Reti Neurali Artificiali per lo studio del mercato

Università degli studi di Brescia - Dipartimento di metodi quantitativi
Marco Sandri (sandri.marco@gmail.com)

Funzione di risposta di una RNA - Parte 2

Obiettivi:

(1) Realizzare una rete neurale artificiale (RNA) **facendo uso del toolboox Neural Networks**; si prende prima di tutto in considerazione il comando `newff` (riga 27) che serve a creare in modo rapido e agevole un oggetto "rete feedforward" di Matlab (`net`, una struttura di Matlab) che potrà poi essere utilizzato per vari scopi; successivamente (righe 29-35) si vede come è possibile accedere alle proprietà dell'oggetto `net` per impostare in modo casuale i pesi delle connessioni; da ultimo, il comando `sim` (riga 38) permette di ottenere i valori dell'output (o risposta) della rete in funzione dei diversi valori degli input.

(2) Mostrare come cambia la funzione di risposta della RNA al variare di:
- numero di neuroni nello strato nascosto; questo parametro (che determina la complessità delle rete) nello script viene chiamato **`Nhid`** (riga 5);
- intensità dei pesi delle connessioni; questo parametro nello script viene chiamato **`amp_pesi`** (riga 17);

Variando i parametri `Nhid` e `amp_pesi` si osservano alcuni fatti molto importanti:
- all'aumentare del numero di neuroni nello strato nascosto la risposta della rete si fa sempre più irregolare, frastagliata, complessa, in grado quindi di rappresentare comportamenti sempre meno "lisci" (smooth);
- indipendentemente dalla complessità dalla rete, l'intensità delle connessioni può determinare comportamenti più o meno lisci della RNA: intensità dei pesi molto basse (e quindi connessioni molto deboli o addirittura disattivate) portano ad avere reti con funzioni di risposta molto lisce, molto regolari. Man mano che i pesi aumentano e quindi man mano che le connessioni si fanno più intense, la risposta della rete diventa sempre più irregolare, sempre liscia.

(3) Le considerazioni del punto (2) fanno luce su una importante idea volta a controllare il fenomeno dell'overfitting, idea che tratteremo più avanti parlando della cosiddetta **regularization**: premesso che una RNA con una funzione di risposta molto liscia rischia di entrare in overfitting molto meno di una RNA con una funzione di risposta molto irregolare, un metodo per controllare l'overfitting sarà quello di **controllare non solo il numero di neuroni nello strato nascosto ma anche l'intensità dei pesi delle connessioni**.
In fase di stima di una RNA vedremo che per evitare l'overfitting, si andrà ad introdurre una penalità per le RNA con pesi di elevata intensità, andando così a preferire quelle con connessioni debolmente attivate e con una più bassa attitudine a sovra-adattarsi ai dati.

Attenzione! In questo esempio **i pesi della rete non vengono ottimizzati** ma impostati in modo del tutto casuale. Non stiamo cioè assolutamente stimando i pesi della rete sulla base di un criterio di ottimo (minimi quadrati). Lo scopo è qui soltanto di vedere come cambia la funzione di risposta della rete al variare del numero di neuroni nello strato nascosto e dell'intensità delle connessioni. La stima dei parametri della RNA feedforward (cioè l'apprendimento della rete) verrà esaminato più avanti.

Calcolo della funzione di risposta della rete

```
close all; clear all;
Ninp=2; Nhid=10; Nout=1; N=50;
```

Imposta i parametri della rete: no. input, no. neuroni strato nascosto, no. output e no. osservazioni per ciascun input

```
x1 = linspace(-2,2,N);
x2 = x1;
[X1,X2] = meshgrid(x1,x2);
X = [ones(N^2,1) X1(:) X2(:)];
```

Genera i dati campionari

```
pesi = repmat([-20 20],Ninp+1,1);
net = newff(pesi,[Nhid Nout],{'logsig','purelin'});
net.IW{1} = 6*randn(Nhid,Ninp+1);
net.LW{2,1} = 10*randn(Nout,Nhid);
```

Imposta l'architettura e i pesi della rete

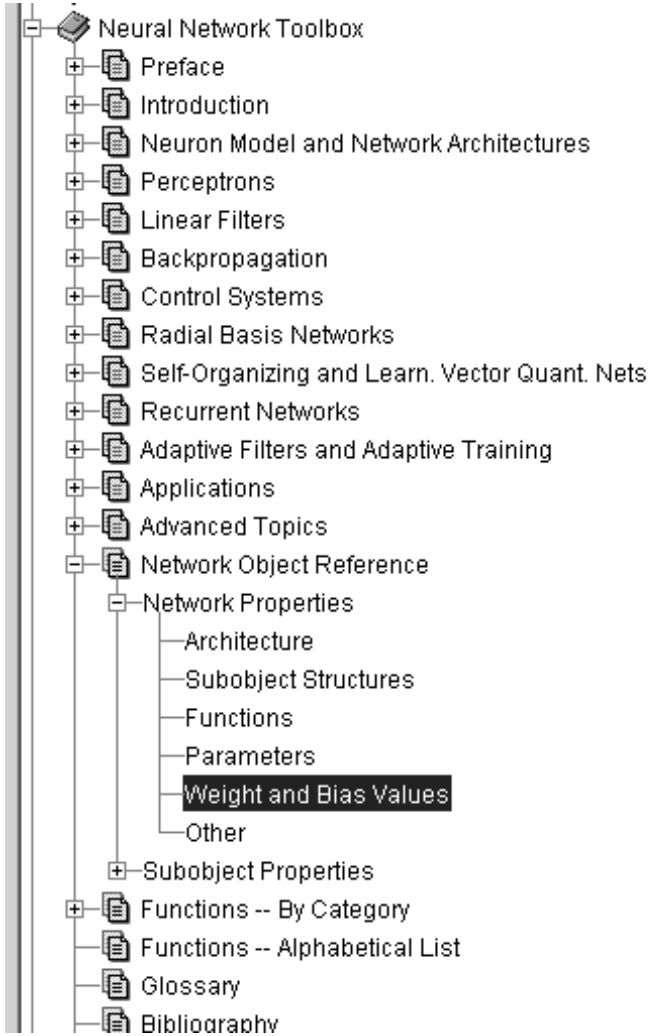
```
y = sim(net, X');
Y = reshape(y,N,N);
surf(X1,X2,Y);
```

