2 Reference

This chapter contains a detailed description of all the functions in the Neural Network Based System Identification Toolbox. The information given here is more or less identical to that obtained from the online help facility.

General Network Training Algorithms	
batbp	Batch version of the back-propagation algorithm.
igls	Iterated generalized least squares training of multi-output networks
incbp	Recursive (/incremental) version of back-propagation.
marq	Levenberg-Marquardt method.
marqlm	Memory-saving implementation of the Levenberg-Marquardt method.
rpe	Recursive prediction error (~Gauss-Newton) method.

Data Manipulation	
dscale	Scale data to zero mean and variance 1.

Nonlinear System Identification	
lipschit	Determine the lag space.
nnarmax1	Identify a neural network ARMAX (or ARMA) model (linear MA-filter).
nnarmax2	Identify a neural network ARMAX (or ARMA) model.
nnarx	Identify a neural network ARX (or AR) model.
nnarxm	Identify a multi-output neural network ARX (or AR) model.
nnigls	Iterated generalized LS training of multi-output NNARX models
nniol	Identify a neural network model suited for I-O linearization type control.
nnoe	Identify a neural network Output Error model.
nnssif	Identify a neural network state space innovations form model.
nnrarmx1	Recursive counterpart to NNARMAX1.
nnrarmx2	Recursive counterpart to NNARMAX2.
nnrarx	Recursive counterpart to NNARX.

Determination of Optimal Network Architecture	
netstruc	Extract weight matrices from matrix of parameter vectors.
nnprune	Prune models of dynamic systems with Optimal Brain Surgeon (OBS).
obdprune	Prune feed-forward networks with Optimal Brain Damage (OBD).
obsprune	Prune feed-forward networks with Optimal Brain Surgeon (OBS).

Evaluation of Trained Networks	
fpe	Final Prediction Error estimate of generalization error for feed-forward nets.
ifvalid	Validation of models generated by NNSSIF.
ioleval	Validation of models generated by NNIOL.
kpredict	*k*-step ahead prediction of network output.
loo	Leave-One-Out estimate of generalization error for feed-forward networks.
nneval	Validation of feed-forward networks (trained by marq, batbp, incbp, or rpe).
nnfpe	FPE-estimate for I-O models of dynamic systems.
nnloo	Leave-One-Out estimate of generalization error for NNARX models
nnsimul	Simulate model of dynamic system.
nnvalid	Validation of I-O models of dynamic systems.
wrescale	Rescale weights of a trained network.
xcorrel	High order cross-correlation functions.

Miscellanous Utilities	
crossco	Calculate correlation coefficients.
drawnet	Draws a two-layer feed-forward network.
getgrad	Derivative of network outputs w.r.t. the weights.
pmntanh	Fast tanh-function.
settrain	Set parameters for training algorithms.

Demonstration Programs	
test1	Demonstrates different training methods on a curve fitting example.
test2	Demonstrates the NNARX function.
test3	Demonstrates the NNARMAX2 function.
test4	Demonstrates the NNSSIF function.
test5	Demonstrates the NNOE function.
test6	Demonstrates the effect of regularization by simple weight decay.
test7	Demonstrates pruning by OBS on the sunspot benchmark problem.

batbp

Purpose

Batch version of the back-propagation algorithm.

Synopsis

[w1,w2,critvec,iter]=batbp(NetDef,W1,W2,PHI,Y,trparms)

Input

NetDef: Network definition.
W1: Input-to-hidden layer weights. The matrix dimension is [(# of hidden units) * (inputs + 1)] (the 1 is due to the bias) Use [] for a random initialization.
W2: Hidden-to-output layer weights. The matrix dimension is [(outputs) * (# of hidden units + 1)] Use [] for a random initialization.
PHI: Input data [(# of inputs) * (# of data)]
Y: Output data [(outputs) * (# of data)]
trparms: Data structure containing parameters associated with the training algorithm (optional). Use the function SETTRAIN if you do not want to use the default values.

Output

w1, w2: Weight matrices when the training is completed.
critvec: Vector containing the criterion of fit after each iteration.
iter: # of iterations.

Description

Given a set of corresponding input-output pairs and an initial network [W1,W2,critvec,iter] = batbp(NetDef,W1,W2,PHI,Y,trparms) trains the network with back-propagation.

The activation functions must be either *linear* or *tanh*. The network architecture is defined by the matrix 'NetDef' consisting of two rows. The first row specifies the hidden layer while the second specifies the output layer.

E.g.:
```
NetDef =  ['LHHHH'
           'LL---']
```
(L = Linear, H = tanh)

Notice that the bias is included as the last column in the weight matrices.

Example

Generate data as sinusoidal+noise

```
>>  PHI = 2*pi*rand(1,300);
>>  Y = sin(PHI) + 0.2*randn(1,300);
>>  plot(PHI,Y,'+');
```

Initialize Network. 5 *tanh* hidden units, 1 *linear* output.

```
>>  W1 = rand(5,2);
>>  W2 = rand(1,6);
>>  NetDef = ['HHHHH';'L----'];
>>  drawnet(W1,W2,eps)
>>  trparms = settrain;
>>  trparms = settrain(trparms,'maxiter',1000,'eta',2e-4);
>> [W1,W2,critvec,iter]=batbp(NetDef,W1,W2,PHI,Y,trparms);
```

Plot the value of the criterion as a function of the iteration number

```
>>  semilogy(critvec); grid;
>>  xlabel('Iteration');
>>  ylabel('Criterion')
```

Algorithm

Back-propagation is a gradient descent algorithm where the computations are ordered in a simple fashion by taking advantage of the special architecture of a neural network. In this implementation the step size is fixed.

See Also

INCBP, NNEVAL, MARQ, RPE.

References

J. Hertz, A. Krogh & R.G. Palmer: *"Introduction to the theory of Neural Computation,"* Addison-Wesley, 1991.

crossco

Purpose

Calculate correlation coefficients.

Synopsis

Cross-correlation coefficients:
coefs = crossco(v,w)
coefs = crossco(v,w,maxlag);

Autocorrelation coefficients:
coefs = crossco(v,v,maxlag);

Input

v and w are two signals contained in vectors of equal length.
Default max. lag is 25 or the vector length -1.

Description

The correlation coefficient is the correlation function normalized such that the autocorrelation will be 1 for lag 0.

$$\hat{r}_{vw}(\tau) = \frac{\sum_{t=1}^{N-\tau}(v(t)-\overline{v})(w(t-\tau)-\overline{w})}{\left(\sum_{t=1}^{N}(v(t)-\overline{v})^2\right)^{1/2}\left(\sum_{t=1}^{N}(w(t)-\overline{w})^2\right)^{1/2}}$$

drawnet

Purpose

Draw a two layer neural network.

Synopsis

drawnet(W1,W2)
drawnet(W1,W2,CancelVal,instring,outstring)

Input

W1:	Input-to-hidden layer weights. The matrix dimension is [(# of hidden units) * (inputs + 1)] (the 1 is due to the bias)
W2:	Hidden-to-output layer weights. The matrix dimension is [(outputs) * (# of hidden units + 1)]
CancelVal:	(Optional) Draw only weights/biases exceeding this value.
instring:	(Optional). A cell structure containing in each cell a string to be assigned to the corresponding input. The number of cells should thus match the number of inputs. If it is not present, or it is empty {}, the inputs are simply numbered.
outstring:	(Optional). A cell structure containing in each cell a string to be assigned to the corresponding output. The number of cells should thus match the number of outputs.

