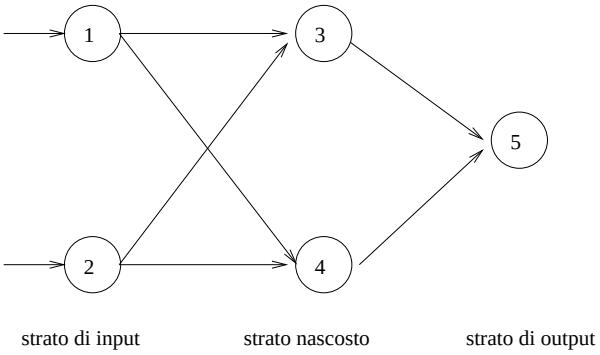
Reti Neurali

In una piccola ditta di assemblaggio di circuiti integrati è appena arrivata una partita di chip logici con due piedini di input e uno di output, di cui purtroppo s'è persa la bolla di accompagnamento: non si sa né da dove arrivino i chip né che funzione abbiano. Un tecnico di laboratorio conduce alcuni esperimenti su coppie di input di voltaggio prese a caso, e l'output è dato dalla tabella sotto.

0.1	0.1	0.1
0.9	0.9	0.05
0.0	0.95	0.98
0.95	0.1	0.95

Per ottenere un responso affidabile sarebbero necessari almeno un centinaio di esperimenti simili, ma purtroppo fare questi esperimenti ha danneggiato un numero non indifferente di chip, e quindi si è deciso di utilizzare un modello matematico per simulare la risposta dei chip sulla base delle risposte già date.

Si utilizzi la rete neurale in figura sotto e il set di apprendimento nella tabella sopra per individuare un insieme di pesi sui lati che possa ben approssimare gli esperimenti già fatti. Si calcolino le uscite della rete con le coppie di input (0,0), (0,1), (1,0), (1,1) e si stabilisca che tipo di chip è stato consegnato alla ditta.



Traccia.

1. Si utilizzi dapprima l'algoritmo Multistart visto nel laboratorio precedente per determinare un buon punto iniziale. Si utilizzi poi l'algoritmo del gradiente — o quello di Newton — con una tolleranza $\varepsilon > 0$ molto bassa e un numero massimo di iterazioni molto alto per trovare un buon ottimo locale.

2. Si noti che gli algoritmi utilizzati risultano molto lenti per via del fatto che la ricerca esatta dei sottoproblemi unidimensionali prende molto tempo. Si implementi una ricerca unidimensionale di *backtracking* e si paragonino i risultati con quelli ottenuti in precedenza.

Soluzione

Si considerino i valori di output associati a ogni neurone $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5)$ dove x_i è l'output dell'i-esimo neurone; si considerino inoltre i pesi sulle sinapsi $\mathbf{u} = (u_{13}, u_{14}, u_{23}, u_{24}, u_{35}, u_{45})$. Siano $\mathbf{y} = (y_1, y_2)$ i valori di input ai neuroni 1, 2 nello strato di input, e sia $z = x_5$ l'output della rete. Sia $h : \mathbb{R}^2 \to \mathbb{R}$ la mappa $\mathbf{y} \to z$ data dalla rete neurale, parametrizzata da $\mathbf{u}$. Per una formalizzazione di questa rappresentazione, si veda il Riquadro 1.

Sia $\varphi(\xi) = \frac{e^{\xi} - e^{-\xi}}{e^{\xi} + e^{-\xi}}$ la funzione di attivazione. Si ha:

$$\begin{aligned} z = x_5 &= \varphi(u_{35}x_3 + u_{45}x_4) \\ x_3 &= \varphi(u_{13}x_1 + u_{23}x_2) \\ x_4 &= \varphi(u_{14}x_1 + u_{24}x_2) \\ x_1 &= \varphi(y_1) \\ x_2 &= \varphi(y_2). \end{aligned}$$

Pertanto si può scrivere $z = h(\mathbf{u}, \mathbf{y})$ come:

$$h(\mathbf{u}, \mathbf{y}) = \varphi(u_{35}\varphi(u_{13}\varphi(y_1) + u_{23}\varphi(y_2)) + u_{45}\varphi(u_{14}\varphi(y_1) + u_{24}\varphi(y_2))).$$

Si tratta quindi di risolvere il problema:

$$\min_{\mathbf{u}} \sum_{i=1}^{5} ||z_i - h(\mathbf{u}, \mathbf{y}_i)||^2,$$

dove $(\mathbf{y}_i, z_i)$ sono le triple date in tabella nel testo del problema.

