1. Simulazioni numeriche

Generazione del campione secondo un predefinito meccanismo generatore dei dati (DGP)

Stima dei parametri del modello

Stima delle 'performance' del modello (capacità predittiva, adattamento ai dati, etc.)

1.1 Simulazione e stima DGP lineare

```
close all; clear all;
n  = 1000; Nvar = 10; sd_noise = 1;
a = 2;  b = -3;
X = rand(n,Nvar);
ydet = a*X(:,2) + b;
y = ydet + sd_noise*randn(n,1);
```

Genera DGP:

$y_i = a*x_i + b + \varepsilon_i$

a = 2

b = -3

```
[x2,ord2]=sort(X(:,2));
plot(x2,ydet(ord2),'-',x2,y(ord2),'.')
```

Grafico parte deterministica e stocastica del DGP

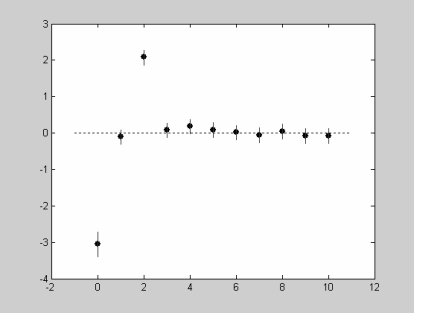
```
Xc = [ones(n,1) X];
[beta, beta_ci] = regress(y, Xc)
```

Stima dei coefficienti del modello lineare

1.1 Simulazione e stima DGP lineare

Grafico dei coefficienti stimati e del loro intervallo di confidenza

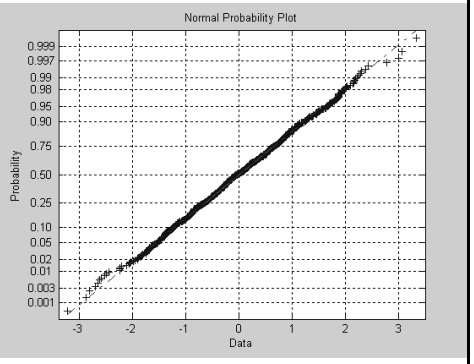
```
figure;
plot([0:Nvar], beta, '. k', 'MarkerSize', 20);
hold on;
for ii = 0:Nvar
   line([ii ii], [beta_ci(ii+1,1) beta_
end
line([-1 Nvar+1], [0 0], 'Color', 'b',
hold off;
```



1.1 Simulazione e stima DGP lineare

Grafico dei residui e analisi della loro distribuzione

```
figure;
yprev = Xc*beta;
residui = y - yprev;
hist(residui,20);

figure;
plot(ydet, yprev, '.', ydet, ydet, '.');

figure;
normplot(residui);

[h,p,jbstat,cv] = jbtest(residui)
```



1.1 Simulazione e stima DGP lineare

Sperimentare e commentare i risultati ottenuti variando:

- dimensione campionaria (n);
- deviazione standard della componente stocastica ε_i (sd_noise);
- i coefficienti a e b del DGP lineare;
- il numero di variabili osservate (Nvar);
- il numero di variabili esplicative

(si modifichi la linea di comando:

ydet = a*X(:,2) - b;

ad es. con

ydet = 3*X(:,2) - 2.7*X(:,6) - 1.8*X(:,7) - 4.6;)

1.1 Simulazione e stima DGP lineare

```
close all; clear all;
n = 1000; Nvar = 10;
Niter = 100; sd_noise = 1;
a = 3; b = -2.7; c = -1.8; d = -4.6;

beta = zeros(Niter,Nvar+1);
for ii = 1:Niter
   X = rand(n,Nvar);
   ydet = a*X(:,2) + b*X(:,6) + c*X(:,7) + d;
   y = ydet + sd_noise*randn(n,1);
   Xc = [ones(n,1) X];
   [coeff, coeff_ci] = regress(y, Xc);
   beta(ii,:) = coeff';
end
```

