

Reti neurali applicate al Data mining

Andrea Burattin

1 ottobre 2008

Sommario

In questo documento si tratterà la tecnologia delle Reti Neurali, applicata al Data mining: dopo una breve introduzione sul funzionamento delle Reti Neurali e sulle principali tecniche di Data Mining, verranno approfondite queste due tematiche e, infine, verranno presentati un paio di casi di studio che illustrano l'applicazione della tecnologia delle Reti Neurali a problemi reali.

Indice

1	Introduzione	2
1.1	Le Reti Neurali	2
1.2	Reti Neurali e Data mining	3
2	Reti Neurali	3
2.1	Cenni storici	3
2.2	Cosa sono le Reti Neurali	4
2.3	Riferimenti biologici	5
2.4	Neuroni artificiali	5
2.5	Funzionamento delle Reti Neurali	7
	2.5.1 Apprendimento per un singolo neurone	8
	2.5.2 Apprendimento per reti di neuroni	8
2.6	Tipologie di Reti Neurali	12
	2.6.1 Reti Feed-forward	12
	2.6.2 Reti ricorrenti	13
	2.6.3 Self-organizing map	14
2.7	Pro e contro dell'uso delle Reti Neurali	17

3	Data Mining con Reti Neurali	17
3.1	Perché usare le Reti Neurali	18
3.2	Applicazioni pratiche	18
A	Caso di studio (rete feed–forward): classificazione di cifre manoscritte tramite Rete Neurale	19
A.1	Introduzione al problema	19
A.2	Soluzioni implementate	21
A.3	Rete unica	23
A.4	Reti multiple	24
A.5	Risultati con rete unica	24
A.6	Risultati con reti multiple	25
A.7	Conclusioni	25
B	Caso di studio (SOM): “Islands of Music”	26
B.1	Realizzazione	26
B.2	Risultati ed analisi	27
	Riferimenti bibliografici	28

1 Introduzione

In questa prima sezione, verranno illustrati i principali concetti e termini legati alle Reti Neurali ed al Data mining.

1.1 Le Reti Neurali

Le Reti Neurali Artificiali, più spesso identificate solo come Reti Neurali, sono un modello matematico realizzato come “copia” concettuale di una rete di neuroni umani. Una esatta definizione di questa tecnologia non è nota, mentre è assolutamente accettato che una Rete Neurale è composta da semplicissimi elementi “singoli”, detti neuroni, che, una volta interconnessi fra loro, permettono di ottenere dei comportamenti molto complessi.

Una rete neurale sufficientemente allenata è in grado di classificare in maniera plausibile degli input non noti fino a quel momento e, per questo motivo, si utilizza il termine *apprendimento* associato a questa tecnologia. Una nota semantica: il termine *apprendimento* è usato con la stessa accezione

con cui si usa il termine *volare* sia per un uccello che per un aereo (sebbene i principi ed il funzionamento siano differenti)¹.

1.2 Reti Neurali e Data mining

L'applicazione principale delle reti neurali, applicate al data mining, è quella di effettuare clustering e classificazione dei dati in base ai parametri a disposizione. Sono inoltre spesso utilizzate come strumento per effettuare previsioni di vendita ed analisi sui dati.

2 Reti Neurali

Le reti neurali sono state studiate in maniera estremamente estesa, con grandi contributi provenienti da settori molto diversi fra loro, quali la Biologia (neurofisiologia), la Matematica (ottimizzazione, approssimazione), l'Informatica (intelligenza artificiale), la Statistica (regressione, **classificazione**), l'Economia (studio di serie temporali), la Psicologia (apprendimento, scienze cognitive), ...

È interessante osservare che le principali motivazioni che spingono lo studio delle Reti Neurali sono:

1. la **riproduzione del cervello umano**: ovvero il tentativo di modellare in maniera affidabile alcuni comportamenti del cervello umano, tramite la riproduzione dei fenomeni neurofisiologici principali;
2. l'estrazione dei **principi fondamentali** del calcolo usati nel cervello umano: quindi senza dare importanza alla modellazione fisica del cervello, ma concentrandosi solo nelle astrazioni e semplificazioni al fine di ottenere un sistema in grado di riprodurre alcune funzionalità del cervello umano.

2.1 Cenni storici

I primi a parlare di neuroni artificiali furono Warren McCulloch e Walter Pitts, in un loro articolo del 1943 (*"A logical calculus of the ideas immanent in nervous activity"*). In questo documento venivano presentati i neuroni artificiali come dei modelli di neuroni biologici, e le reti erano caratterizzate da un neurone in entrata ed un neurone in uscita, inoltre tutti i dati dovevano essere binari. Nell'articolo, veniva esposta una tecnica che prevedeva di

¹A questo proposito si potrebbe discutere sulle differenze fra AI forte e debole, ma si rimanda al capitolo 26 di [8] per una ampia trattazione dell'argomento.

collegare fra loro molte di queste sequenze e si riteneva possibile che questa struttura fosse in grado di calcolare delle funzioni booleane: tale affermazione derivava dall'osservazione di come si potesse implementare una rete che realizzasse la funzione AND ed una per la OR.

Nel 1958, Frank Rosenblatt, al Cornell Aeronautical Laboratory, propose l'idea del *perceptron*: una entità caratterizzata da uno strato di ingresso ed uno di uscita, la cui regola di apprendimento si basa sulla minimizzazione dell'errore. Il sistema confronta l'uscita effettiva della rete rispetto a quella prevista ed aggiorna i pesi delle varie connessioni di conseguenza. È con questa eccezionale scoperta che si inizia a intuire la potenzialità di un sistema che fosse in grado di adattarsi alla realtà e quindi di **apprendere**.

Alcuni anni dopo, nel 1969, Minsky e Papert pubblicarono il libro *An introduction to computational geometry*. In questo testo vennero dimostrati i limiti delle reti proposte da Rosenblatt, che sono in grado di calcolare solo funzioni linearmente separabili². Il fatto che, date le limitazioni, neanche una funzione semplice come lo XOR fosse realizzabile causò un forte rallentamento nelle ricerche in questo campo.

Passati alcuni anni, si ebbe una nuova impennata nelle ricerche dopo che Hopfield, nel 1982, rilanciò lo studio cercando di confutare le tesi di Minsky e Papert. Nel 1986, David E. Rumelhart, G. Hinton e R. J. Williams diedero vita al famoso algoritmo *Backpropagation* che prevede la correzione dei pesi delle connessioni fra i nodi, in base alla correttezza o meno del risultato della rete.

Si iniziarono ad introdurre dei livelli intermedi (detti anche nascosti) e, tramite la discesa di gradiente, è possibile addestrare la rete affinché trovi il minimo locale di una funzione appartenente ad un particolare spazio. In questa maniera, la rete sarà in grado di calcolare delle risposte plausibili anche per dei in input che la rete non ha mai visto. Con questa tecnica è stato possibile oltrepassare la limitazione delle funzioni linearmente separabili, potendo finalmente apprendere anche lo XOR, ottenendo un forte rilancio di questa tipologia di reti.

2.2 Cosa sono le Reti Neurali

Come detto, le Reti Neurali Artificiali, traggono ispirazione dalle Reti Neurali presenti nel cervello umano. Quest'ultimo è costituito, in media da 10^{10} neuroni, che sono connessi in maniera estremamente fitta fra loro (circa 10^4 connessioni per ogni neurone) ed il cui tempo di risposta medio (per un sin-

²Una funzione si dice *linearmente separabile* quando esiste un iperpiano in gradi di separare gli esempi positivi da quelli negativi.

golo neurone) si aggira attorno a 0,001 secondi. Quest’ultima affermazione, unita al fatto che un essere umano — in media — impiega 0,1 secondi per riconoscere il contenuto di una scena, implica l’uso estremamente pesante, da parte del cervello umano, del **calcolo parallelo**: secondo quanto detto non potrebbero essere fatti più di

$$\frac{0,1}{0,001} = 100$$

calcoli in serie (valore che da solo non consente il riconoscimento di una scena).