Description

Draws the network specified by the weights in W1 and W2. Positive weights are represented by a solid line while a dashed line represents a negative weight. Only weights and biases larger than 'CancelVal' are drawn. A bias is represented by a vertical line through the neuron.

Example

Initialize Network. 5 *tanh* hidden units and 1 *linear* output
>> *W1 = rand(5,3);*
>> *W2 = rand(1,6);*
>> *str1 = {‘ x1’ ’ x2’ ’x253’};*
>> *str2 = {‘y’};*
>> *drawnet(W1,W2,eps,str1,str2)*

See Also

OBDPRUNE, OBSPRUNE, NNPRUNE.

dscale

Purpose

Scale data to zero mean and variance 1 before training

Synopsis

[Xs,Xscale]=dscale(X)
Xs=dscale(X,Xscale)

Input

X: Data matrix (dimension is # of data vectors in matrix * # of data points).
Xscale: If Xscale is provided the data in X is scaled to the mean in Xscale(1) and the standard deviation Xscale(2).

Output

Xs: Scaled data matrix
Xscale: Matrix containing sample mean (column 1) and standard deviation (column 2) for each data vector in X.

See Also

WRESCALE on how to rescale the weights of the trained network.

References

Y. Le Cun, I. Kanter, S.A. Solla: *"Eigenvalues of Covariance Matrices: Application to Neural-Network Learning,"* Physical Review Letters, Vol 66, No. 18, pp. 2396-2399, 1991.

fpe

Purpose

Final prediction error (FPE) estimate of the avarage generalization error.

Synopsis

[FPE,deff,varest,H] = fpe(NetDef,W1,W2,PHI,Y,trparms)

Input

See for example the function MARQ.

Output

FPE: The Final prediction error estimate.
deff: The effective number of weights.
varest: Estimate of the noise variance.
H: The Gauss-Newton Hessian.

Description

[FPE,deff,varest,H] = fpe(NetDef,W1,W2,PHI,Y,trparms) calculates Akaike's final prediction error estimate of the average generalization error. The function returns the final prediction error estimate (FPE), the effective number of weights in the network if the network has been trained with weight decay, an estimate of the noise variance, and the Gauss-Newton Hessian. It is important that the network has been trained to the minimum of the criterion before this function is called.

See Also

LOO for the Leave-One-Out estimate.
NNFPE gives the FPE estimate for models of dynamic systems.

References

J. Larsen & L.K. Hansen: *"Generalization Performance of Regularized Neural Network Models,"* Proc. of the IEEE Workshop on Neural networks for Signal Proc. IV, Piscataway, New Jersey, pp.42-51, 1994.

L. Ljung: *"System Identification - Theory for the User,"* Prentice-Hall, 1987.

getgrad

Purpose

Derivative of network output with respect to the weights.

Synopsis

[PSI,E] = getgrad(method,NetDef,NN,W1,W2,Chat,Y,U)

Inputs

See NNVALID.
For time series, U is either left out or passed as a [].

Output

PSI: Matrix containing the derivative of the output w.r.t. each weight for each input-output pair in the data set. The dimension is [# of weights * # of data]

E: Prediction errors.

Description

Produces a matrix of derivatives of the network output w.r.t. each network weight for use in the functions NNPRUNE and NNFPE.

Examples

Network generated by nnarx (or nnrarx):
\>> *[PSI,E] = getgrad('nnarx',NetDef,NN,W1,W2,[],Y,U)*

Network generated by nnarmax1 (or nnrarmx1):
\>> *[PSI,E] = getgrad('nnarmax1',NetDef,NN,W1,W2,Chat,Y,U)*

Network generated by nnarmax2 (or nnrarmx2):
\>> *[PSI,E] = getgrad('nnarmax2',NetDef,NN,W1,W2,[],Y,U)*

Network generated by nnoe:
\>> *[PSI,E] = getgrad('nnoe',NetDef,NN,W1,W2,[],Y,U)*

See Also

NNPRUNE and NNFPE

ifvalid

Purpose

Validate state space models.

Synopsis

[Yhat,NSSE] = ifvalid(NetDef,nx,W1,W2,obsidx,Y,U)

Input

See the function NNSSIF.

Output

Yhat: Prediction of output(s).
NSSE: Normalized sum of squared errors.

Description

Validate a neural network based state space model of a dynamic system. I.e., a network model trained with the function NNSSIF.

The following plots are produced:
- Output(s) together with predicted output(s).
- Prediction error.
- Autocorrelation function of prediction error and cross-correlation between prediction error(s) and input(s).
- Histogram(s) showing the distribution of the prediction errors.
- Coefficients of extracted linear models.

Example

```
>> load spmdata
>> NetDef = ['HHHH';'LL--'];
>> trparms = settrain;
>> trparms = settrain(trparms,'maxiter',100,'D',1e-4,'skip',10);
>> [W1,W2,obsidx,critvec,iter,lambda] =...
                    nnssif(NetDef,2,[],[],[],trparms,y1,u1);
>> [yhat,NSSE]=ifvalid(NetDef,2,W1,W2,obsidx,y2,u2);
```

See Also

NNSSIF, NNVALID, NNEVAL, IOLEVAL

igls

Purpose

Iterated Generalized Least Squares training of neural networks with multiple outputs.

Synopsis

[w1,w2,lambda,Gamma]=igls(NetDef,W1,W2,trparms,Gamma0,PHI,Y)

Input

NetDef, W1, W2, trparms, PHI, Y: See the function MARQ.
Gamma0: Initial estimate of the covariance matrix for the noise. If passed as [] it is set to the identity matrix.
trparms: Vector containing parameters associated with the training (see MARQ). Default values (obtained if *trparms=[]*): trparms=[50 0 1 0]

Output

w1, w2, lambda: See the function MARQ.
Gamma: The estimated covariance matrix.

Description

A multi-output feedforward network and the noise covariance matrix are estimated with an iterative relaxation procedure.

It is important to notice that the network returned from this function will produce predictions of scaled outputs (see the *Algorithm* paragraph). It is necessary to multiply the output by *sqrtm(Gamma)* to obtain the unscaled predictions. If the network has linear output units one can instead scale the hidden-to-output layer weights: *W2= sqrtm(Gamma)*W2*.

Example

Generate data as two sinusoidals+noise

```
>> PHI = 2*pi*rand(1,300);
>> Y = [sin(PHI);cos(PHI)] + [0.1*randn(1,300);0.8*randn(1,300)]
>> plot(PHI,Y(1,:),'+',PHI,Y(2,:),'o');
```

Train an initial network with 5 *tanh* hidden units, 2 *linear* output

```
>> W1 = rand(5,2);
>> W2 = rand(1,6);
>> NetDef = ['HHHHH';'LL---'];
```

>> drawnet(W1,W2,eps,{'phi'}, {'y1' 'y2'})
>> trparms = settrain;
>> [W1,W2]=marq(NetDef,[],[],PHI,Y,trparms);

Apply the IGLS procedure 10 times and train 30 iterations in each step.
>> trparms=settrain(trparms,'maxiter',30,'repeat',10);
>> [w1,w2,lambda,Gamma]=igls(NetDef,W1,W2,trparms, [],PHI,Y);
*>> w2u=sqrtm(Gamma)*w2;*
>> [Yhat,E,NSSE]=nneval(NetDef,w1,w2u,PHI,Y);

Algorithm

The implemented IGLS procedure is very simple
for j=1:repeat,
Train the network
Estimate the covariance matrix
end