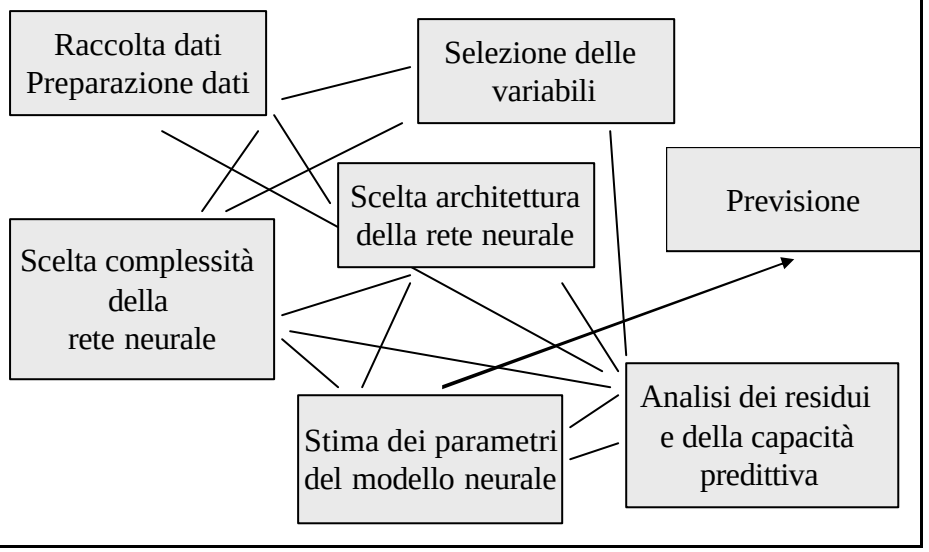
Generazione di Niter campioni da DGP lineare e analisi della distribuzione dei coefficienti stimati

1.1 Simulazione e stima DGP lineare

```
Nfig= floor((Nvar+1)/4)+1; Nv = 1;
for jj=1:Nfig
   figure;
   for ii = 1:4
      if Nv <= (Nvar+1)
         subplot(2,2,ii);
         normplot(beta(:,Nv));
         title(['Media = ',num2str(mean(beta(:,Nv))), ...
               ' -- SD = ', num2str(std(beta(:,Nv)))]);
         Nv = Nv+1;
      else
         break;
      end
   end
end
```

Istogrammi relativi alla distribuzione di tutti i coefficienti stimati

2. Costruire un modello neurale



2.1 Pre-processing dei dati

Talune trasformazioni dei dati di partenza possono essere utili per migliorare la successiva fase di stima dei parametri della rete

Riscalamento fra min e max

```
X = randn(100,2); y = unidrnd(10, 100, 1);
[Xn,minX,maxX,yn,miny,maxy] = premnmx(X,y);
```

Trasformazione X e y

```
netopt = train(net,Xn,yn);
```

Training rete

```
Xnewtra = tramnmx(Xnew,minX,maxX);
yout = sim(netopt,Xnewtra);
yhat = postmnmx(yout,miny,maxy);
```

Trasforma nuove X
Previsione
Antitrasf. Previsioni Y

2.1 Pre-processing dei dati

Standardizzazione

```
X = randn(100,2); y = unidrnd(10, 100, 1);
[Xn,meanX,sdX,yn,meany,sdy] = prestd(X,y);
```

Standardizzazione X e y

```
netopt = train(net,Xn,yn)
```

Training rete

```
Xnewtra = trastd(Xnew,meanX,sdX);
yout = sim(netopt,Xnewtra)
yhat = poststd(yout,meany,sdy);
```

Standardizzazione nuove X
Previsione
Antistandardizz. Previsioni Y

2.1 Pre-processing dei dati

Analisi delle componenti principali (PCA)

```
X = randn(100,2); y  = unidrnd(10, 100, 1);
[Xn,meanX,sdX,yn,meany,sdy] = prestd(X,y);
[Xpca,pcamat] = prepca(Xn, 0.05)
```

Standardizzazione X e y e PCA su X

Elimina comp. princ. che spiegano meno del 5% della varianza totale

```
netopt = train(net,Xpca,yn)
```