Come già detto in precedenza, una Rete Neurale Artificiale è costituita da un insieme di Neuroni Artificiali interconnessi fra loro. L’elemento di interesse per questo tipo di tecnologia è la capacità di *apprendimento*: dati un task da risolvere, un insieme di funzioni F ed insieme di osservazioni, è possibile trovare una funzione $f^* \in F$ che risolva il task assegnato, tale che, data una funzione costo $C : F \rightarrow \mathbb{R}$,

$$C(f^*) \leq C(f) \quad \forall f \in F$$

2.3 Riferimenti biologici

Le reti neurali traggono ispirazione dalle strutture biologiche che costituiscono il sistema nervoso per molti esseri viventi. Il cervello umano, ad esempio, è una delle più complesse strutture che consentono l’apprendimento: esso è costituito da un numero estremamente elevato di componenti elementari — i neuroni — interconnessi fra loro da una fittissima rete di collegamenti.

Un neurone biologico, schematizzato in **Figura 1**, è caratterizzato da tre componenti principali: il *dendrite*, l’*assone* ed il *soma*. Il dendrite “raccolge” i segnali in ingresso, provenienti da altri neuroni, i quali arrivano attraverso le sinapsi. Questo trasferisce il segnale al soma, detto anche *corpo cellulare* che si occupa di eseguire una “somma pesata” dei segnali in ingresso e, nel caso in cui questa superi un valore soglia prestabilito, allora si attiva l’uscita. Il segnale di output è trasferito tramite l’assone, che ha il compito di inviare il segnale, tramite le sue migliaia di diramazioni, ad una serie di altri neuroni.

2.4 Neuroni artificiali

Un neurone artificiale, è una astrazione matematica che trae ispirazione dai neuroni biologici: può ricevere uno o più input (che rappresentano i vari dendriti in ingresso), calcola la somma pesata dei vari valori e ne restituisce il risultato.

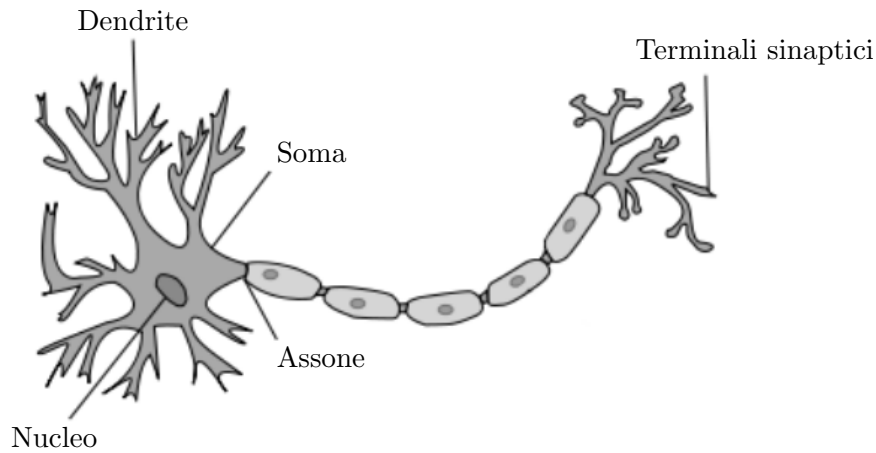


Figura 1: Struttura di un neurone biologico

Esistono varie tipologie di neuroni artificiali, i più diffusi sono quelli con *hard-threshold* e quelli *sigmoidali*. I primi consentono di implementare semplici funzioni booleane che siano linearmente separabili (quali ad esempio AND e OR). Per realizzare funzioni più complesse è necessario usare più neuroni e collegarli fra loro. Il funzionamento di questa tipologia di neuroni si basa, in prima istanza sulla somma — pesata sui pesi — dei vari input e, se questa supera un valore soglia, viene ritornato un valore piuttosto di un altro. In [Figura 2](#) è riportata la schematizzazione di un neurone di questa tipologia.

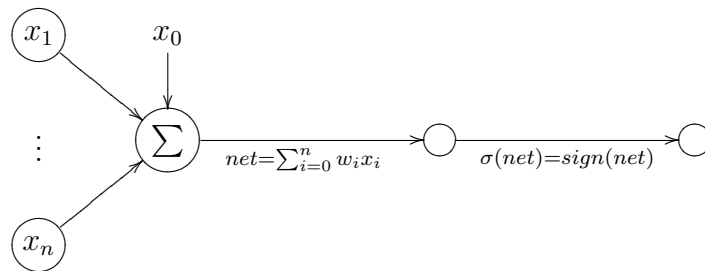


Figura 2: Struttura di un neurone artificiale con *hard-threshold*

I neuroni sigmoidali consentono di rappresentare funzioni (fortemente) non lineari. Come nel caso dell'altra tipologia di neuroni, come prima operazione si calcola la combinazione lineare degli input e, successivamente, si applica un limite al risultato. In questo caso, tuttavia, il limite non è più determinato da un valore fisso, ma è una funzione continua. Più precisamente,

L'output o , del neurone, è dato da:

$$o = \sigma(\vec{w} \cdot \vec{x})$$

dove

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

La funzione σ è detta funzione sigmoideale, ed il suo output, compreso fra 0 e 1 cresce con andamento monotono. Questa funzione è facilmente derivabile e tale proprietà la rende particolarmente adatta all'uso nel contesto dell'algoritmo che prevede la discesa di gradiente. La derivata prima della funzione σ risulta essere:

$$\sigma'(x) = \frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$$

2.5 Funzionamento delle Reti Neurali

L'apprendimento, nel campo delle reti neurali, generalmente si suddivide in tre grosse tipologie: quello supervisionato, quello non supervisionato e quello con rinforzo.

L'**apprendimento supervisionato** prevede di avere una sequenza di valori in input ed una di output, definiti tramite una funzione f che, di conseguenza, sarà una mappa da tutti i dati in ingresso a tutte le uscite. Si assume quindi che, se si fornisce un insieme sufficientemente grande di elementi "campione" sarà possibile trovare una funzione f' che approssimi correttamente f . Ciò che in effetti succede è che la rete tenta di inferire il "comportamento" di f , analizzandone gli ingressi e le relative uscite. Se tale apprendimento viene effettuato correttamente, la rete neurale sarà in grado di analizzare e restituire risultati corretti anche per valori che non ha mai analizzato in precedenza. L'algoritmo più utilizzato per questa tipologia di apprendimento è il Backpropagation.

L'**apprendimento non supervisionato**, a differenza della tipologia precedente, assume di avere solamente dati in ingresso che cercherà di raggruppare sino a formare insiemi consistenti fra loro. Gli algoritmi che appartengono a questa classe non sempre funzionano bene: molto dipende dalla tipologia dei dati in ingresso. Se vengono forniti dei dati che sono difficilmente ordinabili (e quindi confrontabili) sarà difficile che l'algoritmo trovi un risultato soddisfacente.

Nell'**apprendimento con rinforzo** molta importanza è data alla ricompensa derivante dall'aver intrapreso una azione anziché un'altra: ciascuna operazione è associata ad un valore "premio" (o "punizione"). Lo scopo finale dell'algoritmo è quello di massimizzare il premio ricevuto, e quindi effettuare la serie di azioni che inducono al più alto guadagno.

2.5.1 Apprendimento per un singolo neurone

Come già detto in precedenza, un singolo neurone è in grado di apprendere iperpiani, ovvero funzioni linearmente separabili. Queste, sono quelle contenuto in questo spazio:

$$\mathcal{H} = \{f_{(\vec{w},b)}(\vec{y}) \mid f_{(\vec{w},b)}(\vec{y}) = \text{sign}(\vec{y} \cdot \vec{w} + b), \vec{w}, \vec{y} \in \mathbb{R}^n, b \in \mathbb{R}\}$$

dove, \vec{w} rappresenta l'insieme dei pesi, \vec{y} è l'insieme degli input del neurone e b rappresenta il valore soglia.

