi che la minimizzano, utilizzando ad esempio un metodo di discesa del gradiente.

L'algoritmo utilizzato per valutare le derivate dell'errore rispetto ai coefficienti è detto back-propagation in quanto, come si vedrà, può essere interpretato in termini di propagazione dell'errore all'indietro nella rete, vale a dire dalle unità di output verso le unità interne.

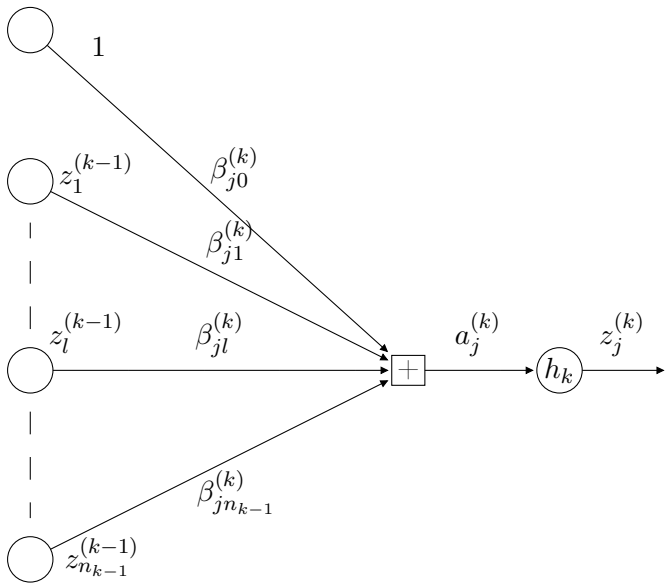

Figura 3. Struttura locale della rete.

In generale, un numero consistente di algoritmi di apprendimento sono incentrati su una qualche procedura di minimizzazione di una funzione di errore: dato che solo in casi fortunati tale minimizzazione può essere risolta in modo analitico, tipicamente la procedura comporta un aggiustamento iterativo di valori, come nel caso della discesa del gradiente o anche del metodo di Newton-Raphson. Ad ogni passo dell'iterazione, possiamo distringuere due fasi: nel corso della prima, vengono valutate le derivate dell'errore rispetto ai coefficienti, mentre nella seconda fase i valori risultanti per le derivate sono utilizzati per calcolare gli aggiustamenti da apportare ai coefficienti.

Un aspetto fondamentale della back-propagation è che essa fornisce un metodo efficiente per calcolare tali derivate, per mezzo di una propagazione degli errori all'indietro, all'interno della rete. In effetti, questo meccanismo può essere applicato più in generale per il calcolo di derivate, come ad esempio per il calcolo delle matrici Jacobiane e Hessiane.

La determinazione degli aggiustamenti dei coefficienti, nella seconda fase, può essere effettuata applicando tecniche diverse, eventualmente più sofisticate della semplice discesa del gradiente considerata qui.

Una matrice Jacobiana comprende le derivate di ogni variabile in output rispetto a ogni variabile di input. La matrice Hessiana è la matrice delle derivate seconde dell'errore rispetto ai coefficienti.

Consideriamo una rete neurale feed-forward con topologia arbitraria e funzioni di attivazione derivabili; consideriamo inoltre una funzione di errore derivabile. Ogni unità, come visto, calcola il valore di output come somma pesata degli input (assumiamo $z_0^{(k-1)} = 1$ per tener conto del coefficiente $\beta_{j0}^{(k)}$)

$$a_j^{(k)} = \sum_{i=1}^{m} \beta_{ji}^{(k)} z_i^{(k-1)}$$

Il valore di attivazione $a_j^{(k)}$ è quindi trasformato mediante applicazione di una funzione di attivazione non lineare $h_k()$ nel valore $z_j^{(k)}$ in input all'unità successiva (si veda la figura 3):

$$z_j^{(k)} = h_k(a_j^{(k)})$$

Si noti che una qualunque variabile z_i potrebbe essere di input alla rete, o di output da una unità precedente, così come a_j potrebbe anche restituito direttamente come output della rete.

L'ipotesi, poco limitativa, che facciamo sulla funzione di errore è che essa sia esprimibile, dato un training set, come somma degli errori relativi ai singoli elementi del training set

$$E(\boldsymbol{\beta}) = \sum_{i=1}^{n} E_i(\boldsymbol{\beta})$$

le funzioni di errore che abbiamo considerato hanno tutte questa caratteristica. Assumiamo inoltre, come preannunciato, che E_i sia una funzione derivabile, evidentemente allora anche E sarà derivabile e la sua derivata sarà la somma delle derivate delle funzioni E_i.