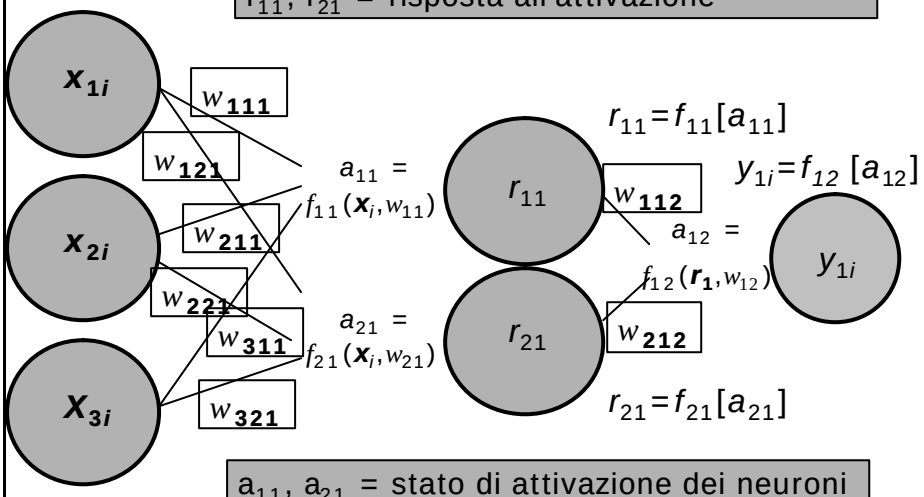
Training rete

```
Xnewtra = trastd(Xnew,meanX,sdX);
Xnewpca = trapca(Xnewtra, pcamat);
yout = sim(netopt,Xnewpca)
yhat = postmnmx(yout,meany,sdy);
```

Standardizz. e PCA nuove X
Previsione
Antistandardizz. Previsioni Y

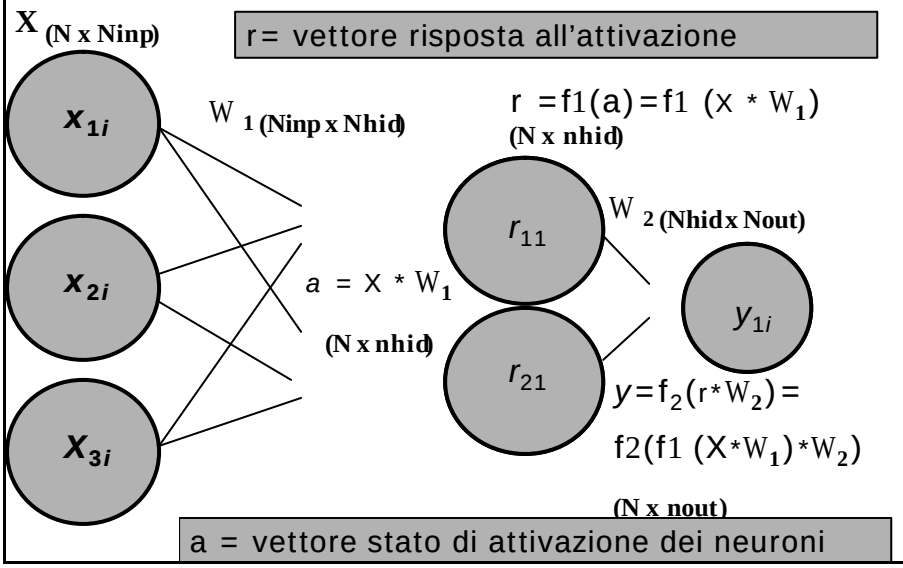
2.2 Calcolo funzione di risposta della rete

r_{11}, r_{21} = risposta all'attivazione



a_{11}, a_{21} = stato di attivazione dei neuroni

2.2 Calcolo funzione di risposta della rete



2.2 Calcolo funzione di risposta della rete

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

Imposta parametri rete

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

Genera dati campionari

```
Omega1 = 6*randn(Ninp+1,Nhid);
Omega2 = 10*randn(Nhid,Nout);
```

Imposta pesi rete

```
y = logsig(X*Omega1)*Omega2;
Y = reshape(y,N,N);
surf(X1,X2,Y);
```

Calcola output rete e traccia grafico output = f(input)

2.2 Calcolo funzione di risposta della rete

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

Imposta parametri rete

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

Genera dati campionari

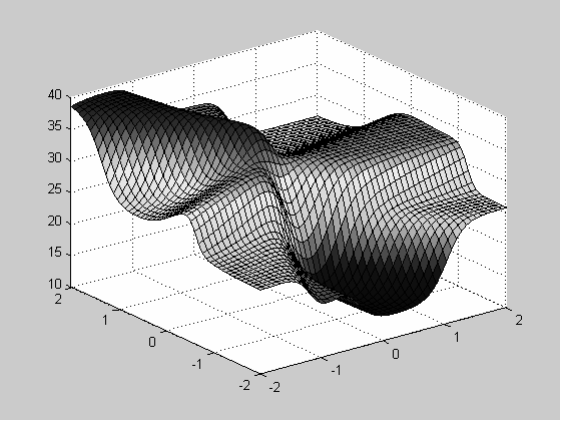
```
pesi = repmat([-20 20],Ninp+1,1);
net = newff(pesi,[Nhid Nout],{'logsig','purelin'});
net.IW{1} = 6*randn(Nhid,Ninp+1);
net.LW{2,1} = 10*randn(Nout,Nhid);
```