The network is trained with the function MARQ according to the criterion

$$\hat{\theta}_j = V_N(\theta, Z^N) = \frac{1}{2N}\sum_{t=1}^{N}\left(y(t) - \hat{y}(t|\theta)\right)^T \hat{\Lambda}_{j-1}^{-1}\left(y(t) - \hat{y}(t|\theta)\right)$$
$$= \frac{1}{2N}\sum_{t=1}^{N}\varepsilon^T(t,\theta)\hat{\Lambda}_{j-1}^{-1}\varepsilon(t,\theta)$$

and the covariance matrix is estimated as

$$\hat{\Lambda}_j = \frac{1}{N}\sum_{t=1}^{N}\varepsilon(t,\hat{\theta}^{(j)})\varepsilon^T(t,\hat{\theta}^{(j)})$$

To reduce the amount of computations the network is trained by first scaling the outputs as

$$\bar{y}(t) = \Sigma y(t)$$

where

$$\Lambda = \Sigma^T \Sigma$$

and subsequently train the network according to

$$\hat{\theta}_j = V_N(\theta, Z^N) = \frac{1}{2N}\sum_{t=1}^{N}\left(\bar{y}(t) - \hat{y}(t|\theta)\right)^T\left(\bar{y}(t) - \hat{y}(t|\theta)\right)$$

If the network has linear output units, W2 should be scaled by $W2u = \Sigma^{-1}W2$.

See Also

MARQ for Levenberg-Marquardt training.
NNARXM for identification of multi-output NNARX models
NNIGLS for igls estimation of multi-output NNARX models.

References

T.J Fog, J. Larsen, L.K. Hansen: *Training and Evaluation of Neural Networks for Multi-Variate Time-Series Processing*. Proc. IEEE International Conference on Neural Networks, Perth, Australia.

incbp

Purpose

Recursive (/incremental) version of the back-propagation algorithm.

Synopsis

[w1,w2,critvec,iter]=incbp(NetDef,W1,W2,PHI,Y,trparms)

Input

NetDef:	Network definition
W1:	Input-to-hidden layer weights. The matrix dimension is [(# of hidden units) * (inputs + 1)] (the 1 is due to the bias) Use [] for a random initialization.
W2:	Hidden-to-output layer weights. The matrix dimension is [(outputs) * (# of hidden units + 1)] Use [] for a random initialization.
PHI:	Input data [(# of inputs) * (# of data)]
Y:	Output data [(outputs) * (# of data)]
trparms:	Data structure containing parameters associated with the training algorithm (optional). Use the function SETTRAIN if you do not want to use the default values.

Output

w1, w2:	Weight matrices after training.
critvec:	Vector containing the criterion evaulated after each iteration.
iter :	# of iterations.

Description

Given a set of corresponding input-output pairs and an initial network INCBP trains a network with recursive back-propagation.

The activation functions must be either *linear* or *tanh*. The network architecture is defined by the matrix 'NetDef' consisting of two rows. The first row specifies the hidden layer while the second specifies the output layer.

E.g.:
```
NetDef =  ['LHHHH'
           'LL---']
```
(L = Linear, H = tanh)

Notice that the bias is included as the last column in the weight matrices!

Example

Generate data as sinusoidal+noise

```
>> PHI = 2*pi*rand(1,300);
>> Y = sin(PHI) + 0.2*randn(1,300);
>> plot(PHI,Y,'+');
```

Initialize Network. 5 *tanh* hidden units, 1 *linear* output

```
>> NetDef = ['HHHHH';'L----'];
>> trparms = settrain;
>> trparms = settrain(trparms,'maxiter',400,'eta',0.02);
>> [w1,w2,critvec,iter]=incbp(NetDef,[],[],PHI,Y,trparms);
>> drawnet(w1,w2);
```

Plot criterion evaluated after each iteration

```
>> semilogy(critvec); grid;
>> xlabel('Iteration');
>> ylabel('Criterion')
```

Algorithm

Back-propagation is a gradient descent algorithm where the computations are ordered in a simple fashion, taking advantage of the special architecture of a neural network. In this implementation the step size is fixed.

See Also

BATBP for the batch version.
RPE for a recursive Gauss-Newton algorithm.
MARQ, NNEVAL.

References

J. Hertz, A. Krogh & R.G. Palmer: *"Introduction to the theory of Neural Computation,"* Addison-Wesley, 1991.

ioleval

Purpose

Validate models generated by NNIOL.

Synopsis

[Yhat,NSSE]=ioleval(NetDeff,NetDefg,NN,W1f,W2f,W1g,W2g,Y,U)

Inputs

See the function NNIOL for an explanation of the inputs.

Outputs

Yhat: One-step ahead prediction of output.
NSSE: Normalized sum of squared error (SSE/2N).

Description

Evaluate a neural network based model on a form well-suited for control by discrete input-output linearization; i.e., a network model trained with the function NNIOL.

The following plots are produced:
- Observed output together with predicted output.
- Prediction error.
- Histogram showing the distribution of the prediction errors.

Example

```
>> load spmdata
>> NetDeff = ['HHHHH';'L----'];
>> NetDefg = ['HHH';'L--'];
>> NN=[2 2 1];
>> trparms = settrain;
>> trparms = settrain(trparms,'D',1e-3);
>> [W1f,W2f,W1g,W2g,critvec,iter,lambda] =...
                nniol(NetDeff,NetDefg,NN,[],[],[],[],trparms,y1,u1);
>> [yhat,NSSE]=ioleval(NetDeff,NetDefg,NN,W1f,W2f,W1g,W2g,y2,u2);
```

See Also

NNIOL, NNVALID, NNEVAL, IFVALID

kpredict

Purpose

k-step ahead prediction of system output.

Synopsis

Network generated by NNARX (or NNRARX):
Ypred = kpredict('nnarx',NetDef,NN,k,W1,W2,Y,U);

(likewise for networks generated with NNARMAX1+2 and NNOE)

Input

See NNVALID

Output

Ypred: Vector containing the *k*-step ahead predictions of the outputs.

NB! The function does not work for models generated by NNIOL, NNARXM, or NNSSIF.

Description

Determine the *k*-step ahead prediction of the output of a dynamic system and compare it to the observed output. The predictions are determined by feeding past predictions into the network where observations are not available and by setting unavailable residuals to zero. Except for NNOE models a predictor defined in this manner cannot be expected to be the optimal predictor.

Example

```
>> load spmdata
>> NetDef = ['HHHH';'L---'];
>> NN=[2 2 1];
>> trparms = [100 0 1 1e-3];
>> [W1,W2,critvec,iter,lambda]=nnarx(NetDef,NN,[],[],trparms,y1,u1);
>> ypred=kpredict('nnarx',NetDef,NN,10,W1,W2,y1,u1);
```

lipschit

Purpose

Determine the lag space.

Synopsis

[OrderIndexMat]=lipschit(U,Y,m,n)

Inputs

U: Sequence of inputs (row vector)
Y: Sequence of outputs (row vector)
m: Vector specifying the input lag spaces to investigate
n: Vector specifying the ouput lag spaces to investigate

Outputs

OrderIndexMat: A matrix containing the order indices for each combination of elements in the vectors *m* and *n*. The number of rows corresponds to the number of elements in *m*, while the number of columns corresponds to the number of elements in *n*.

Description

Given corresponding input and output sequences the function calculates a matrix of indices that can be helpful for determining a proper lag space structure (m and n) before identifying a model of a dynamic system:

$$y(t) = f(y(t-1),...,y(t-n), u(t-1),..., u(t-m))$$

An insufficient lag space structure leads to a large index. While increasing the lag space the index will decrease until a sufficiently large lag space structure is reached. Increasing the lag space further will not change the index significantly. In other words: look for the knee-point of the plot.

m is a vector specifying which input lag spaces to investigate and *n* is ditto for the output. If one is only interested in the order index for one particular choice of lag structure, *n* and *m* are specified as scalars, and only the order index is returned. In the more general case, where one or both are vectors, the function will also produce one or two plots.