Dato l'elemento i-esimo del training set, $(\mathbf{x}_i, \mathbf{t}_i)$, supponiamo di avere fornito in input alla rete i relativi valori in $\mathbf{x}$ e di avere determinato i valori di attivazione di tutte le unità (interne e di output) nella rete stessa: questo processo viene denominato *forward propagation.*

Vogliamo determinare la derivata di E_i rispetto a un coefficiente $\beta_{jl}^{(k)}$ che, si ricorda, "pesa" il contributo dell'output della l-esima unità al livello $k-1$ alla j-esima unità al livello k-esimo (o al j-esimo output). Notiamo, per iniziare, come E_i dipenda da $\beta_{jl}^{(k)}$ soltanto per mezzo della somma

$$a_j^{(k)} = \sum_{r=1}^{m} \beta_{jr}^{(k)} z_r^{(k-1)}$$

Quindi, possiamo scrivere

$$\frac{\partial E_i}{\partial \beta_{jl}^{(k)}} = \frac{\partial E_i}{\partial a_j^{(k)}} \frac{\partial a_j^{(k)}}{\partial \beta_{jl}^{(k)}}$$

Indichiamo ora con $\delta_j^{(k)}$ la derivata di E_i rispetto a a_j^k:

$$\delta_j^{(k)} = \frac{\partial E_i}{\partial a_j^{(k)}}$$

e osserviamo che

$$\frac{\partial a_j^{(k)}}{\partial \beta_{jl}^{(k)}} = \frac{\partial}{\partial \beta_{jl}^{(k)}} \sum_{r=1}^{m} \beta_{jr}^{(k)} z_r^{(k-1)} = z_l^{(k-1)}$$

Quindi

$$\frac{\partial E_i}{\partial \beta_{jl}^{(k)}} = \delta_j^{(k)} z_l^{(k-1)}$$

la derivata considerata risulta quindi uguale al prodotto tra il valore in input all'arco della rete etichettato con $\beta_{jl}^{(k)}$ e la derivata della medesima funzione rispetto al valore calcolato in fondo all'arco. Quindi, per calcolare le derivate rispetto a tutti i coefficienti, è necessario calcolare i valori $\delta_j^{(k)}$ per tutte le unità della rete.

Nel caso in cui $z_j^{(k)}$ sia un output della rete, e quindi $z_j^{(k)} = y_j$, sappiamo da quanto osservato nella sezione precedente che, almeno per la regressione lineare, la regressione logistica e la softmax,

$$\delta_j^{(k)} = \frac{\partial E_i}{\partial a_j^{(k)}} = y_j - t_j$$

Nel caso di unità interne, possiamo osservare che una variazione di $a_j^{(k)}$ possono avere un effetto su E_i soltanto inducendo variazioni per tutte le variabili $a_l^{(k+1)}$,

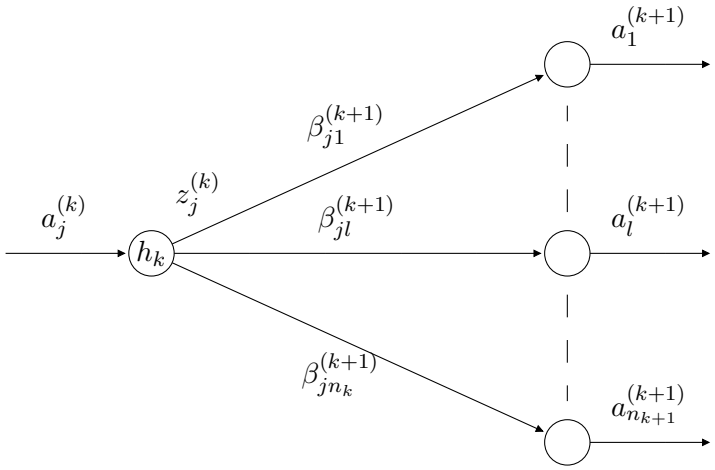

Figura 4. Relazione tra $a_j^{(k)}$ e $a_r^{(k+1)}$.

e l'effetto su E_i sarà dipendente dalla somma delle variazioni delle $a_r^{(k+1)}$. Quindi,

$$\delta_j^{(k)} = \frac{\partial E_i}{\partial a_j^{(k)}} = \sum_{r=1}^{n_{k+1}} \frac{\partial E_i}{\partial a_r^{(k+1)}} \frac{\partial a_r^{(k+1)}}{\partial a_j^{(k)}} = \sum_{r=1}^{n_{k+1}} \delta_r^{(k+1)} \frac{\partial a_r^{(k+1)}}{\partial a_j^{(k)}}$$