Imposta architettura e pesi rete

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

Calcola output rete e traccia grafico output = f(input)

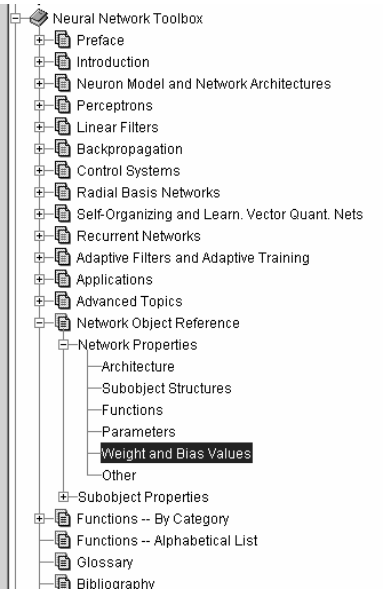
2.2 Calcolo funzione di risposta della rete



2.3 L'oggetto *'net'*

Neural Network object:

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



2.3 L'oggetto *'net'*

Neural Network object:

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 vettori di input della rete (1)
Da non confondere con:
net.inputs{1}.size
che sono invece 3 !

net.numLayers
Numero di strati (1 nascosto + 1 output)

net.biasConnect
layers che hanno bias (1 nascosto+output)
net.inputConnect
quali layer hanno connessioni con l'input
net.layerConnect
quali layer hanno connessione con altri layer
etc...

2.3 L'oggetto *'net'*

Neural Network object:

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}
.range .size .userdata
net.layers{1}
.dimensions .distanceFcn .distances .initFcn
.netInputFcn … … .transferFcn .userdata
net.layers{1}.netinputfcn $f_{11}(\boldsymbol{x}_i, w_{11})$
'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

2.3 L'oggetto *'net'*

Neural Network object:

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

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.

2.3 L'oggetto *'net'*

Neural Network object:

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 e' sceso al di sotto di .goal
- quando il gradiente e' 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

2.3 L'oggetto *'net'*

Neural Network object:

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 W_1 che collegano gli input al primo strato nascosto
net.LW{2,1}
Matrice dei pesi W_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

2.4 Stima dei parametri del modello neurale

Con il termine *'training'* si indica quello che in ambito statistico viene indicato con il termine di <u>stima dei parametri</u> della rete.
La stima dei parametri di un modello equivale al problema della ricerca delle soluzioni di un problema di ottimo:
- trovare i parametri del modello che massimizzano la funzione di (log-)verosimiglianza
- trovare i parametri del modello che minimizzano una funzione d'errore (mae, mse, msereg, etc..)
Sia f(x, θ) il nostro modello, dove x è il vettore di input (variabili osservate) e θ il vettore dei parametri del modello.
La stima dei parametri consiste ad es. nella soluzione del problema di ottimo:

$$\min_{\theta \in \Theta} \sum_{i=1}^{N} (y_i - f(x_i, \theta))^2$$

2.4 Stima dei parametri del modello neurale

A volte, specie con i modelli lineari, la soluzione al problema di ottimo la si può trovare in modo semplice e indicare con una formula. Nel modello di regressione lineare ad es. la stima a minimi quadrati del vettore dei parametri è:

$$\beta^* = (\mathbf{X^T X})^{-1} \mathbf{X^T Y}$$

Nella maggior parte dei casi la soluzione del problema di ottimo deve però essere trovata attraverso <u>metodi numerici.</u>
L'impiego di questi metodi richiede la conoscenza dei diversi algoritmi ed una notevole esperienza.