Examples

o NNFIR model structure expected:
m=[1:20]; n=0;

o Time series:
U=[]; m=0;

o Check only *n*=*m*:
m=[1:5]; n=m;

Algorithm

The function should be used with some care. Do not rely on the results if the data is too corrupted by noise. Physical insight is by far the best tool for determination of the lag space.

At this point the function works for SISO systems only. Extension to the multivariable case should be straightforward, however.

See Also

Use the function DSCALE to scale the data.

Reference

X. He & H. Asada: *"A New Method for Identifying Orders of Input-Output Models for Nonlinear Dynamic Systems,"* Proc. of the American Control Conf., S.F., California, 1993.

loo

Purpose

Estimate the average generalization error by leave-one-out cross-validation.

Synopsis

[Eloo,H] = loo(NetDef,W1,W2,PHI,Y,trparms)

Input

NetDef, W1, W2,
PHI, Y, trparms : See the function MARQ
If the *maxiter* field in the data structure *trparms* is *0* linear unlearning is used for obtaining a cheap approximation to the LOO estimate. If *maxiter>0* the network will be retrained a maximum of *maxiter* iterations for each input-output pair that is left out.

Output

Eloo: The leave-one-out cross-validation estimate of the average generalization error.
H: The Gauss-Newton Hessian.

Description

LOO calculates an approximation to the leave-one-out estimate of the average generalization error. The function returns the loo-estimate along with the Gauss-Newton Hessian.

Algorithm

When the *maxiter* field in *trparms* is *0* “linear unlearning” is used to get a quick approximation to the LOO-estimate. This approximation is much easier to compute than the true LOO-estimate, but is in general less reliable. Typically it is comparable to the FPE-estimate. See the reference below for a derivation. Unless *maxiter*=0 it is recommended to set *maxiter* to 20-40.

See Also

FPE for Akaike’s final prediction error estimate.

Reference

L.K. Hansen and J. Larsen: *"Linear Unlearning for Cross-Validation,"* Advances in Computational Mathematics, 5, pp. 269-280, 1996.

marq

Purpose

Train a network with the Levenberg-Marquardt method.

Synopsis

[w1,w2,critvec,iteration,lambda] = marq(NetDef,W1,W2,PHI,Y,trparms)

Input

NetDef: Network definition.
W1: Input-to-hidden layer weights. The matrix dimension is [(# of hidden units) * (inputs + 1)] (the 1 is due to the bias) Use [] for a random initialization.
W2: Hidden-to-output layer weights. The matrix dimension is [(outputs) * (# of hidden units + 1)] Use [] for a random initialization.
PHI: Input data [(# of inputs) * (# of data)]
Y: Output data [(outputs) * (# of data)]
trparms: Data structure containing parameters associated with the training algorithm (optional). Use the function SETTRAIN if you do not want to use the default values.

Output

W1, W2: Weight matrices after training.
critvec: Vector containing the criterion evaluated after each iteration.
iteration: # of iterations.
lambda: The final value of lambda. Relevant if retraining is desired.

Description

Given a set of corresponding input-output pairs and an initial network, a two layer neural network is trained with the Levenberg-Marquardt method. If desired it is possible to use regularization by weight decay. Also pruned (i.e., not fully connected) networks can be trained. The activation functions can be either *linear* or *tanh*. The network architecture is defined by matrix 'NetDef' which has two rows. The first row specifies the hidden layer while the second specifies the output layer.

E.g.:
```
NetDef =  ['LHHHH'
          'LL---']
```
(L = linear, H = tanh)

Notice that the bias in is included as the last column in the weight matrices and that a weight is pruned (i.e., 0 and not updated) by initializing it to 0.

It is possible to train networks with regularization by simple weight decay. The field *D* in *trparms* can be set as forllows: *D* is a vector containing the weight decay parameters. If *D* has one element a scalar weight decay will be used. If *D* has two elements the first element will be used as weight decay for the hidden-to-output layer and while second will be used for the input-to-hidden layer weights. For individual weight decays, *D* must contain as many elements as there are weights in the network.

Example

Generate data as sinusiodal +noise
>> *PHI = 2*pi*rand(1,300);*
>> *Y = sin(PHI) + 0.2*randn(1,300);*
>> *plot(PHI,Y,'+');*

Initialize network. 5 *tanh* hidden units, 1 *linear* output
>> *NetDef = ['HHHHH';'L----'];*
>> *[W1,W2,critvec,iter,lambda]=marq(NetDef,[],[],PHI,Y);*
>> *drawnet(W1,W2)*

Plot criterion evaluated after each iteration
>> *semilogy(critvec); grid;*
>> *xlabel('Iteration');*
>> *ylabel('Criterion')*

Algorithm

The algorithm is a standard Levenberg-Marquardt method as described in the references below. The trust region is adjusted in an indirect fashion by directly increasing/decreasing the diagonal added to the Hessian according to the ratio between actual and predicted change in the criterion.

See Also

MARQLM, RPE, BATBP, INCBP, NNEVAL.

References

R. Fletcher: *"Practical Methods of Optimization,"* Wiley, 1987.
M. Nørgaard, O. Ravn, N. K. Poulsen, L. K. Hansen: *"Neural networks for Modelling and Control of Dynamic Systems,"* Springer-Verlag, London, 2000.

marqlm

Purpose

Implementation of the Levenberg-Marquardt method that uses less memory than MARQ.

Description

A less memory consuming (but slower) version of the Levenberg-Marquardt training algorithm implemented in MARQ. The difference in speed occurs because the function is less "vectorized" (which is a MATLAB problem), but also because some calculations might be repeated.

netstruc

Purpose

Extract weight matrices from parameter vector.

Synopsis

[W1,W2]=netstruc(NetDef,thd,index)

Inputs

NetDef: Architecture definition.
thd: Matrix containing parameter vectors returned by OBDPRUNE, OBSPRUNE or NNPRUNE.
index: Specifies the location in 'thd' where the optimal parameter vector is located.

Outputs

W1, W2: Weight matrices.

Description

NETSTRUC extracts the weight matrices from the matrix of parameter vectors produced by the pruning functions OBDPRUNE, OBSPRUNE and NNPRUNE.

Example

Prune network by OBS
```
>> [thd,tre,fpevec,tee,deff,pvec]=...
        obsprune(NetDef,W1,W2,PHI1,Y1,trparms,[],PHI2,Y2)
```

Find index to minimum FPE
```
>> [minfpe,index] = min(fpevec(pvec));
>> index = pvec(index);
```

Extract weights from matrix of parameter vectors
```
>> [w1,w2] = netstruc(NetDef,thd,index);
>> drawnet(w1,w2,eps)
```

See Also

OBDPRUNE, OBSPRUNE, NNPRUNE.

nnarmax1

Purpose

Identify a Neural Network ARMAX (or ARMA) model (linear MA-filter).

Synopsis

[w1,w2,chat,critvec,iteration,lambda]=...
nnarmax1(NetDef,NN,W1,W2,Chat,trparms,Y,U)

Input

U:	Input (= control signal) (left out in the nnarma case) matrix. Dimension: [(inputs) * (# of data)]
Y:	Output data. Dimension: [1 * (# of data)]
NN:	NN=[na nb nc nk]. na = # of past outputs used for determining the prediction. nb = # of past inputs. nc = # of past residuals (= order of C). nk = time delay (usually 1). For multi-input systems, nb and nk contain as many columns as there are inputs.
W1,W2:	Input-to-hidden layer and hidden-to-output layer weights. dim(W1)= [(# of hidden units) * (na+nb+1)] dim(W2)=[1 * (# of hidden units)] If they are passed as [], they are initialized automatically.
Chat:	Initial MA-filter estimate (initialized automatically if Chat=[]).
trparms:	Data structure containing parameters associated with the training algorithm (optional). Use the function SETTRAIN if you do not want to use the default values.