Calcola l'output della rete e traccia il grafico della funzione di risposta output = f(input)

L'oggetto *neural network*

Le proprietà di un *Neural Network Object* si suddividono in 5 macro categorie:

- architecture
- subobject structures
- functions
- parameters
- weight and bias values



L'oggetto *neural network*

Architecture:

numInputs: 1
numLayers: 2
biasConnect: [1; 1]
inputConnect: [1; 0]
layerConnect: [0 0; 1 0]
outputConnect: [0 1]
targetConnect: [0 1]

numOutputs: 1 (read-only)
numTargets: 1 (read-only)
numInputDelays: 0 (read-only)
numLayerDelays: 0 (read-only)

net.numInputs
Numero di strati di input della rete
Da non confondere con il numero di neuroni
nello strato di input (net.inputs{1}.size)
che sono invece 3 nel nostro esempio !

net.numLayers
Numero di strati (1 nascosto + 1 output)
net.biasConnect
Strati che hanno bias (nascosto e output)
net.inputConnect
Quali strati hanno connessioni con l'input
net.layerConnect
Quali strati hanno connessione con altri layer

L'oggetto *neural network*

Subobject Structures:

inputs: {1x1 cell} of inputs
layers: {2x1 cell} of layers
outputs: {1x2 cell} ...
targets: {1x2 cell} ...
biases: {2x1 cell} ...
inputWeights: {2x1 cell} ...
layerWeights: {2x2 cell} ...

net.inputs{1}
Parametri strato di input
.range .size .userdata
net.layers{1}
Parametri strato nascosto
.dimensions .distanceFcn .distances .initFcn
.netInputFcn … … .transferFcn .userdata
net.layers{1}.netinputfcn $\phi_{11}(\boldsymbol{x}_i, \boldsymbol{\omega}_{11})$
'netprod' 'netsum'
net.layers{1}.transferfcn
'logsig' 'tansig' 'satlin' 'hardlim' etc…
net.layers{1}.initfcn
Funzione di inizializzazione dei pesi:
'initnw' Nguyen-Widrow init. function
'initbw' By-weight-and-bias init. function

L'oggetto *neural network*

Functions:

adaptFcn: 'trains'
initFcn: 'initlay'
performFcn: 'mse'
trainFcn: 'trainlm'

net.performfcn
Funzione di errore: 'mae', 'mse', 'msereg', 'sse'
net.trainfcn
Algoritmo usato per minimizzare la funzione d'errore e quindi stimare i pesi della rete

Training Functions	
trainbfg	BFGS quasi-Newton backpropagation.
trainbr	Bayesian regularization.
traincgb	Powell-Beale conjugate gradient backpropagation.
traincgf	Fletcher-Powell conjugate gradient backpropagation.
traincgp	Polak-Ribiere conjugate gradient backpropagation.
traingd	Gradient descent backpropagation.
traingda	Gradient descent with adaptive lr backpropagation.
traingdm	Gradient descent with momentum backpropagation.
traingdx	Gradient descent with momentum and adaptive lr backprop.
trainlm	Levenberg-Marquardt backpropagation.
trainoss	One-step secant backpropagation.
trainrp	Resilient backpropagation (Rprop).
trainscg	Scaled conjugate gradient backpropagation.
trainb	Batch training with weight and bias learning rules.
trainc	Cyclical order incremental training with learning functions.
trainr	Random order incremental training with learning functions.

L'oggetto *neural network*

Parameters:

adaptParam: .passes
initParam: (none)
performParam: (none)
trainParam: .epochs, .goal, .max_fail, .mem_reduc, .min_grad, .mu, .mu_dec, .mu_inc, .mu_max, .show, .time

net.trainparam
Arresto dell'algoritmo di ottimizzazione:
- dopo un tempo pari a .time
- dopo un numero di iterazioni .epochs
- quando l'errore è sceso al di sotto di .goal
- quando il gradiente è sceso sotto .min_grad
- quando l'errore sul set di validazione è aumentato più di .max_fail volte dall'ultima volta che è diminuito (questo ovviamente quando si usa la cross-validation)

net.trainparam.show
Mostra il grafico dell'andamento dell'errore in-sample ogni .show iterazioni

L'oggetto *neural network*

Weight and bias values:

IW: {2x1 cell}
containing 1 input weight matrix
LW: {2x2 cell}
containing 1 layer weight matrix
b: {2x1 cell}
containing 2 bias vectors

net.IW
Matrice dei pesi Ω_1 che collegano gli input al primo strato nascosto
net.LW{2,1}
Matrice dei pesi Ω_2 che collegano il primo strato nascosto al secondo (strato di output)
net.b{1}
Vettore dei bias (pesi degli input a valore 1) dello strato nascosto
net.b{2}
Vettore del bias dello strato di output