2.4 Stima di un modello neurale

```
close all;clear all;
N=100; ninp=5; nhid=30;
X =rand(N,ninp);
ydet= 1.2*(X(:,1).^3 -.5)+sin(X(:,1)*pi*1.1);
y= ydet+0.2*randn(N,1);
```

Genera il campione

```
plot(X(:,1),y,'.',X(:,1),ydet,'x')
```

```
[Xn,minX,maxX,yn,miny,maxy] = premnmx(X',y');
net=newff(minmax(Xn),[nhid1],{'tansig','tansig'},'trainlm');
net=init(net);
optnet=train(net, Xn , yn);
yout=sim(optnet, Xn);
yhat = postmnmx(yout,miny,maxy);
```

Stima e previsione

```
[xs, idx]= sort(X(:,1));
y= y(idx); yhat=yhat(idx); ydet=ydet(idx);
figure;
plot(xs,y,'.',xs,yhat,'-',xs,ydet,'x')
```

2.5 Migliorare la capacità di generalizzazione

'Regularization': si modifica la funzione d'errore (*performance function*) introducendo un termine che esprime la somma dei quadrati dei pesi della rete:

$$msereg = \gamma mse + (1-\gamma) msw$$

$$F = mse = \frac{1}{N}\sum_{i=1}^{N}(e_i)^2 = \frac{1}{N}\sum_{i=1}^{N}(t_i - a_i)^2 \qquad msw = \frac{1}{n}\sum_{j=1}^{n} w_j^2$$

L'idea e' di penalizzare i pesi della rete con valori elevati. Ciò equivale a forzare la rete ad avere una funzione di risposta più *'smooth'*.
Il parametro $\gamma \in [0,1]$ è detto *'performance ratio'* e tanto più è elevato più questa penalizzazione è marcata. Il valore ottimo di questo parametro è difficile da stimare. L'algoritmo di MacKay (Bayesian regularization) propone una stima di questo parametro. In Matlab il comando per la bayesian regularization è: trainbr

2.5 Migliorare la capacità di generalizzazione

'Regularization'

```
close all;clear all;
N=100; ninp=5; nhid=30;
X =rand(N,ninp);
ydet= 1.2*(X(:,1).^3 -.5)+sin(X(:,1)*pi*1.1);
y= ydet+0.2*randn(N,1);
plot(X(:,1),y,'.',X(:,1),ydet,'x')
[Xn,minX,maxX,yn,miny,maxy] = premnmx(X',y');
net=newff(minmax(Xn),[nhid1],{'tansig','tansig'},'trainbr');
net=init(net);
optnet=train(net, Xn , yn);
yout=sim(optnet, Xn);
yhat = postmnmx(yout,miny,maxy);
[xs, idx]= sort(X(:,1));
y= y(idx); yhat=yhat(idx); ydet=ydet(idx);
figure;
plot(xs,y,'.',xs,yhat,' -',xs,ydet,'x')
```

Genera campione

Stima e previsione

2.5 Migliorare la capacità di generalizzazione

'Regularization'

Importanti vantaggi:
(1) controllo dell'overfitting
(2) l'algoritmo suggerisce un valore 'ottimale' della complessità della rete, cioè del numero di parametri (pesi e bias) della rete.

```
TRAINBR, Epoch 0/100, SSE 6597.55/0, SSW 651.893, Grad 5.23e+003/1.00e-010, #Par 3.51e+002/351
TRAINBR, Epoch 25/100, SSE 13.7522/0, SSW 9.6795, Grad 4.12e-001/1.00e-010, #Par 2.25e+001/351
TRAINBR, Epoch 50/100, SSE 13.4953/0, SSW 10.6611, Grad 3.53e-001/1.00e-010, #Par 2.34e+001/351
TRAINBR, Epoch 75/100, SSE 13.4937/0, SSW 10.6654, Grad 3.50e-001/1.00e-010, #Par 2.34e+001/351
TRAINBR, Epoch 100/100, SSE 13.4939/0, SSW 10.663, Grad 3.50e-001/1.00e-010, #Par 2.34e+001/351
```

2.5 Migliorare la capacità di generalizzazione

'Early stopping': si divide il campione in training e crossvalidation (CV) set. Con il primo si stimano i parametri della rete. Sul secondo si mantiene sotto controllo l'errore out-of-sample (una misura della capacità di generalizzazione). Quando la rete inizia ad entrare in overfitting l'errore sul training set continua a decrescere mentre sul CV set inizia a crescere. L'early stopping suggerisce di fermare la procedura di stima dei parametri in prossimità del punto di minimo dell'errore sul CV set.
In Matlab questo tipo di procedura è disponibile per qualsiasi algoritmo di ottimizzazione. L'early stopping lavora in modo talora molto efficace se combinato con la bayesian regularization.