Ordiniamo $\mathbf{u}$ come $(u_{13}, u_{14}, u_{23}, u_{24}, u_{35}, u_{45})$. Nei file di MATLAB si troverà naturalmente la notazione vettoriale `u(1)`, ..., `u(6)`.

Il codice per l'algoritmo del gradiente è dato dai file `grad.m`, `linesearch.m`, `steepestdescent.m`; il codice per l'algoritmo Multistart è in `multistart.m`. Tutti questi file possono essere trovati nel testo del laboratorio precedente.

La funzione di attivazione è data da:

```
% file activation.m
function phival = activation(xi)
  phival = (exp(xi) - exp(-xi)) / (exp(xi) + exp(-xi));
%end function
```

La funzione h è data da:

```
% file hmap.m
function hval = hmap(u, y)
  hval = activation(u(5)*activation(u(1)*activation(y(1)) +
                                    u(2)*activation(y(2))) +
                    u(6)*activation(u(3)*activation(y(1)) +
                                    u(4)*activation(y(2))));
%end function
```

Infine, la funzione obiettivo da minimizzare è data da:

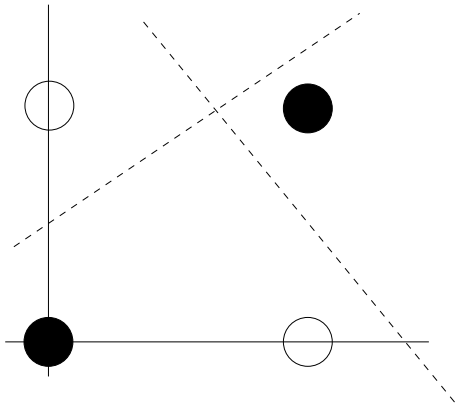
```
% file neuralnet.m
function xval = neuralnet(u)
  y = [ 0.1 , 0.1; ...
        0.9 , 0.9; ...
        0.0 , 0.95; ...
        0.95, 0.1 ...
      ];
  z = [ 0.1; 0.05; 0.98 ; 0.95  ];
  n = length(z);
  xval = 0;
  t = zeros(2,1);
  for i = 1:n
    t(1) = y(i,1);
    t(2) = y(i,2);
    xval = xval + (z(i) - hmap(u, t))^2;
  end
%end function
```

Si ottiene:

```
>> [ustar, fstar, k] = multistart('neuralnet', 6, 0.1, 300)
ustar =

    4.5181
    4.4230
   -0.6932
   -0.8472
    2.8475
    3.3667

fstar = 0.74498
k = 301

>> [ustar, fstar, tol, k] = steepestdescent('neuralnet', u, 0.01, 100)
ustar =

   3.74970
   3.68348
  -0.31021
  -0.31640
   1.67045
   3.49302

fstar = 0.53459
tol = 0.10794
k = 101

>> hmap(ustar, [0;0])
ans = 0

>> hmap(ustar, [0;1])
ans = 0.68181

>> hmap(ustar, [1;0])
ans = 0.69066

>> hmap(ustar, [1;1])
ans = 0.11889
```

Si conclude che i chip sono probabilmente delle porte logiche XOR. Si noti che la funzione XOR non è lineare, quindi non è sufficiente utilizzare i metodi di classificazione lineare. Si veda nella figura sotto: non esiste un iperpiano che separa i punti "neri" dai

punti "bianchi" (le linee tratteggiate sono iperpiani separatori che però non riescono a classificare correttamente tutti i punti).



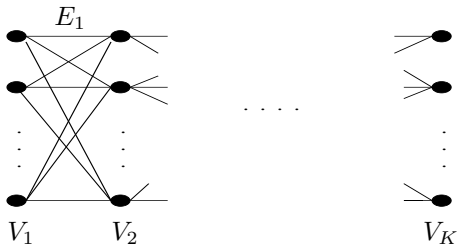
Si invita il lettore a provare lo stesso esercizio anche con la funzione di attivazione sigmoidale $\psi(\xi)$ definita nel Riquadro 1.