Dato che, come noto,

$$a_r^{(k+1)} = \sum_l \beta_{rl}^{(k+1)} z_l^{(k)}$$

e

$$z_j^{(k)} = h_k(a_j^{(k)})$$

abbiamo che

$$\frac{\partial a_r^{(k+1)}}{\partial a_j^{(k)}} = \frac{\partial a_r^{(k+1)}}{\partial z_j^{(k)}} \frac{\partial z_j^{(k)}}{\partial a_j^{(k)}} = \beta_{rj}^{(k+1)} h_k'(a_j^{(k)})$$

e quindi, in definitiva,

$$\delta_j^{(k)} = h_k'(a_j^{(k)}) \sum_{r=1}^{n_{k+1}} \delta_r^{(k+1)} \beta_{rj}^{(k+1)}$$

Questa espressione può essere valutata, quindi, conoscendo i valori $\beta_{rj}^{(k+1)}$ (fissati, per il singolo passo di back propagation), il valore $a_j^{(k)}$ (calcolato a partire da $\boldsymbol{\beta}$ e dall'input) e i valori $\delta_r^{(k+1)}$ relativi al livello $k+1$-esimo della rete. Ricordando che i valori δ sono determinabili, per le unità di output, a partire dal valore calcolato dalla rete e dal target, ne deriva che i δ relativi a tutte le unità possono essere determinati mediante una propagazione dei valori calcolati dagli ultimi livelli verso i primi livelli della rete, vale a dire in direzione opposta a quella di calcolo dell'output a partire dall'input.

La procedura di back-propagation opera quindi nel modo seguente (ad esempio per una rete a due livelli):

1. Fornire un vettore $\mathbf{x}_i$ in input alla rete: propagare l'informazione nella rete fino a determinare tutti i valori $a_j^{(k)}$ e $z_j^{(k)}$, per $K = 1, 2$, ivi inclusi quindi gli output $y_j = z_j^{(2)}$

Indichiamo $z_j^{(2)}$ come y_j e $z_j^{(1)}$ come z_j.

2. A partire dai valori di output restituiti e dai target, determinare i valori $\delta_j^{(2)} = y_j - t_j$ per tutte le unità di output

3. Determinare i valori $\delta_j^{(1)}$ per tutte le unità interne, utilizzando la relazione

$$\delta_j^{(1)} = h_1'(a_j^{(1)}) \sum_{i=1}^{K} \beta_{ij}^{(2)} \delta_i^{(2)} = h_1'(a_j^{(1)}) \sum_{i=1}^{K} \beta_{ij}^{(2)} (y_j - t_j)$$

   che, nel caso comune di $h_1(x) = \sigma(x)$, risulta

$$\delta_j^{(1)} = \sigma(a_j^{(1)})(1 - \sigma(a_j^{(1)})) \sum_{i=1}^{K} \beta_{ij}^{(2)} (y_j - t_j) = z_j(1 - z_j) \sum_{i=1}^{K} \beta_{ij}^{(2)} (y_j - t_j)$$

4. Per ogni coefficiente $\beta_{jl}^{(k)}$, con $k = 1, 2$, ottenere il valore della derivata della funzione di errore rispetto al coefficiente, calcolata per il valore attuale dei coefficienti $\boldsymbol{\beta}$, in quanto

$$\frac{\partial E_i}{\partial \beta_{jl}^{(k)}} = \delta_j^{(k)} z_l^{(k-1)}$$

   e quindi,

$$\frac{\partial E_i}{\partial \beta_{jl}^{(2)}} = z_l (y_j - t_j)$$

$$\frac{\partial E_i}{\partial \beta_{jl}^{(1)}} = x_l z_j (1 - z_j) \sum_{i=1}^{K} \beta_{ij}^{(2)} (y_j - t_j)$$

Iterando i passi precedenti su tutti gli elementi del training set, e ricordando che $E(\boldsymbol{\beta}) = \sum_{i=1}^{n} E_i(\boldsymbol{\beta})$ e che quindi possiamo esprimere

$$\frac{\partial E}{\partial \beta_{jl}^{(k)}} = \sum_{i=1}^{n} \frac{\partial E_i}{\partial \beta_{jl}^{(k)}}$$

concludiamo che l'iterazione dei passi precedenti su tutti gli elementi del training set ci fornisce la valutazione del gradiente $\nabla_{\boldsymbol{\beta}}(E(\boldsymbol{\beta}))$ per il valore attuale di $\boldsymbol{\beta}$.

A questo punto, a partire dalla conoscenza del gradiente, possiamo effettuare un singolo passo di discesa del gradiente

$$\boldsymbol{\beta}^{(i+1)} = \boldsymbol{\beta}^{(i)} - \eta \nabla_{\boldsymbol{\beta}}(E(\boldsymbol{\beta}))|_{\boldsymbol{\beta}^{(i)}}$$

Si può cercare di rendere più efficiente il procedimento precedente prevedendo che, ad ogni iterazione della discesa del gradiente, l'algoritmo di back-propagation operi con riferimento ad un solo elemento del training set (estratto casualmente, o ciclando tra i diversi elementi).