2.5 Migliorare la capacità di generalizzazione

'Early stopping'

```
close all; clear all;
N=500; Ncv=0.5*N; ;ninp=5; nhid=60;
X =rand(N,ninp);
ydet= 1.2*(X(:,1).^3 -.5)+sin(X(:,1)*pi*1.1);
y= ydet+0.2*randn(N,1);
plot(X(:,1),y,'.',X(:,1),ydet,'x')
[Xn,minX,maxX,yn,miny,maxy] = premnmx(X',y');
Xtrain=Xn(:,1:(N-Ncv)); ytrain=yn(1:(N-Ncv));
val.P=Xn(:,(N-Ncv+1):end); val.T=yn((N-Ncv+1):end);
net=newff(minmax(Xn),[nhid1],{'tansig','purelin'},'trainlm
net=init(net); net.trainParam.epochs=300;
optnet=train(net, Xtrain, ytrain,[],[],val);
yout=sim(optnet, Xn);
yhat = postmnmx(yout,miny,maxy);
[xs, idx]= sort(X(:,1));
y= y(idx); yhat=yhat(idx); ydet=ydet(idx);
figure; plot(xs,y,'.',xs,yhat,'-',xs,ydet,'x')
```

Genera campione

Stima e previsione

2.6 Scelta dell'architettura

La scelta dell'architettura della rete dipende prima di tutto dalla natura del problema che si intende risolvere.

- **Se si deve individuare un modello di regressione (non lineare), allora la scelta cade ad es. sulle reti feedforward**
- **Se si vuole modellizzare/predire una serie temporale, allora si possono utilizzare le reti ricorrenti (ad es. le reti di Elman); in esse gli output ritardati sono input della rete;**
- **Se il problema invece è di stabilire a quale di un set di k classi le singole unità campionarie appartengono (classificazione) allora la scelta dovrà ad es. cadere sulle reti feedforward opportunamente adattate o sulle reti LVQ (learning vector quantization);**
- **Se il problema è di individuare nel campione, sulla base delle variabili osservate, dei gruppi omogenei di unità (cluster), allora si dovranno scegliere ad es. le SOM (self-organizing map);**

2.6 Scelta dell'architettura

Questi comandi creano un oggetto *'net'* ed impostano le sue proprietà in accordo con l'architettura scelta.

Analogamente, è possibile costruire nuove architetture ed effettuare poi la stima dei parametri (comando *'train'*) e la previsione (comando *'sim'*)

New Networks Functions	
network	Create a custom neural network.
newc	Create a competitive layer.
newcf	Create a cascade-forward backpropagation network.
newelm	Create an Elman backpropagation network.
newff	Create a feed-forward backpropagation network.
newfftd	Create a feed-forward input-delay backprop network.
newgrnn	Design a generalized regression neural network.
newhop	Create a Hopfield recurrent network.
newlin	Create a linear layer.
newlind	Design a linear layer.
newlvq	Create a learning vector quantization network
newp	Create a perceptron.
newpnn	Design a probabilistic neural network.
newrb	Design a radial basis network.
newrbe	Design an exact radial basis network.
newsom	Create a self-organizing map.

2.7 Algoritmi numerici di ottimizzazione

La scelta dell'algoritmo di ottimizzazione si basa su diversi elementi:

- la natura e le dimensioni del problema di ottimo (n. di parametri da stimare, no. di dati nel campione,tipo di funzione da ottimizzare);

- la disponibilità di risorse computazionali (velocità processore, memoria disponibile) e le corrispondenti richieste computazionali dell'algoritmo (velocità di convergenza, tempi di calcolo per ogni iterazione, quantità di memoria richiesta);

2.7 Algoritmi numerici di ottimizzazione

Gli algoritmi di ottimizzazione numerica sono in generale processi iterativi del tipo: $x_{t+1} = x_t + D_t$

D_t **esprime lo spostamento dell'algoritmo da** x_t **verso il punto di ottimo. Di questo spostamento di deve determinare direzione e ampiezza. Molti algoritmi calcolano** D_t **sulla base del gradiente della funzione e della sua Hessiana.**

L'arresto dell'algoritmo di ottimizzazione avviene:

- dopo che e' trascorso un tempo massimo

- dopo un numero massimo di iterazioni

- quando l'errore e' sceso al di sotto di una soglia fissata

- quando il gradiente e' sceso sotto una soglia fissata

- quando l'errore sul set di validazione è aumentato più di un certo numero di volte dall'ultima volta che è diminuito (questo ovviamente quando si usa la cross-validation)

2.7 Algoritmi numerici di ottimizzazione

Nel "Neural Network Toolbox" di Matlab ci sono diversi algoritmi di ottimizzazione disponibili, alcuni specificamente pensati per il training delle reti neurali

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.