For time series (NNARMA models) use NN=[na nc].

Ouput

See the function MARQ for an explanation of the returned variables.

Description

Determines a nonlinear ARMAX model of a dynamic system by training a two layer neural network with the Levenberg-Marquardt method. The function can handle multi-input single-output systems (MISO). It is assumed that the noise can be modeled by filtering the residuals with a linear MA-filter:

$$\hat{y}(t|\theta) = g\left(y(t-1),\ldots,y(t-n_a),u(t-n_k),\ldots,u(t-n_b-n_k+1)\right)+C(q^{-1})\varepsilon(t)$$

in which case problems with instability of the predictor are avoided.

Example

```
>>load spmdata
>>NetDef = ['HHHHH';'L----'];
>> NN=[2 2 2 1];
>> trparms = settrain;
>> trparms = settrain(trparms,'maxiter',100,'D',1e-3,'skip',10);
>> [W1,W2,Chat,critvec,iter,lambda] = ...
                    nnarmax1(NetDef,NN,[],[],[],trparms,y1,u1);
>> [yhat,NSSE]=nnvalid('nnarmax1',NetDef,NN,W1,W2,Chat,y2,u2);
```

Algorithm

The name NNARMAX has been chosen because the regressors are the same as those in ARMAX models.

See Also

NNRARMX1, NNARMAX2

Reference

L. Ljung: *"System Identification - Theory for the User,"* Prentice-Hall, 1987.

M. Nørgaard, O. Ravn, N. K. Poulsen, L. K. Hansen: *"Neural networks for Modelling and Control of Dynamic Systems,"* Springer-Verlag, London, 2000.

nnarmax2

Purpose

Identify a Neural Network ARMAX (or ARMA) model.

Synopsis

[w1,w2,critvec,iteration,lambda]=...
nnarmax2(NetDef,NN,W1,W2,trparms,Y,U)

Input

U:	Input (= control signal) (left out in the nnarma case) matrix. Dimension: [(inputs) * (# of data)]
Y:	Output data. Dimension: [1 * (# of data)]
NN:	NN=[na nb nc nk]. na = # of past outputs used for determining the prediction. nb = # of past inputs. nc = # of past residuals (= order of C). nk = time delay (usually 1). For multi-input systems, nb and nk contain as many columns as there are inputs.
W1,W2:	Input-to-hidden layer and hidden-to-output layer weights. dim(W1)= [(# of hidden units) * (na+nb+1)] dim(W2)=[1 * (# of hidden units)] If they are passed as [], they are initialized automatically.
trparms:	Data structure containing parameters associated with the training algorithm (optional). Use the function SETTRAIN if you do not want to use the default values.

For time series (NNARMA models) use NN=[na nc].

Ouput

See the function MARQ for an explanation of the returned variables.

Description

Determines a nonlinear ARMAX model:

$$\hat{y}(t|\theta) = g\big(y(t-1),\ldots,y(t-n_a),u(t-n_k),\ldots,u(t-n_b-n_k+1),\varepsilon(t-1),\ldots,\varepsilon(t-n_c)\big)$$

of a dynamic system by training a two layer neural network with the Levenberg-Marquardt method. The function can handle multi-input single-output systems (MISO).

Example

```
>> load spmdata
>> NetDef = ['HHHHH';'L----'];
>> NN=[2 2 2 1];
>> trparms = settrain;
>> trparms = settrain(trparms,'maxiter',100,'D',1e-3,'skip',10);
>> [W1,W2,critvec,iter,lambda] = ...
                    nnarmax2(NetDef,NN,[],[],trparms,y1,u1);
>> [yhat,NSSE] = nnvalid('nnarmax2',NetDef,NN,W1,W2,y2,u2);
```

Algorithm

The name NNARMAX has been chosen because the regressors are the same as those in ARMAX models.

See Also

NNRARMX2, NNARMAX1

Reference

L. Ljung: *“System Identification - Theory for the User,”* Prentice-Hall, 1987.

M. Nørgaard, O. Ravn, N. K. Poulsen, L. K. Hansen: *”Neural networks for Modelling and Control of Dynamic Systems,”* Springer-Verlag, London, 2000.

nnarx

Purpose

Identify a Neural Network ARX (or AR) model.

Synopsis

[w1,w2,critvec,iteration,lambda]=nnarx(NetDef,NN,W1,W2,trparms,Y,U)

Input

U:	Input (= control signal) (left out in the nnar case) matrix. Dimension: [(inputs) * (# of data)]
Y:	Output data. Dimension: [1 * (# of data)]
NN:	NN=[na nb nk]. na = # of past outputs used for determining the prediction. nb = # of past inputs. nk = time delay (usually 1). For multi-input systems, nb and nk contain as many columns as there are inputs.
W1,W2:	Input-to-hidden layer and hidden-to-output layer weights. dim(W1)= [(# of hidden units) * (na+nb+1)] dim(W2)=[1 * (# of hidden units)] If they are passed as [], they are initialized automatically.
trparms:	Data structure containing parameters associated with the training algorithm (optional). Use the function SETTRAIN if you do not want to use the default values.

For time series (NNAR models) use NN=na.

Ouput

See the function MARQ for an explanation of the returned variables.

Description

Determines a nonlinear ARX model:

$$\hat{y}(t|\theta) = g\big(y(t-1),\ldots, y(t-n_a), u(t-n_k),\ldots, u(t-n_b-n_k+1)\big)$$

of a dynamic system by training a two layer neural network with the Levenberg-Marquardt method. The function can handle multi-input single-output systems (MISO).

Examples

>> *load spmdata*

```
>> NetDef = ['HHHH';'L---'];
>> NN=[2 2 1];
>> trparms = settrain;
>> trparms = settrain(trparms,'maxiter',100,'D',1e-3);
>> [W1,W2,critvec,iter,lambda]=nnarx(NetDef,NN,[],[],trparms,y1,u1);
>> [yhat,NSSE]=nnvalid('nnarx',NetDef,NN,W1,W2,y2,u2);
```

Algorithm

The name NNARX has been chosen because the regressors are the same as those in ARX models.

See Also

NNRARX, NNARXM, NNPRUNE.

Reference

L. Ljung: *"System Identification - Theory for the User,"* Prentice-Hall, 1987.

M. Nørgaard, O. Ravn, N. K. Poulsen, L. K. Hansen: *"Neural networks for Modelling and Control of Dynamic Systems,"* Springer-Verlag, London, 2000.

nnarxm

Purpose

Identify a multi-output Neural Network ARX (or AR) model.

Synopsis

[W1,W2,critvec,iteration,lambda]=...
nnarxm(NetDef,NN,W1,W2,trparms,Gamma,Y,U)

Input

U:	Input (= control signal) (left out in the nnar case) matrix. Dimension: [(inputs) * (# of data)]
Y:	Output data. Dimension: [(outputs) * (# of data)]
NN:	NN=[na1 nb1 nk1;na2 nb2 nk2;...]. naX = # of past outputs used for determining the prediction. nbX = # of past inputs. nkX = time delay (usually 1). For multi-input systems, nbX and nkX contain as many columns as there are inputs.
W1,W2:	Input-to-hidden layer and hidden-to-output layer weights. dim(W1)= [(# of hidden units) * (na1+nb1+na2+nb2+...+1)] dim(W2)=[(outputs) * (# of hidden units)] If they are passed as [], they are initialized automatically.
Gamma:	Inverse weighting matrix (usually the covariance of the noise).
trparms:	Data structure containing parameters associated with the training algorithm (optional). Use the function SETTRAIN if you do not want to use the default values.