Con questa definizione, è possibile descrivere un algoritmo per l'apprendimento di un perceptron, esso è riportato nell'[Algoritmo 1](#).

Algoritmo 1 Algoritmo di apprendimento per un singolo neurone

Input: Insieme di apprendimento $Tr = \{(\vec{x}, t)\}$, con $t \in \{-1, +1\}$

```
1 Inizializza  $\vec{w}$  al vettore nullo
2 loop
3     Seleziona a caso un esempio  $(\vec{x}, t)$ 
4     if  $out = \text{sign}(\vec{w} \cdot \vec{x}) \neq t$  then
5          $\vec{w} \leftarrow \vec{w} + (t - out)\vec{x}$ 
6     end if
7 end loop
```

Per evitare che il vettore dei pesi subisca variazioni troppo brusche, ogni volta che viene modificato (e quindi che un singolo esempio vada ad alterare significativamente il comportamento del neurone), si modifica l'apprendimento in questa maniera (riga 5 dell'[Algoritmo 1](#)):

$$\vec{w} \leftarrow \vec{w} + \eta(t - out)\vec{x}$$

con $\eta > 0$ e, preferibilmente, $\eta < 1$, detta "coefficiente di apprendimento". In questa maniera si evita che esempi classificati correttamente diventino errati solo a causa di un esempio.

Se il training set Tr è linearmente separabile, si dimostra che l'algoritmo termina in un numero finito di passi, altrimenti \vec{w} cicla in un insieme di valori che non sono ottimali e che, di conseguenza, non commettono il minimo numero di errori possibile.

2.5.2 Apprendimento per reti di neuroni

Se si ha la necessità di apprendere funzioni non linearmente separabili o, più in generale, funzioni particolarmente complesse, è necessario implementare una rete di neuroni. Un esempio di una rete neurale è riportato in [Figura 3](#).



Figura 3: Una rete neurale artificiale con un layer nascosto

Con questa tipologia di reti, è anche possibile apprendere qualsiasi funzione booleana, tuttavia il problema sta nel decidere — nel caso di un errore — quali siano i neuroni che devono subire alterazioni nel loro peso, dato che non è possibile stabilire qual'è il neurone che incide nell'errore del risultato.

Una soluzione, che consente di effettuare l'apprendimento per reti di neuroni, è quella di rendere derivabile il singolo neurone e quindi sfruttare la *discesa di gradiente* per correggere i vari pesi. Questa tecnica consente di ottenere un minimo locale, per una funzione matematica.

Per semplicità si può considerare il caso di neuroni senza hard-threshold (quindi che non necessitano di un valore minimo per essere attivati), ovvero la cui funzione è:

$$out(\vec{x}) = \sum_{i=0}^n w_i \cdot x_i = \vec{w} \cdot \vec{x}$$

È necessario, ora, definire la funzione che si vuole minimizzare. Questa funzione è la misura dell'errore, connesso ad un particolare vettore dei pesi:

$$E[\vec{w}] = \frac{1}{2N_{Tr}} \sum_{(\vec{x}^{(i)}, t^{(i)}) \in Tr} (t^{(i)} - out(\vec{x}^{(i)}))^2$$

dove N_{Tr} è la cardinalità dell'insieme di apprendimento. Questa funzione misura la metà dello scarto quadratico medio del valore target, rispetto al valore in output dalla rete neurale. Cercando di minimizzare il valore di questa funzione, siamo in grado di trovare una configurazione ottimale per i valori dei pesi della rete. Questa idea è applicata seguendo la tecnica della “discesa del gradiente”.

L'**Algoritmo 2** descrive una implementazione della tecnica di discesa di gradiente appena descritta. Questo algoritmo calcola un vettore dei pesi

Algoritmo 2 Algoritmo di apprendimento per rete neurale tramite discesa di gradiente

Input: Insieme di apprendimento Tr , coefficiente di apprendimento η

```
1 Inizializza  $\vec{w}_i$  a valori piccoli a piacere
2 while condizione di terminazione do
3      $\Delta w_i \leftarrow 0$ 
4     for all  $(\vec{x}, t) \in Tr$  do
5         Presenta  $x$  al neurone e calcola  $out$ 
6         for all  $w_i$  do
7              $\Delta w_i \leftarrow \Delta w_i + \eta(t - out)x_i$ 
8         end for
9     end for
10    for all  $w_i$  do
11         $w_i \leftarrow w_i + \Delta w_i$ 
12    end for
13 end while
```

che minimizza E iniziando con una serie di valori casuali. Questo viene modificato ad ogni passo di derivazione: i pesi sono alterati nella direzione opposta rispetto al gradiente (ovvero seguendo la via più ripida possibile) nella superficie dell'errore, fino al raggiungimento di un minimo locale.

Un esempio di superficie descritta dalla funzione errore è riportata in **Figura 4**: in questo caso, la rete neurale è costituita da due neuroni solamente. Gli assi w_0 e w_1 rappresentano i possibili valori per i due pesi, ovvero rappresentano l'intero spazio delle ipotesi, mentre l'asse verticale sta ad indicare il valore della funzione errore. La freccia indica la direzione verso cui è necessario far tendere i valori (discesa di gradiente), al fine di arrivare ad un minimo locale.

Nel caso si stia considerando una rete neurale costituita da più di un layer, l'algoritmo per la "discesa di gradiente semplice" non è sufficiente, si deve utilizzare l'algoritmo **Backpropagation**. Questo algoritmo, sfruttando sempre la discesa di gradiente, cerca di minimizzare lo scarto quadratico fra i valori di output della rete ed i valori target (valori conosciuti). A questo punto, poiché si sta lavorando con reti costituite da più livelli, è necessario modificare la funzione per l'errore affinché tenga conto di questo fatto (ovvero di tutti i neuroni di output):

$$E[\vec{w}] = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$$

dove $output$ è l'insieme di neuroni di output, associato all'esempio d del

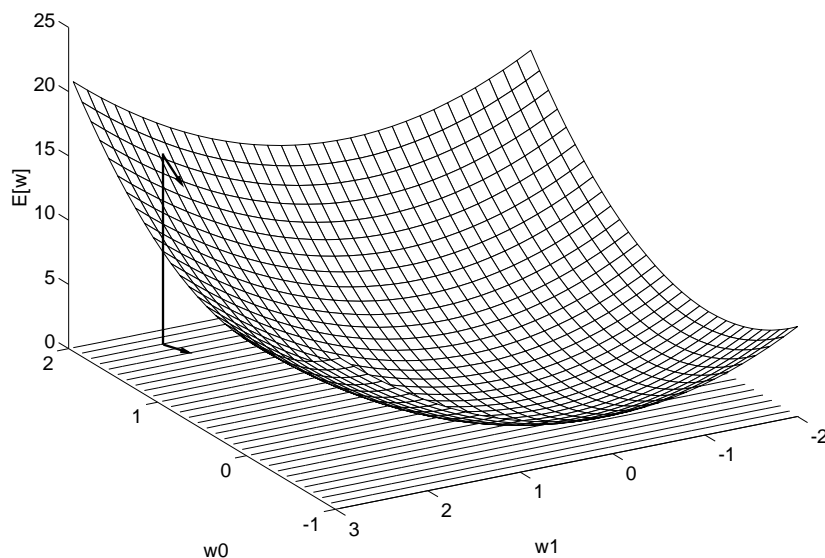


Figura 4: Grafico dell'errore. La freccia sta ad indicare la direzione con cui verranno aggiornati i pesi (ovvero la più ripida possibile)

training set.