MATLAB dispone di demo interessanti sulle reti neurali (si veda nell'online help). Si ricorda infine che esistono i comandi MATLAB (non implementati su Octave) `newff`, `train`, `sim`. Il seguente materiale è tratto dall'header del file `newff.m`.

```
Here is a problem consisting of inputs P and targets T that we would
like to solve with a network.

  P = [0 1 2 3 4 5 6 7 8 9 10];
  T = [0 1 2 3 4 3 2 1 2 3 4];

Here a two-layer feed-forward network is created.  The network's
input ranges from [0 to 10].  The first layer has five TANSIG
neurons, the second layer has one PURELIN neuron.  The TRAINLM
network training function is to be used.

  net = newff([0 10],[5 1],{'tansig' 'purelin'});

Here the network is simulated and its output plotted against
the targets.

  Y = sim(net,P);
   plot(P,T,P,Y,'o')

Here the network is trained for 50 epochs.  Again the network's
output is plotted.

  net.trainParam.epochs = 50;
  net = train(net,P,T);
  Y = sim(net,P);
   plot(P,T,P,Y,'o')
```

Riquadro 1.

La fase di apprendimento delle reti neurali, tramite l'algoritmo di back-propagation, è in sostanza la soluzione iterativa di un problema di minimi quadrati per mezzo dell'algoritmo del gradiente. Si consideri un modello neurale multistrato modellato da un grafo K-partito $G = (V, E)$ dove V è partizionato in $V_1, \ldots, V_K$, E in $E_1, \ldots, E_{K-1}$ e ogni (V_k, V_{k+1}, E_k) $(k \leq K-1)$ è un grafo bipartito completo. I nodi di questo grafo sono i neuroni, e i lati le sinapsi, come indicato nella figura sotto.



Ad ogni lato del grafo è associato un peso u_k^{sj}, dove $k \leq K$ è lo strato e $\{s, j\}$ indica il lato, con $s \in V_k$ e $j \in V_{k+1}$. Ad ogni vertice si associa un valore di output x_k^j, dove k è lo strato j è il vertice in V_k. Il valore di input del j-esimo vertice nel $k+1$-esimo strato è una funzione lineare dei valori di output allo strato precedente: $\sum_{s=1}^{|V_k|} u_k^{sj} x_k^s$, per $j \leq |V_{k+1}|$. L'output del j-esimo vertice nel $k+1$-esimo strato è una funzione φ del suo input, detta funzione di attivazione. Quindi

$$x_{k+1}^j = \varphi\left(\sum_{s=1}^{|V_k|} u_k^{sj} x_k^s\right) \quad \forall\, j \leq |V_{k+1}|, 2 \leq k \leq K-1.$$

Di solito si utilizza la funzione di attivazione $\varphi(\xi) = \frac{e^{\xi}-e^{-\xi}}{e^{\xi}+e^{-\xi}}$, oppure la funzione sigmoidale $\psi(\xi) = \frac{1}{1+e^{-\xi}}$. Si assume che lo strato $k = 1$ sia lo strato di input dell'intera rete neurale: i vertici del primo strato hanno un valore di input dato dall'utente, che codifica l'input della rete. I valori di output dell'ultimo strato forniscono la risposta della rete. In pratica, la rete è una mappa h, parametrizzata dal vettore dei pesi sui lati $\mathbf{u}$, che trasforma un vettore di input $\mathbf{x}_1$ in un vettore di output $\mathbf{x}_K$, in modo che $\mathbf{x}_K = h(\mathbf{u}, \mathbf{x}_1)$.
Supponiamo di avere un'insieme di m coppie input/output $T = \{(\mathbf{y}_i, \mathbf{z}_i) \mid i \leq m\}$. La fase di apprendimento della rete neurale fa sì che i pesi $\mathbf{u}$ vengano modificati affinché a ogni input corrisponda l'output desiderato. Al termine di questa fase, la rete è pronta per l'uso: si spera che ad ogni nuovo vettore di input (di cui non si conosce l'output a priori) la rete fornisca un output "sensato" in base alle coppie in T che ha già "visto" prima.
L'algoritmo di back-propagation utilizzato per l'apprendimento non è altro che l'algoritmo del gradiente applicato strato per strato al problema di minimi quadrati

$$\min_{\mathbf{u}} \sum_{i=1}^{m} ||\mathbf{z}_i - h(\mathbf{u}, \mathbf{y}_i)||^2.$$

Il fatto che le variabili siano i pesi $\mathbf{u}$ e che h in generale può essere nonlineare e nonconvessa rende l'intero problema nonconvesso. Si consideri poi che nei problemi pratici m è di solito un numero molto grande; gli approcci risolutivi devono quindi limitarsi ad algoritmi molto semplici (come per l'appunto quello del gradiente) e certe volte applicati addirittura in modo stocastico (il gradiente viene computato solo per un sottoinsieme casuale delle componenti del vettore).

Figura 1: Problema di ottimizzazione nelle reti neurali.