For time series (NNAR models) use NN=[na1;na2; …].

Ouput

See the function MARQ for an explanation of the returned variables.

Description

Determines a nonlinear ARX model:

$$\hat{y}(t|\theta) = g\big(y(t-1),\ldots,y(t-n_a),u(t-n_k),\ldots,u(t-n_b-n_k+1)\big)$$

of a dynamic system with multiple outputs by training a two layer neural network with the Levenberg-Marquardt method. The function can handle multi-input, multi-output systems (MIMO).

Examples

```
>> load spmdata
>>Y1=[y1;y1*3];
>>Y2=[y2;y2*3];
>> NetDef = ['HHHH';'LL--'];
>> NN=[2 2 1;2 0 0];
>> trparms = settrain;
>> trparms = settrain(trparms,'maxiter',100,'D',1e-3);
>> [W1,W2]=nnarxm(NetDef,NN,[],[],trparms,[],Y1,u1);
>> [yhat,NSSE]=nnvalid('nnarxm',NetDef,NN,W1,W2,eye(2),Y2,u2);
```

In this example *NN=[2 2 1;2 0 0]*. This does not mean that output # 2 does not depend on past inputs at all. If *NN* had been chosen to *[2 2 1;2 2 1]* the input signal would then have entered the network twice. This is of course not relevant except when physical knowledge motivates that an output depends on certain inputs and delayed inputs and it should only be used when appropriate entries in W1 and W2 are set to 0.

Algorithm

The network is trained to minimize the criterion

$$\hat{y}(t|\theta) = g\left(y(t-1), \ldots, y(t-n_a), u(t-n_k), \ldots, u(t-n_b-n_k+1)\right)$$

using a Levenberg-Marquardt algorithm. The weighting matrix *Gamma* is usually selected as the noise covariance. This matrix can be estimated with the function NNIGLS.

See Also

NNVALID, NNIGLS, NNARX.

Reference

L. Ljung: *“System Identification - Theory for the User,”* Prentice-Hall, 1987.

M. Nørgaard, O. Ravn, N. K. Poulsen, L. K. Hansen: *”Neural networks for Modelling and Control of Dynamic Systems,”* Springer-Verlag, London, 2000.

nneval

Purpose

Validation of feedforward neural networks.

Synopsis

[Yhat,E,NSSE] = nneval(NetDef,W1,W2,PHI,Y)

Inputs

See for example one of the functions: MARQ, RPE, BATBP, INCBP.

Outputs

Yhat:	Network predictions.
E:	Prediction errors.
NSSE:	Normalized sum of squared errors (SSE/2N).

Description

The function validates models trained with MARQ, RPE, BATBP, INCBP, MARQLM. The following plots are produced:

- Output together with predicted output.
- Prediction error.
- Autocorrelation function of prediction error.
- A histogram showing the distribution of the prediction errors

Example

```
>> PHI = 2*pi*rand(1,300);
>> Y = sin(PHI) + 0.2*randn(1,300);
>> NetDef = ['HHHHH';'L----'];
>> [W1,W2,critvec,iter,lambda]=marq(NetDef,[],[],PHI,Y);
>> PHI2 = 2*pi*rand(1,300);
>> Y2 = sin(PHI2) + 0.2*randn(1,length(PHI2);
>> nneval(NetDef,W1,W2,PHI2,Y2);
```

See Also

NNVALID, IFVALID, IOLEVAL.

nnfpe

Purpose

Final Prediction Error estimate (FPE) for I/O models of dynamic systems.

Synopsis

[FPE,deff,varest,H] =...
nnfpe(method,NetDef,W1,W2,U,Y,NN,trparms,Chat)

Input

See the function that was used for creating the model.The argument *Chat* should only be included if *method='nnarmax1'*.

Output

FPE: The Final prediction error estimate.
deff: The effective number of parameters.
varest: Estimate of noise variance.
H: The Gauss-Newton Hessian.

Description

The function calculates Akaike's final prediction error estimate of the average generalization error for models generated by NNARX, NNOE, NNARMAX1+2. The function produces the final prediction error estimate (FPE), the effective number of weights in the network if the network has been trained with weight decay, an estimate of the noise variance, and the Gauss-Newton Hessian.

See Also

LOO, FPE.

References

J. Larsen & L.K. Hansen: *"Generalization Performance of Regularized Neural Network Models."* Proc. of the IEEE Workshop on Neural networks for Signal Proc. IV, Piscataway, New Jersey, pp.42-51, 1994.

nnigls

Purpose

Iterated Generalized Least Squares training of NNARX models with multiple outputs.

Synopsis

[w1,w2,Gamma,lambda]=...
nnigls(NetDef,NN,W1,W2,trparms,Gamma0,Y,U)

Input

U,Y,NN,W1,W2,trparms:	See NNARXM
Gamma0:	Initial estimate of the covariance matrix for the noise. If passed as [] it is set to the identity matrix.

Output

w1, w2, lambda:	See the function NNARXM.
Gamma:	The estimated covariance matrix.

Description

A multi-output NNARX model and the noise covariance matrix are estimated with an iterative relaxation procedure.

It is important to notice that the model returned from this function will produce predictions of scaled outputs (see the *Algorithm* paragraph). It is necessary to multiply the output by *sqrtm(Gamma)* to obtain the unscaled predictions. If the network has linear output units one can instead scale the hidden-to-output layer weights: *W2= sqrtm(Gamma)*W2*.

Algorithm

The IGLS procedure is a straightforward relaxation procedure:

for j=1:repeat,
Train the network
Estimate the covariance matrix
end

The factor *repeat* is a field in *trparms*. Its default value is 5 but can be changed with SETTRAIN. It is recommended to use SETTRAIN to reduce the max. number of iterations performed by the training algorithm. 50 is a value that generally will work in well.

The network is trained with the function MARQ according to the criterion

$$\hat{\theta}_j = V_N(\theta, Z^N) = \frac{1}{2N}\sum_{t=1}^{N}\left(y(t) - \hat{y}(t|\theta)\right)^T \hat{\Lambda}_{j-1}^{-1}\left(y(t) - \hat{y}(t|\theta)\right)$$

$$= \frac{1}{2N}\sum_{t=1}^{N}\varepsilon^T(t,\theta)\hat{\Lambda}_{j-1}^{-1}\varepsilon(t,\theta)$$

and the covariance matrix is estimated as

$$\hat{\Lambda}_j = \frac{1}{N}\sum_{t=1}^{N}\varepsilon(t,\hat{\theta}^{(j)})\varepsilon^T(t,\hat{\theta}^{(j)})$$

To reduce the amount of computations the network is trained by first scaling the outputs as

$$\bar{y}(t) = \Sigma y(t)$$

where

$$\Lambda = \Sigma^T\Sigma$$

and subsequently train the network according to

$$\hat{\theta}_j = V_N(\theta, Z^N) = \frac{1}{2N}\sum_{t=1}^{N}\left(\bar{y}(t) - \hat{y}(t|\theta)\right)^T\left(\bar{y}(t) - \hat{y}(t|\theta)\right)$$

If the network has linear output units, W2 should be scaled by $W2u = \Sigma^{-1}W2$.

See Also

NNARXM, NNVALID, MARQ, IGLS.