L'obiettivo finale di Backpropagation è quello di trovare una soluzione fra tutti i valori possibili per tutti i neuroni della rete, come detto, anche qui si utilizza la tecnica di discesa di gradiente. La differenza principale, rispetto al caso di rete con livello singolo è che ora è possibile che la funzione errore abbia più di un minimo locale. L'algoritmo, quindi può garantire solo di trovare una soluzione ottima localmente, ma non l'ottimo globale. Si è comunque verificato sperimentalmente (anche nel caso di studio, riportato in appendice) che l'algoritmo funziona molto bene per molti problemi pratici.

L'**Algoritmo 3** è una trascrizione dello pseudocodice di Backpropagation.

Questo algoritmo aggiorna i pesi correggendo un solo errore per volta. Esistono tuttavia due approcci diversi alla discesa: quello "batch" e quello "stocastico". Il primo aggiorna i pesi dopo che è stato valutato tutto il training set, il secondo (detto anche "incrementale") effettua le correzioni ad ogni esempio. In entrambi i casi c'è l'uso del passo di discesa η , e la sua valorizzazione è di estrema importanza: la discesa stocastica può approssimare quella batch solo con valori di η sufficientemente piccoli. La scelta di questo valore è estremamente importante anche nello stabilire il rapporto fra velocità di apprendimento ed accettazione del rischio di fermare l'apprendimento su dei minimi locali.

Algoritmo 3 Algoritmo Backpropagation

Input: Insieme di apprendimento Tr , coefficiente di apprendimento η , la topologia della rete

```
1 Inizializza  $\vec{w}_i$  a valori piccoli a piacere
2 while condizione di terminazione do
3   for all  $(\vec{x}, \vec{t}) \in Tr$  do
4     Presenta  $x$  al neurone e calcola  $out$ 
5     for all unità di output  $k$  do
6        $\delta_k = o_k(1 - o_k)(t_k - o_k)$ 
7     end for
8     for all unità nascosta  $j$  do
9        $\delta_j = o_j(1 - o_j) \sum_{k \in output} w_{k,j} \delta_k$ 
10    end for
11     $w_{s,q} = w_{s,q} + \eta \Delta w_{s,q}$ 
```

$$\text{dove } \Delta w_{s,q} = \begin{cases} \delta_s x_q & \text{se } s \in \text{hidden} \\ \delta_s y_q & \text{se } s \in \text{out} \end{cases}$$

```
12   end for
13 end while
```

2.6 Tipologie di Reti Neurali

Esistono molte tipologie di Reti Neurali che si differenziano, essenzialmente, per la loro struttura topologica. Ciascuna di queste reti ha particolari proprietà, ed ogniuna si adatta meglio alla risoluzione di task specifici.

2.6.1 Reti Feed-forward

Le Reti Feed-forward sono state le prime ad essere realizzate e sono fra le più semplici. La caratteristica principale di queste reti è che i neuroni sono separati in “layer”, ed i neuroni di ciascun layer sono collegati a tutti i neuroni del layer successivo, partendo dal livello con i neuroni in input fino ai neuroni di output (che non sono collegati a nulla). Volendo vedere la rete come un grafo, abbiamo un grafo orientato ed aciclico. La [Figura 3](#) rappresenta una rete neurale di tipo feed-forward.

Esiste un teorema che stabilisce la capacità espressiva delle reti feed-forward: esso afferma che questa tipologia di reti neurali consentono di approssimare qualsiasi funzione continua:

Teorema 2.1 *Sia $\varphi(\cdot)$ una funzione continua, monotona, crescente, limitata e non costante; si indichi con I_n l'ipercubo n -dimensionale $[0, 1]^n$ e sia*

$C(I_N)$ lo spazio delle funzioni continue definite su esso. Data una qualunque funzione $f \in C(I_n)$ e $\varepsilon > 0$, allora esiste un intero M e insiemi di costanti reali α_i , θ_i e w_{ij} , dove $i = 1, \dots, M$ e $j = 1, \dots, n$ tale che $f(\cdot)$ possa essere approssimata da

$$F(x_1, \dots, x_n) = \sum_{i=1}^M \alpha_i \varphi \left(\sum_{j=1}^n w_{ij} x_j - \theta_i \right) \quad (2)$$

in modo tale che

$$|F(x_1, \dots, x_n) - f(x_1, \dots, x_n)| < \varepsilon$$

per tutti i punti $[x_1, \dots, x_n] \in I_n$.

Si noti che, se si utilizza una rete neurale con funzione sigmoideale (ovvero quelle descritte dall'equazione 1) possiamo soddisfare le richieste dal teorema per $\varphi(\cdot)$.

L'equazione 2, può essere pensata come l'output di una rete feed-forward in cui:

- la rete ha n nodi in input, un solo strato nascosto con M unità e gli input sono x_1, \dots, x_n ;
- l' i -esima unità ha i pesi: $w_{i,1}, \dots, w_{i,n}$ e soglia θ_i ;
- l'output della rete è una combinazione lineare degli output delle reti nascoste, in cui i coefficienti sono dati da $\alpha_1, \dots, \alpha_M$.

In questa maniera, data una “soglia di tolleranza” ϵ , una rete feed-forward con un solo strato nascosto può approssimare qualsiasi funzione in $C(I_n)$.

Resta un problema tutt'altro che banale, collegato alla definizione delle reti (che non riguarda solo le reti di questa tipologia): questo teorema non fornisce alcuna indicazione su come trovare il numero M di unità nascoste (necessarie per approssimare la funzione target), ma si limita a segnalarne l'esistenza.

2.6.2 Reti ricorrenti

Una rete ricorrente è una rete che può essere vista come un grafo ciclico, a differenza di quelle viste fino ad ora, che erano *acicliche*. Questa tipologia di reti calcola l'output delle sue componenti all'istate t che sarà usato come input per la altre unità della rete a $t + 1$. Proseguiamo la descrizione con un esempio pratico: si deve costruire una rete in grado di predire gli incassi del

giorno successivo $y(t+1)$, di una qualche attività, a partire dai dati economici del giorno corrente $x(t)$.

Per il problema appena indicato, si potrebbe pensare di utilizzare la rete presentata al punto (a) della **Figura 5**, ovvero che la predizione di $y(t+1)$ sia il risultato del calcolo fatto da una rete feed-forward con, in input, $x(t)$. Il problema di questo tipo di reti sorge se la finestra temporale che si vuole poter consultare è ampia. Ad esempio, le previsioni potrebbero essere influenzate non solo da $x(t)$, ma anche da $x(t-1)$ e non è possibile prevedere questa situazione con questa tipologia di reti (si potrebbe aggiungere un nodo con il valore di $x(t-1)$ ma non si sa se si necessita anche di $x(t-2)$, etc.).

Al punto (b) della **Figura 5** si presenta una soluzione completamente diversa rispetto a quella proposta pocanzi: si utilizza infatti una rete ricorrente. In questo caso si è aggiunto una nuova unità b al layer nascosto, ed una nuova unità di input $c(t)$ che prende il valore che b aveva nell'istante $t-1$. In questa maniera, grazie al nodo b , è possibile descrivere delle relazioni storiche fra gli input della rete e, essendo questo nodo parametrico sia in x che in t esso, in qualche maniera, può descrivere l'andamento della rete per tutto il suo periodo di esistenza. Naturalmente, si sarebbe potuto aggiungere un numero superiore di unità nascoste che descrivono la ricorrenza, ma non si è voluto complicare troppo questo caso, del tutto esemplificativo.