2.7 Algoritmi numerici di ottimizzazione

Ottimizzazione non lineare non vincolata.

[xopt,fval,exitflag,output] = fminsearch(fun,x0,options,P1,P2,...)

Questo comando usa il metodo of Nelder-Mead per minimizzare funzioni non vincolate non lineari *f: R^n ® R*, senza richiedere l'uso del gradiente $\tilde{N}$ (in forma numerica o analitica).
L'algoritmo fa parte di una sottoclasse di *metodi di ricerca diretta* che ad ogni passo mantengono un simplesso non degenere di *n* dimensioni.
Tale metodo è in grado di trattare funzioni discontinue, specie se la discontinuità giace lontano dal punto di ottimo.
Esso inoltre può individuare solo ottimi locali.c

2.7 Algoritmi numerici di ottimizzazione

Input:
fun = nome della funzione;
x0 = condizione iniziale;
options = insieme di opzioni;
P1, P2, … = eventuali parametri di fun

Output:
xopt = valore ottimo di x
fval = valore della f obbiettivo in xopt
exitflag = indica se l'algoritmo converge o non converge
output = struttura contenente informazioni sull'ottimizzazione

Elenco opzioni di fminsearch: optimset('fminsearch')
Display = Tipo di visualizzazione. 'off' nessun output; 'iter' output ad ogni iterazione; 'final' solo l'output finale; 'notify' (default) output solo quando la funzione converge
MaxFunEvals = Numero massimo consentito di valutazioni di funzione
MaxIter = Numero massimo di iterazioni consentite
TolX = Tolleranza di arresto

2.7 Algoritmi numerici di ottimizzazione

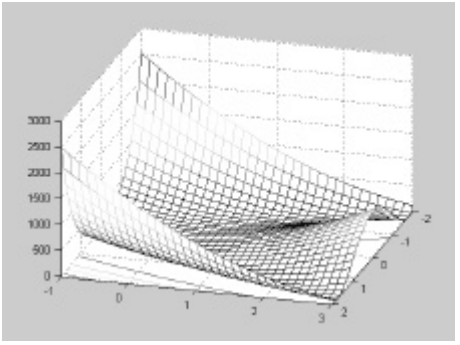
Esempio. Trovare il minimo della funzione di Rosenbrock:

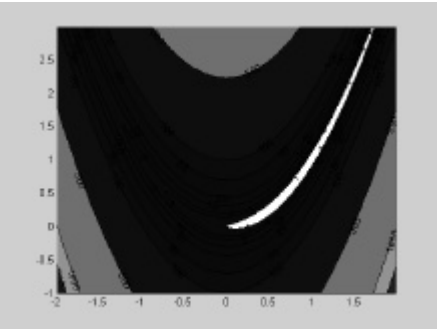
$$f(x,y) = 100*(y - x^2)^2 + (1-x)^2$$

```
xx = [-2: 0.125: 2];
yy = [-1: 0.125: 3];
[X,Y]=meshgrid(xx,yy);
Z = 100.*(Y-X.^2).^2 + (1-X).^2;

figure(1)
meshc(X,Y,Z);
```

```
xx = [-2: 0.0156: 2];
yy = [-1: 0.0156: 3];
[X,Y]=meshgrid(xx,yy);
Z = 100.*(Y-X.^2).^2 + (1-X).^2;

figure(2)
[c,h]=contourf(X,Y,Z,…
[2100 1600 500 100 50 20 10 5 3 1]);
clabel(c,h);
```