Reference

T.J Fog, J. Larsen, L.K. Hansen: *Training and Evaluation of Neural Networks for Multi-Variate Time-Series Processing.* Proc. IEEE International Conference on Neural Networks, Perth, Australia.

nniol

Purpose

Identify a neural network model well-suited for control by discrete input-output linearization.

Synopsis

[w1f,w2f,w1g,w2g,critvec,iteration,lambda]=...
nniol(NetDeff,NetDefg,NN,W1f,W2f,W1g,W2g,trparms,Y,U)

Input

U:	Input data (= control signal) (left out in the nnarma case) matrix. Dimension: [(inputs) * (# of data)]
Y:	Output data. Dimension: [1 * (# of data)]
NN:	NN=[na nb nk]. na = # of past outputs used for determining the prediction. nb = # of past inputs. nk = time delay (usually 1). For multi-input systems, nb and nk contain as many columns as there are inputs.
NetDeff:	Architecture of network used for modelling the function *f* (see below).
NetDefg:	Archtecture of network used for modelling the function *g*.
W1f,W2f: W1g,W2g	Input-to-hidden layer and hidden-to-output layer weights for the ”f” and ”g” nets, respectively. dim(W1f / W1g) = [(# of hidden units) * (na+nb)] dim(W2f / W2g) = [1 * (# of hidden units)] If the weight matrices are passed as [] they will be initialized automatically.
trparms:	Data structure containing parameters associated with the training algorithm (optional). Use the function SETTRAIN if you do not want to use the default values.

Ouput

See the function MARQ for an explanation of the returned variables.

Description

Train a neural network to model a dynamic system on the following form:

$$\hat{y}(t|\theta) = f\left(y(t-1),\dots,y(t-n_a),u(t-n_k-1),\dots,u(t-n_k-n_b+1)\right)$$
$$+\ g\left(y(t-1),\dots,y(t-n_a),u(t-n_k-1),\dots,u(t-n_k-n_b+1)\right)u(t-n_k)$$

with the Levenberg-Marquardt method. This type of model is particularly relevant in the context of control by discrete input-output linearization.

Examples

```
>> load spmdata
>> NetDeff = ['HHHHH';'L----'];
>> NetDefg = ['HHH';'L--'];
>> NN=[2 2 1];
>> trparms = settrain;
>> [W1f,W2f,W1g,W2g,critvec,iter,lambda] =...
                    nniol(NetDeff,NetDefg,NN,[],[],[],[],trparms,y1,u1);
>> [yhat,NSSE]=ioleval(NetDeff,NetDefg,NN,W1f,W2f,W1g,W2g,y2,u2);
```

See Also

IOLEVAL.

Reference

M. Nørgaard, O. Ravn, N. K. Poulsen, L. K. Hansen: *”Neural networks for Modelling and Control of Dynamic Systems,”* Springer-Verlag, London, 2000.

nnloo

Purpose

Estimate the average generalization error for NNARX models of dynamic systems by using leave-one-out cross-validaton.

Synopsis

Eloo =nnloo(NetDef,NN,W1,W2,trparms,U,Y)

Input

NetDef, W1, W2, NN
U, Y, trparms : See the function NNARX
If the *maxiter* field in the data structure *trparms* is *0* linear unlearning is used for obtaining a cheap approximation to the LOO estimate. If *maxiter>0* the network will be retrained a maximum of *maxiter* iterations for each input-output pair that is left out.

Output

Eloo: The leave-one-out cross-validation estimate of the average generalization error

Description

LOO calculates an approximation of the leave-one-out estimate of the average generalization error.

Algorithm

When the *maxiter* field in *trparms* is *0* “linear unlearning” is used to get a quick approximation to the LOO-estimate. This approximation is much easier to compute than the true LOO-estimate, but is in general less reliable. Typically it is comparable to the FPE-estimate. See the reference below for a derivation. Unless *maxiter*=0 it is recommended to set *maxiter* to 20-40.

See Also

NNFPE for Akaike’s final prediction error estimate.

Reference

L.K. Hansen and J. Larsen: *"Linear Unlearning for Cross-Validation,"* Advances in Computational Mathematics, 5, pp. 269-280, 1996.

nnoe

Purpose

Identify a neural network output error model.

Synopsis

[W1,W2,critvec,iter,lambda]=nnoe(NetDef,NN,W1,W2,trparms,Y,U)

Input

U: Input data (= control signal) (left out in the nnarma case) matrix. Dimension: [(inputs) * (# of data)]

Y: Output data. Dimension: [1 * (# of data)]

NN: NN=[na nb nk].
na = # of past predictions used for determining the prediction.
nb = # of past inputs.
nk = time delay (usually 1).
For multi-input systems, nb and nk contain as many columns as there are inputs.

W1,W2: Input-to-hidden layer and hidden-to-output layer weights.
dim(W1)= [(# of hidden units) * (na+nb+1)]
dim(W2)=[1 * (# of hidden units)]
If they are passed as [], they are initialized automatically.

trparms: Data structure containing parameters associated with the training algorithm (optional). Use the function SETTRAIN if you do not want to use the default values.

Ouput

See the function MARQ for an explanation of the returned variables.

Description

Determines a nonlinear output error (OE) model:

$$\hat{y}(t|\theta) = g\left(\hat{y}(t-1|\theta), \ldots, \hat{y}(t-n_a|\theta), u(t-n_k), \ldots, u(t-n_b-n_k+1)\right)$$

of a dynamic system by training a two layer neural network with the Levenberg-Marquardt method. The function can handle multi-input single-output systems (MISO).

Example

```
>> load spmdata
>> NetDef = ['HHHH';'L---'];
>> NN=[2 2 1];
>> trparms=settrain;
>> trparms=settrain(trparms,'maxiter',100,'D',1e-3,'skip',10);
>> [W1,W2,critvec,iter,lambda]=nnoe(NetDef,NN,[],[],trparms,y1,u1);
>> [yhat,NSSE]=nnvalid('nnoe',NetDef,NN,W1,W2,y2,u2);
```

Algorithm

The name NNOE is used because the regressors are the same as those used in output error (OE) models.

See Also

NNPRUNE, NNVALID.

Reference

L. Ljung:*"System Identification - Theory for the User,"* Prentice-Hall, 1987.

M. Nørgaard, O. Ravn, N. K. Poulsen, L. K. Hansen: *"Neural networks for Modelling and Control of Dynamic Systems,"* Springer-Verlag, London, 2000.

nnprune

Purpose

Prune a neural network model of a dynamic systems with the Optimal Brain Surgeon algorithm (OBS).

Synopsis

[thd,NSSEvec,FPEvec,NSSEtestvec,deff_vec,pvec] = ...
nnprune(method,NetDef,W1,W2,U,Y,NN,trparms,prparms,U2,Y2,Chat)

Input

method:	The function used for creating the model. For example method='nnarx' or method='nnoe'.
NetDef, W1, W2, U, Y, trparms:	See the function used for creating the model.
U2,Y2 (optional):	Test data. This can be used for pointing out the the optimal network architecture. Pass two []'s if a test set is not available.
Chat (optional):	See NNARMAX1
prparms:	Parameters associated with the pruning session. prparms = [iter RePercent] iter: Max. number of retraining iterations. RePercent : Prune 'RePercent' percent of the remaining weights (0 = prune one weight at a time). If passed as [] is will be reset to prparms = [50 0].

Output

thd:	Matrix containing all the parameter vectors.
NSSEvec:	Vector containing the normalized sum of squared errors (SSE/2N), the *training error*, after each weight elimination.
FPEvec:	Contains the FPE estimate of the average generalization error.
NSSEtestvec:	Contains the test error (SSE/2N for test set).
deff_vec:	Contains the ”effective” number of weights.
pvec:	Index into the above vectors.