Per effettuare il training di una rete neurale ricorsiva sono state proposte varie tecniche, ma la più semplice ed utilizzata è una variante dell'**algoritmo Backpropagation**. Per capire come possa funzionare tale algoritmo si osservi la parte (c) della **Figura 5**, nella quale è stato “fatto un passo” nell'esplicazione della ricorsione della rete. A questo punto la “nuova” rete risulta di più semplice comprensione, dato che non contiene alcun ciclo al suo interno, è quindi possibile eseguire normalmente il Backpropagation. Chiaramente, non si vuole effettuare il training sulla rete “splittata” ma su quella originale, per cui i vari pesi che l'algoritmo modificherà dovranno essere aggiornati tenendo conto di questo fatto: si modifica più volte lo stesso peso, all'interno di uno stesso aggiornamento. Tale operazione anche se fattibile, risulta abbastanza complicata.

Le reti neurali ricorrenti, in definitiva, risultano difficili da essere addestrate e, in generale, da essere gestite, tuttavia sono da ritenersi estremamente importanti per l'elevato potere espressivo di cui sono dotate.

2.6.3 Self-organizing map

Le self-organizing map sono una tipologia di reti neurali che ha il compito di consentire la visualizzazione dei dati del training set in una maniera particolarmente familiare agli utenti, proiettando questi dati in uno spazio a bassa

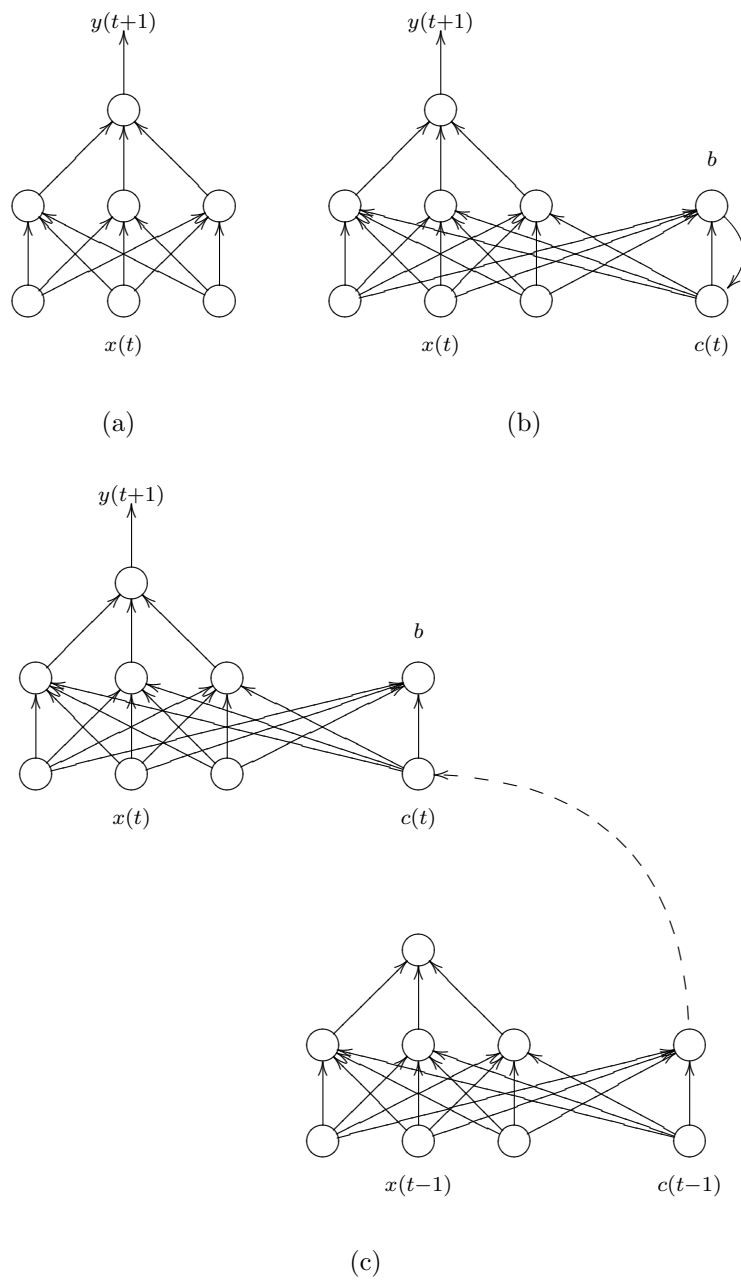


Figura 5: Al punto (a) è presente una normale rete feed-forward; al punto (b) è stato aggiunto un nuovo neurone in input che la rende ricorrente e, al punto (c), c'è la stessa rete ricorrente (b) con uno step ricorsivo esplicitato: si può pensare al nodo b come ad una variabile che accumula informazione storiche sui valori che la rete assume nel tempo.

dimensionalità (generalmente uno spazio bidimensionale). Questo modello fu originariamente descritto come una rete neurale dal professore finlandese Teuvo Kohonen e, per questo motivo, spesso sono dette anche **mappe di Kohonen**.

Questa tipologia di reti è simile alle feed-forward: una SOM è infatti una rete a singolo strato in cui ciascuno dei neuroni in ingresso sono collegato a tutti i neuroni di uscita. Dato che questo strumento è spesso utilizzato per ridurre la dimensionalità del vettore in input abbiamo che, generalmente, il numero di neuroni in ingresso è molto maggiore al numero di neuroni in uscita.

Nel caso di una rete con un numero abbastanza piccolo di nodi, il comportamento della rete è paragonabile a quello dell'algoritmo K-means³, se invece si aumenta il numero di nodi si ottiene una organizzazione più topologica.

L'addestramento delle SOM è di tipo non supervisionato ovvero senza sapere a priori quali dati saranno estratti per l'analisi. L'apprendimento viene fatto con i seguenti passi:

1. si seleziona un vettore in input ξ dal training set, secondo una distribuzione di probabilità;
2. si calcola la distanza euclidea fra ξ rispetto a tutti i neuroni, e si trova il neurone k (detto BMU, Best Matching Unit) per il quale è minima la distanza con il vettore in input;
3. si aggiorna il vettore dei pesi nella seguente maniera:

$$w_i = w_i + \eta\phi(i, k)(\xi - w_i) \quad i = \{1, \dots, m\}$$

dove w è il vettore dei pesi, m è il numero di unità, η è un coefficiente di apprendimento e ϕ è una funzione di vicinanza e dipende dalla distanza fra il neurone ξ ed il BMU⁴;

4. se il massimo numero di iterazioni è stato raggiunto ci si ferma, altrimenti si continua dal passo 1.

Tramite questa procedura, i neuroni più vicini all'BMU sono ravvicinati e, poiché la funzione di vicinanza deve diminuire nel tempo, gli effetti maggiori (ovvero i maggiori avvicinamenti) si hanno nelle fasi iniziali dell'algoritmo,

³K-means è un algoritmo di clusterizzazione non gerarchica che utilizza la distanza fra le variabili come criterio di somiglianza.

⁴Un semplice esempio di definizione della funzione di vicinanza è $\phi(i, k) = 1$ per tutte le unità i ad una distanza massima di r , 0 per tutte le altre. Spesso è usata una funzione gaussiana. Questa funzione dovrebbe diminuire nel tempo.

portando un numero di correzioni maggiore all'inizio piuttosto che alla fine. Tale processo porta alla definizione di due macro-fasi: quella iniziale detta *ordering phase* nella quale il vicinato è ampio e la mappa subisce una auto-organizzazione a livello globale; successivamente, al diminuire del numero di vicini, si ha una convergenza dei neuroni ad una stima locale, detta *tuning phase*.

2.7 Pro e contro dell'uso delle Reti Neurali

Le reti neurali sono utilizzate quando si vogliono realizzare delle funzioni delle quali abbiamo solo alcuni dati e per i quali non appare chiaro il comportamento (ovvero quando il legame fra i dati è particolarmente complesso e risultano impraticabili altri approcci più tradizionali).