2.7 Algoritmi numerici di ottimizzazione



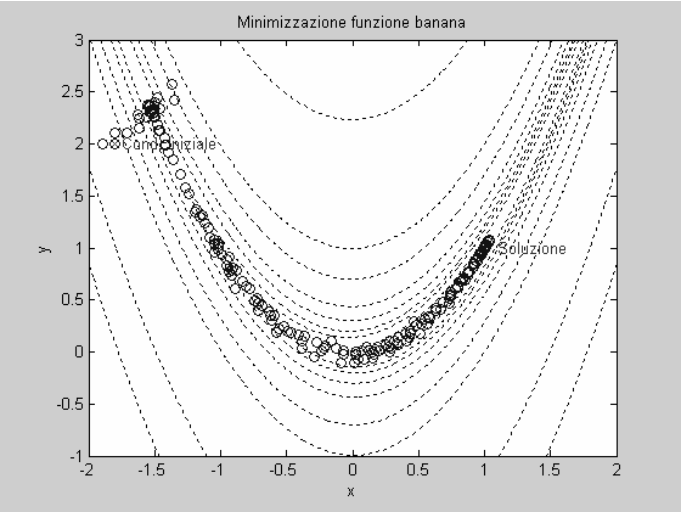


2.7 Algoritmi numerici di ottimizzazione

```
function out =  traccia_punto(x)
plot(x(1),x(2),'o','Erasemode','none')
pause(.2); drawnow;
out = [ ];
```

```
figure(1)
conts = [1000 500 100 50 20 10 5 3 1];
contour(X,Y,Z,conts,'k:');
xlabel('x'); ylabel('y'); title('Minimizzazione funzione di Rosenbrock');
hold on
plot(-1.8,2,'xr'); text(-1.75,2, 'Cond. Iniziale', 'Color', 'r');
plot(1,1,'x'); text(1.1,1,'Soluzione', 'Color', 'r');
drawnow;

f ='[100*(x(2)-x(1)^2)^2+(1-x(1))^2; traccia_punto(x)]';
x0=[-1.8,2];
opzioni=optimset('LargeScale','off','MaxFunEvals',300);
[xopt, fval, exitflag, output] = fminsearch( f, x0, opzioni);
```

2.7 Algoritmi numerici di ottimizzazione



xopt = [1.0000 1.0000] fval = 2.2999e-010

1.1 Simulazione e stima DGP lineare

Visualizza parametri dell'algoritmo di ottimizzazione:
optimset('fminsearch')

Sperimentare e commentare i risultati ottenuti variando:
- Condizione iniziale (*x0*);
- Numero massimo di iterazioni (*MaxIter*);
- Sostituire *fminsearch* con *fminunc*

2.7 Algoritmi numerici di ottimizzazione

Ottimizzazione non lineare non vincolata.
Curve fitting.

[x,resnorm,residual,exitflag,output,lambda,jacobian] = lsqnonlin(fun,x0,lb,ub,options,P1,P2, ...)

Questo comando permette di trovare le soluzioni di problemi di minimi quadrati non lineari per funzioni *f: R^n ® R*:

$$\min \sum_{i=1}^{n} f(x_i)^2$$

2.7 Algoritmi numerici di ottimizzazione

```
% Simulazione dati
N=200;
x = -pi+2*pi*rand(N,1);
y=-2*x+3.5*sin(x)-1+randn(N,1);
```

```
% Definisci function
function diff=f1(coef,x,y)

a = coef(1);b = coef(2);c = coef(3);
diff = (a .* x + b .* sin(x)+c) - y;
```

```
% Metodo Levenberg Marquardt
x0=50*randn(1,3);
opzioni = optimset('LargeScale','off','MaxFunEvals',3000);
opzioni = optimset(opzioni,'LevenbergMarq','on','Jacobian','off');
[stime,rnorm,res,extflag,out]= lsqnonlin('f1',x0,[],[],opzioni,x,y);

disp(['Parametri stimati (a, b c) = ', num2str(stime)]);
[rms, yfit]=fun(stime,x,y);
plot(x, y, '.', x, yfit, '.');
```

2.7 Algoritmi numerici di ottimizzazione

Curve fitting. LSQCURVEFIT.
Modello non lineare: a + b x + c sin(x)

```
% Simulazione dati
N=200;
x = -pi+2*pi*rand(N,1);
y=-2*x+3.5*sin(x)-1+randn(N,1);
% Stima parametri
x0=50*randn(1,3);
[stime,resnorm] = lsqcurvefit('f2',x0,x,y)

yfit=fun2(stime,x);
plot(x, y, '.', x, yfit, '.');
```

```
function y = f2(coeff,x)

a = coeff(1);
b = coeff(2);
c = coeff(3);
y = a .* x + b .* sin(x)+c;
```