Il principale vantaggio, nell'uso delle reti neurali, sta nel fatto che queste sono in grado di gestire agilmente una grandissima mole di informazioni, senza alcun tipo di problema anche nel distribuire il calcolo (come già detto, infatti, il cervello umano sfrutta in maniera estremamente pesante questa tecnica). Un ulteriore vantaggio è che, dato che questa tecnica è, in definitiva, uno strumento statistico, si possono avere dati contenenti rumore, ed il risultato non dovrebbe risentirne particolarmente. L'ultimo, e forse più interessante, vantaggio sta nel non dover avere alcuna informazione sulla forma della funzione target: è compito della rete estrarre tale funzione e permetterci di utilizzarla per fare nuove elaborazioni ed inferenze.

In parallelo ai numerosi vantaggi, ci sono alcune note dolenti: primo fra tutti il "black-box problem" ovvero che la struttura della soluzione di una rete non è comprensibile ad un essere umano (salvo casi eccezionalmente semplici): questo ci rende difficile capire la funzione target ed eventualmente modificarla, senza tuttavia impedirci di rieseguire la funzione per valori non ancora sottoposti alla rete. Questa caratteristica rende impossibile una analisi algoritmica della funzione target – ovvero che spieghi il perché di ogni singolo passo – e di come si generino i valori in output. L'altro problema fondamentale, e forse ancora più importante, sta dietro alla realizzazione di queste reti: non esistono teoremi che aiutino a modellare topologicamente una rete ottimale a partire da un problema dato e, di conseguenza, la buona riuscita di tale progetto dipende moltissimo dall'esperienza del creatore.

3 Data Mining con Reti Neurali

Le reti neurali, all'interno del data mining sono, essenzialmente, utilizzate per le operazioni di clustering, ovvero per la suddivisione dei dati in gruppi

molto omogenei al loro interno, ed eterogenei fra loro.

3.1 Perché usare le Reti Neurali

Per le tematiche di data mining ho visto un uso delle reti neurali particolarmente diffuso nelle tematiche legate alla classificazione dei dati. Rispetto ad altri algoritmi che si occupano di fare solo clustering (si pensi, ad esempio, a k-means) le reti neurali hanno un grosso vantaggio: è possibile sottoporre alla rete un esempio del quale non si hanno ancora tutte le informazioni e la rete, sulla base dei dati a disposizione, lo potrà collocare in un cluster piuttosto che in un altro. Sempre prendendo in considerazione il confronto con k-means, non c'è alcun bisogno di identificare e “posizionare” i seed dei cluster inizialmente: da tale attività potrebbe dipendere la buona riuscita o meno della classificazione.

Come già accennato in precedenza, le reti neurali non hanno alcun problema nel gestire grosse moli di dati anzi, spesso, più si hanno dati a disposizione, meglio viene fatto l'apprendimento, evitando problemi legati all'overfitting della rete.

Resta da sottolineare che non è mai bene utilizzare una singola tecnica di data mining, bensì risulta conveniente provare varie strade e varie strumenti e selezionare solo alla fine quella più redditizia.

3.2 Applicazioni pratiche

Dato che, come spiegato più volte in questo documento, le reti neurali necessitano solo di un insieme di osservazioni per fare l'apprendimento, e quindi per realizzare la funzione target, sono di particolare interesse per tutti i settori in cui la complessità dei dati o la difficoltà nell'elaborarli è notevole.

I campi di applicazione delle reti neurali sono estremamente numerosi, e vanno dal riconoscimento del parlato, alla classificazione delle immagini (si veda l'appendice) sino alla predizione di serie temporali finanziarie.

In generale, è possibile identificare tre categorie principali fra i campi d'applicazione:

1. funzioni per l'approssimazione e la regressione, utilizzate per effettuare previsioni;
2. funzioni di classificazione e di riconoscimento;
3. filtraggio di dati, clustering e compressione.

Da queste tre macro-categorie, è possibile derivare un numero pressochè illimitato di potenziali applicazioni: da sistemi per il controllo dei veicoli ai sistemi per il riconoscimento e la categorizzazione delle email (non solo spam, ma anche auto-risponditori, ...), simulazioni di giochi (scacchi, backgammon), riconoscimento di caratteri (OCR), riconoscimento di parametri biometrici (impronte digitali, volto, ...), diagnosi mediche.

A Caso di studio (rete feed-forward): classificazione di cifre manoscritte tramite Rete Neurale

In questa sezione si analizza uno studio che ha avuto come oggetto la realizzazione di un sistema per il riconoscimento di cifre manoscritte tramite reti neurali. L'implementazione è stata compiuta seguendo due strategie per la definizione della struttura del risolutore del problema: in una prima modellazione si è implementata una sola rete neurale per il riconoscimento di tutte le cifre; nella seconda si sono costruite dieci reti neurali, ciascuna in grado di riconoscere una sola cifra, i cui output venivano, in un secondo momento, dati in input ad un'ulteriore rete che dava i risultati finali.

A.1 Introduzione al problema

Il riconoscimento automatico di cifre manoscritte è un importante problema, che può essere riscontrato in molte applicazioni pratiche. Alcuni esempi di applicazioni di questa tecnica:

- sistemi di smistamento automatico della posta cartacea, basato sul riconoscimento del CAP scritto nelle buste;
- inserimento automatico degli importi delle tasse letti dai bollettini;
- riconoscimento automatico degli input per computer palmari.

Ci sono stati diversi progressi, negli ultimi anni, in questo settore, principalmente dovuti all'introduzione di nuovi algoritmi per l'apprendimento, la disponibilità di nuovi *training set* sui quali poter fare l'apprendimento e, non in ultima istanza, l'aumento della potenza di calcolo a disposizione sui computer moderni.

Sia per l'apprendimento, sia per la validazione dei dati di input è stato utilizzato un archivio di immagini molto conosciuto nel settore: “*The MNIST database of handwritten digits*”. Questo è costituito da un training set di

60000 esempi, più un test set di 10000. Un esempio di dati, cos come sono memorizzati nel file degli esempi è riportato in [Figura 6](#).

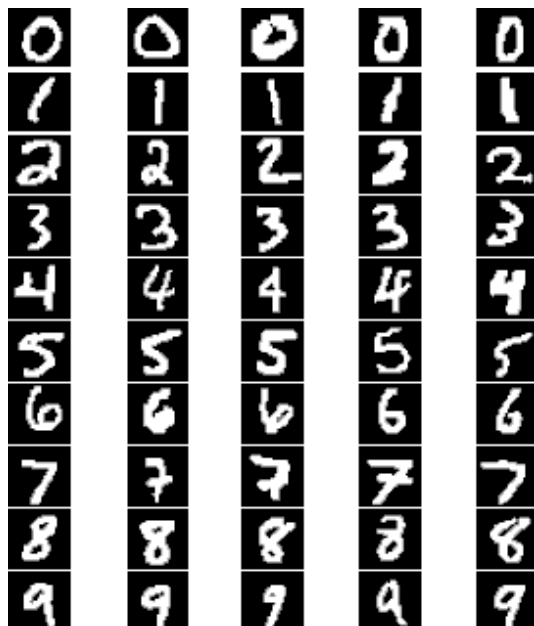


Figura 6: Esempio di dati estratti dal database MNIST

Il database originario è stato costruito dal NIST (National Institute of Standards and Technology). Le immagini di partenza erano in bianco e nero, ma successivamente, al fine di normalizzarle alla dimensione di 20×20 pixel, sono stati introdotti dei livelli di luminosità intermedi, dovuti all'effetto di anti-aliasing del filtro per il ridimensionamento. Successivamente, sono state centrate, nel centro di massa dei pixel, in un'area di 28×28 px, al fine di migliorare il processo di apprendimento.

L'intero database è memorizzato in due file: uno contenente l'insieme con tutti i dati, ed uno con tutte le etichette; i valori aspettati corrisponderanno all'immagine nella stessa posizione: l'immagine i -esima (ovvero quella che corrisponde all'insieme di 28×28 byte posti ad $i \times (28 \times 28)$ dall'inizio del file con gli esempi) avrà come valore aspettato il valore i -esimo (ovvero il valore posto ad i posizioni dall'inizio del file delle etichette). La codifica del file con le etichette, cos come quella del file con le immagini è decisamente semplice da essere utilizzata: i dati sono stati salvati come fosse un dump (ad eccezione di un breve header contenente il numero di esempi campionati e, per le immagini, il numero di righe e di colonne) in modo da facilitare al

massimo la loro lettura, una volta conosciuta la codifica. Il file delle etichette è costruito secondo la struttura illustrata in [Figura 7](#).

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000801(2049)	magic number (MSB first)
0004	32 bit integer	60000	number of items
0008	unsigned byte	??	label
0009	unsigned byte	??	label
...			
xxxx	unsigned byte	??	label

Figura 7: Struttura file delle etichette, dal database MNIST

Il file con gli esempi, invece, è strutturato secondo lo schema di [Figura 8](#).

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
...			
xxxx	unsigned byte	??	pixel

Figura 8: Struttura file con le immagini, dal database MNIST

A.2 Soluzioni implementate

Come suggerito in [\[8\]](#), si è sperimentalmente osservato che i risultati migliori si ottengono con una rete neurale costituita da un neurone in input per ciascun pixel dell'immagine da analizzare, un livello di neuroni nascosti con 300 unità e 10 neuroni in output (uno per ciascuna delle cifre in output).

Una rappresentazione grafica della rete che è stata modellata è riportata in [Figura 9](#).

Dato che le immagini sono codificate in scala di grigi, ciascun pixel è rappresentato dal livello di luminosità in quel determinato punto. L'input per il neurone è quindi facilmente realizzabile, semplicemente considerando il valore del pixel come un intero. Come già detto, successivamente, il livello

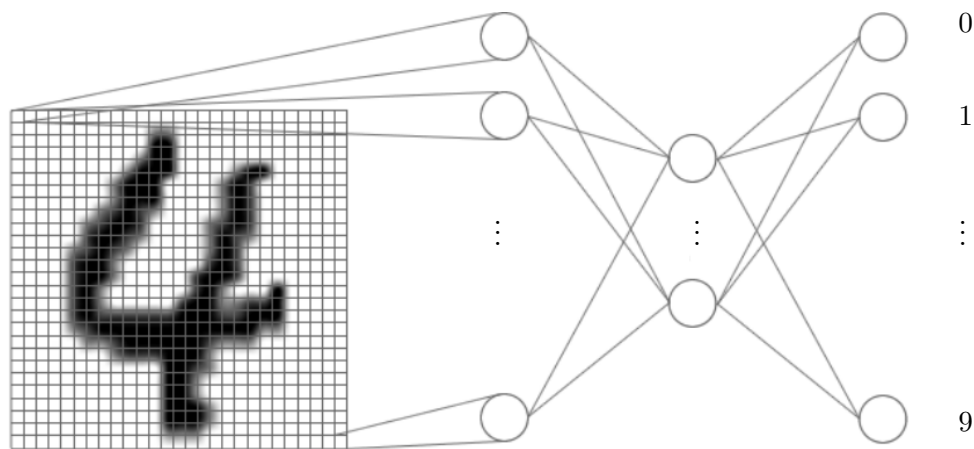


Figura 9: Struttura della rete neurale che si è implementata, per il riconoscimento delle cifre manoscritte

di input è connesso a 300 neuroni nascosti i quali, a loro volta, sono tutti collegati a 10 neuroni di output, ciascuno dei quali rappresenta la probabilità che l'input fornito alla rete rappresenti la cifra in considerazione.

Le sperimentazioni tecnologiche del problema sono state fatte utilizzando le librerie FANN (Fast Artificial Neural Networks).

Le librerie FANN sono un software free e open source per lo sviluppo di reti neurali multilayer. Sono scritte in C, e sono pensate per supportare sia reti *fully connected*, sia *sparsely connected*, inoltre sono cross-platform e adatte all'uso sia di valori interi che in virgola mobile, questa libreria, inoltre, comprende un framework per la gestione semplificata dei training set. Esistono porting per questa libreria in molti linguaggi, fra i quali: PHP, C++, .NET, Ada, Python, Delphi, Octave, Ruby, Prolog Pure Data e Mathematica. D'ora in avanti, si farà sempre riferimento alla versione originale della libreria, scritta in C. La documentazione è molto ben strutturata ed esauriente, ed è suddivisa per package. È correlata da un insieme di tutorial ed esempi che ne illustrano tutte le fasi del ciclo di vita, dall'installazione all'uso.

Gli algoritmi di apprendimento, che le librerie FANN implementano sono:

- l'algoritmo "standard" Backpropagation incrementale;
- l'algoritmo "standard" Backpropagation batch;
- l'algoritmo RPROP;
- l'algoritmo Quickprop.

A.3 Rete unica

L'intero progetto è costituito da più software, ciascuno dei quali è orientato verso una attività specifica. In totale sono stati realizzati cinque programmi:

1. **convert**: programma realizzato al fine di convertire le immagini e le etichette in un formato accettabili per la libreria FANN. Questo, in effetti, è formato da una sequenza di valori separati da uno spazio; ciascun esempio è costituito da due righe: nella prima ci sono tutti i valori per le unit in input, nel secondo ci sono i valori dei neuroni di output;
2. **train**: programma che avvia la sessione di apprendimento per la rete. È parametrizzato per vari aspetti, i più importanti dei quali consentono di decidere se continuare ad apprendere da una rete già esistente o se costruirne una nuova; il massimo numero di epoche per cui fare l'apprendimento; l'errore che si deve raggiungere prima di fermare la fase di learning (espresso come scarto quadratico medio). Sono state inoltre sfruttate alcune caratteristiche peculiari delle FANN, fra cui l'uso delle funzioni di callback tramite i puntatori a funzione, che sono risultate particolarmente utili per consentire la stampa dei dati di log ed il salvataggio della rete aggiornata ad ogni epoca;
3. **test**: questo programma consente di eseguire dei test con degli esempi presi dal test set. Ciascuna esecuzione stampa a terminale una rappresentazione della cifra letta, oltre ai valori delle unit di output;
4. **bulk-test**: programma pensato per costruire delle statistiche sulla bontà della rete, ha il solo scopo di eseguire molte volte la rete, sottoponendole degli esempi pescati in maniera casuale dal test set, e calcolando la percentuale di valori trovati correttamente, confrontando il valore target con quello calcolato dalla rete;
5. l'ultimo programma sviluppato, denominato **own-test**, parte dall'idea di estendere l'uso della rete ad esempi costruiti "al volo". Questo programma, infatti, prende come parametro della linea di comando il nome di un file immagine, che considererà come un esempio da sottoporre alla rete e ne stampa il risultato calcolato (per la lettura e la decodifica del file immagine, sono state usate le librerie DevIL). Tuttavia, come già spiegato in questo documento, gli esempi che la rete apprende sono stati centrati secondo il centro di massa dei pixel. Nella release attuale del programma non viene fatta tale operazione e si ritiene che sia

questa la causa principale del numero di errori molto maggiore rispetto agli esempi provenienti dal test set.

Per effettuare l'apprendimento, si è osservato che l'algoritmo Backpropagation è molto più lento di Quickprop, tuttavia, quando l'errore comincia ad essere sufficientemente basso, il primo si comporta decisamente meglio del secondo. Per questo motivo, si è pensato di combinare i due e di utilizzare Quickprop per raggiungere con estrema rapidità dei primi risultati grezzi, che sono poi stati raffinati tramite Backpropagation.

L'algoritmo Quickprop, secondo quanto l'autore dello stesso riporta in [7], si differenzia dal Backpropagation nella funzione che ricalcola i pesi. La nuova funzione, infatti, assume che la funzione errore sia una parabola e calcola, di conseguenza, la derivata seconda al fine di raggiungere rapidamente il minimo della parabola stessa. Il gradiente è calcolato in questa maniera (con S si fa riferimento alla derivata parziale):

$$\Delta(t)w_{ij} = \frac{S(t)}{S(t-1) - S(t)}\Delta(t-1)w_{ij}$$

A.4 Reti multiple

È stato provato anche un approccio differente al problema. Per ogni cifra si è costruito una rete a tre layer: un neurone di input per ogni pixel dell'immagine della cifra, da 20 a 80 neuroni nel layer nascosto e un unico neurone di output. La singola rete apprende a riconoscere la cifra associata: per esempio la rete associata alla cifra cinque apprende a restituire uno se la cifra in input è un cinque e zero altrimenti. Per riconoscere una cifra si calcola l'output delle diverse reti per una stessa immagine e la cifra riconosciuta quella associata alla rete che ha l'output più alto.

Per ogni cifra si crea una rete con 784 input, un opportuno numero di neuroni nascosti e un unico output. Ogni rete apprende il database, per un numero opportuno di epoche. A questo punto si può testare la bontà delle reti ottenute. Per automatizzare questo procedimento sono stati scritti anche alcuni shell script che combinano i tool di base.

A.5 Risultati con rete unica

Come già descritto pocanzi, nell'esempio che è stato realizzato tramite le librerie FANN sono stati utilizzati più algoritmi di apprendimento, alcune volte anche incrociandoli fra loro, al fine di migliorare le prestazioni. Si riportano, di seguito alcuni dei risultati raggiunti.

Tutti i vari test prevedevano il cambiamento solo dell'algoritmo di apprendimento, la struttura e la topologia della rete sono rimaste invariate. Anche il training set non è cambiato da ciascun test: esso è composto da un sottoinsieme del training set originario, in particolare contiene 30000 esempi, ovvero circa 3000 esempi per ciascuna delle cifre da apprendere.

In particolare, una rete, è stata allenata per un lungo periodo, nel provare diverse configurazioni. Tramite questa, è possibile raggiungere risultati che arrivano anche al 95/96% di esempi classificati correttamente. In [8] si dice che, con questa struttura, si deve riuscire ad arrivare a risultati ancora maggiori: 98.4%. Data la brevità dei tempi di apprendimento accettati per questa sperimentazione, si pensa sia possibile raggiungere risultati come quelli promessi del citato articolo semplicemente aumentando il numero di epoche a disposizione dell'algoritmo. In [8], infatti, il risultato è raggiunto con 7 giorni di apprendimento, e sfruttando l'intero training set.

A.6 Risultati con reti multiple

Sono state testate reti con 20, 30, 40, 60, 80 e 120 neuroni nascosti. L'apprendimento è stato per tutte di sole 60 epoche perché già dopo qualche decina di epoche si notavano fenomeni di overfitting o comunque l'apprendimento non procedeva sensibilmente. L'algoritmo backpropagation incrementale, che non funzionava nell'approccio con rete unica, si è rivelato veloce e stabile. Questa differenza si può ipotizzare dipendere dalla maggiore semplicità della funzione che approssimiamo, dato che deve riconoscere una sola cifra alla volta.

Il risultato migliore non è sempre raggiunto dalla rete con più neuroni nascosti: alcune cifre sono riconosciute meglio da reti più piccole, mentre altre cifre non sono riconosciute efficacemente neanche dalle reti più grandi. Questo indica come la difficoltà di riconoscere una cifra sia fortemente influenzata dalla rappresentazione grafica della stessa.

La precisione migliore è raggiunta dalle dieci reti con 30 neuroni nascosti, quindi un numero di neuroni comparabili con quello usato nell'approccio a rete singola.

A.7 Conclusioni

Per quanto riguarda la risoluzione del problema tramite le librerie FANN, l'esperienza è stata estremamente positiva: la potenza della tecnica delle Reti Neurali è stata addomesticata tramite queste librerie estremamente potenti e relativamente facili da utilizzare. È stato entusiasmante osservare

sperimentalmente la piena riuscita della risoluzione del problema, tramite l'applicazione diretta delle tecniche teoriche.

È stato possibile rendersi conto in maniera abbastanza rilevante della versatilità e delle potenzialità delle Reti Neurali e, dopo l'iniziale stupore sulla loro estrema efficacia si è cercato di capirne a fondo il loro funzionamento e le particolari sfumature degli algoritmi di apprendimento.

B Caso di studio (SOM): “Islands of Music”

Il seguente caso di studio descrive uno studio che è stato effettuato dall'azienda *Last.fm Ltd* per il loro portale <http://www.last.fm>. Questo sito web fornisce una applicazione che consente lo *scrobbling* (ovvero l'invio di informazioni) della musica che ciascuno dei suoi utenti ascolta, attraverso vari media (software per il proprio personal computer, lettori portatili quali iPod, etc.). Il sistema raccoglie i dati sui titoli e gli autori delle varie canzoni ascoltate. Successivamente, sempre tramite la loro web application, questi dati sono elaborati e vengono fornite informazioni che possono essere ritenute utili dagli utenti, quali: artisti più ascoltati, altri utenti con gusti simili ai propri, autori e canzoni che non ho mai sentito ma che potrei trovare piacevoli, etc.

Il sistema consente all'utente di aggiungere ad artisti e canzoni delle etichette di testo, dette “tag”, che possano in qualche maniera descrivere l'oggetto in questione.

Grazie ai dati raccolti, è stato possibile, per i gestori del sito, costruire una mappa detta “Islands of Music”, riportata in [Figura 10](#).

L'obiettivo di questo progetto è quello di verificare la vicinanza o la lontananza di particolari generi musicali (fra quelli ascoltati dagli utenti dell'applicazione).

B.1 Realizzazione

Per la realizzazione di questa mappa è stata utilizzata una self-organizing map di 13000 utenti con tutti i relativi artisti associati. Come prima operazione è stata fatta una normalizzazione sull'insieme dei dati delle canzoni di ciascun utente, al fine di estrarre solo i dati più caratterizzanti. Dopo questo step, si è ottenuto un insieme di 2000 etichette distinte ed i vari utenti sono stati descritti da 13000 matrici sparse, in uno spazio vettoriale a 2000 dimensioni.

Utilizzando la decomposizione ai valori singoli, la dimensionalità delle matrici è stata ridotta a 120 dimensioni. In questa struttura sono state fatte

ged artist”, dove sono raccolti molti degli artisti che non sono stati etichettati o che non risultano particolarmente ascoltati.

Come specificato dagli autori della “mappa” non è generalmente vero che i dati abbiano senso e che la clusterizzazione dia risultati interessanti, a causa del rumore che può essere presente nei dati.

Riferimenti bibliografici

- [1] Alessandro Sperduti. Appunti del corso di Sistemi Intelligenti, 2008. <http://www.math.unipd.it/~sperduti/sistemi-intelligenti.08.html>.
- [2] Ben Kröse and Patrick van der Smagt. *Introduction to Neural Networks*. 1996.
- [3] Elias Pampalk. Islands of Music. http://www.lastfm.it/user/E1i45/journal/2008/05/20/1zjmr8_islands_of_music, 20 May 2008.
- [4] M. Volta. Appunti del corso di Controllo e Gestione dei Sistemi Ambientali. <http://automatica.ing.unibs.it/mco/cgsa/neurali/>.
- [5] Michael J. A. Berry and Gordon S. Linoff. *Mastering Data Mining*. Wiley, 2000.
- [6] Raúl Rojas. *Neural Networks - A Systematic Introduction*. Springer-Verlag, 1996.
- [7] Scott E. Fahlman. Faster-learning variations on back-propagation: An empirical study. In *Proceedings, 1988 Connectionist Models Summer School*, Los Altos CA, 1998. Morgan-Kaufmann.
- [8] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2 edition, 30 December 2002.
- [9] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.