Description

This function applies the Optimal Brain Surgeon (OBS) strategy for pruning neural network input-output models of dynamic systems. That is, models produced by one of the functions: NNARX, NNARMAX1, NNARMAX2, NNOE. Two different procedures are possible:

- Eliminate one weight, retrain, eliminate one weight, retrain, ...

- Eliminate 5% (or some other percentage) of the remaining weights, retrain, eliminate 5% of the remaning weights,retrain, ...

The function will return a matrix containing the parameter vectors (a vector containing all weights), obtained after each retraining. The optimal parameter vector is then chosen afterwards. For example as the one representing the network leading to the smallest FPE or the one leading to the smallest test error (if a test set is available). After having determined the optimal number of weights, the weight matrices are extracted from the thd-matrix with the function NETSTRUC. If a NNARMAX1 model has been pruned, remember to remove the bottom nc rows from thd first since these contain the coefficients of the C-polynomial.

It is important that the network is trained to the minimum of the criterion before the function is applied.

Example

Prune nnarx model with OBS
\>> *[thd,NSSEvec,FPEvec,NSSEtestvec,deff_vec,pvec] = ...*
nnprune('nnarx',NetDef,W1,W2,U,Y,NN,trparms,[],U2,Y2);

Find index to minimum FPE
\>> *[minfpe,index] = min(fpevec(pvec));*
\>> *index = pvec(index);*

Extract weights from matrix of parameter vectors
\>> *[W1,W2] = netstruc(NetDef,thd,index);*
\>> *drawnet(W1,W2,eps)*

Algorithm

If the network has been trained without regularization (weight decay), the basic OBS scheme of Hassibi and Stork is used. To avoid numerical problems, the inverse (Gauss-Newton) Hessian is approximated by the recursive method described in the paper (see also RPE). If regularization was used when training the network the saliences are calculated as the prediced increase in the unregularized portion of the criterion as described by Hansen & Pedersen. If more than one weight is eliminated between each retraining the inverse Hessian after each weight elimination is calculated as the Schur complement of the previous inverse Hessian (see Pedersen et al.).

The OBS-scheme has been implemented so that it is impossible to have hidden units without having weights leading to as well as from them. If a hidden unit has only one weight connecting it to the input layer and one weight connecting it to the output layer the, the entire unit will be removed if it has the smallest total saliency.

See Also

NETSTRUC, OBDPRUNE, OBSPRUNE.

References

L.K. Hansen & M. W. Pedersen: *"Controlled Growth of Cascade Correlation Nets,"* Proc. ICANN '94, Sorrento, Italy, 1994, Eds. M. Marinaro & P.G. Morasso, pp. 797-800.

B. Hassibi, D.G. Stork: *"Second Order Derivatives for Network Pruning: Optimal Brain Surgeon,"* NIPS 5, Eds. S.J. Hanson et al., 164, San Mateo, Morgan Kaufmann, 1993.

M.W. Pedersen, L.K. Hansen, J. Larsen: *"Pruning With Generalization Based Weight Saliences: γOBD, γOBS,"* 1995.

nnrarmx1, nnrarmx2, nnrarx

Purpose

Identify a neural network model of a dynamic system by using a recursive algorithm.

Synopsis

[w1,w2,chat,critvec,iteration]=...
nnrarmx1(NetDef,NN,W1,W2,Chat,trparms,Y,U)

[w1,w2,critvec,iteration]=...
nnrarmx2(NetDef,NN,W1,W2,trparms,Y,U)

[w1,w2,critvec,iteration,lambda]=...
nnrarx(NetDef,NN,W1,W2,trparms,Y,U)

Input

See the “batch” counterparts (NNARMAX1, NNARMAX2, NNARX). The *method* field in *trparms* is particularly important here. It selects one of three different recursive training schemes. The default method is the exponential forgetting factor algorithm. See SETTRAIN for details.

Output

See their batch counterparts.

Description

The three functions are the recursive counterparts to NNARMAX1, NNARMAX2, and NNARX, respectively. The networks are trained with a recursive Gauss-Newton based method (see RPE) instead of a batch method. Most often the disadvantages of a recursive method are overwhelming compared to a batch method, but they can be useful for very large networks+data sets since lack of memory in this case can be a problem. They can also be advantageous compared to batch training when there is high degree of redundancy in the data set.

Example

\>> *load spmdata*
\>> *NetDef = ['HHHHH'; 'L----'];*
\>> *NN=[2 2 1];*
\>> *trparms=settrain;*

>> trparms=settrain(trparms,'maxiter',100,'p0',1e3);
>> [W1,W2,critvec,iter]=nnrarx(NetDef,NN,[],[],trparms,y1,u1);
>> [yhat,NSSE]=nnvalid('nnrarx',NetDef,NN,W1,W2,y2,u2);

Algorithm

Be careful not to use a forgetting factor which is too small when using the forgetting factor method. Because of the many weights usually present in the network, some eigenvalues in the covariance matrix ("the inverse Hessian") will grow uncontrollably.

See Also

NNARMAX1, NNARMAX2, NNARX.

References

L. Ljung: *"System Identification - Theory for the User,"* Prentice-Hall, 1987.

J.E. Parkum: *"Recursive Identification of Time-Varying Systems,"* Ph.D. thesis, IMM, Technical University of Denmark, 1992.

M.E. Salgado, G. Goodwin, R.H. Middleton: *"Modified Least Squares Algorithm Incorporating Exponential Forgetting And Resetting,"* Int. J. Control, 47, pp. 477-491.

M. Nørgaard, O. Ravn, N. K. Poulsen, L. K. Hansen: *"Neural networks for Modelling and Control of Dynamic Systems,"* Springer-Verlag, London, 2000.

nnsimul

Purpose

Simulate the response of model of dynamic system to a sequence of control inputs.

Synopsis

Network generated by NNARX (or NNRARX):
Ysim = nnsimul('nnarx',NetDef,NN,W1,W2,Y,U);

(likewise for networks generated by NNARMAX1+2 and NNOE)

Network generated by NNSSIF:
Ysim = nnsimul('nnssif',NetDef,nx,W1,W2,Y,U,obsidx);

Input

See nnvalid/ifvalid.

Output

Ysim: Vector containing simulated outputs.

NB! The function does not work for models generated by NNIOL.

Description

Simulate how a neural network model of a dynamic system responds to a specific sequence of control inputs alone. The simulated output is compared to the observed output. For NNARMAX1+2 models the initial unknown residuals are assumed to be 0.

Examples

```
>> load spmdata
>> NetDef = ['HHHH';'L---'];
>> NN=[2 2 1];
>> trparms=settrain;
>> trparms=settrain(trparms,'maxiter',300,'D',1e-3);
>> [W1,W2,critvec,iter,lambda]=nnarx(NetDef,NN,[],[],trparms,y1,u1);
>> ysim=nnsimul('nnarx',NetDef,NN,W1,W2,y1,u1);
```

nnssif

Purpose

Identify a neural network model in state space innovations form.

Synopsis

[w1,w2,obsidx,critvec,iteration,lambda]=...
nnssif(NetDef,nx,W1,W2,obsidx,trparms,Y,U)

Inputs:

U: Input data (= control signal). dim(U)=[(# of inputs) * (# of data)]
Y: Output data. dim(Y)=[1 * (# of data)]
nx: # of states (= the order of the system)
W1,W2: Input-to-hidden layer and hidden-to-output layer weights.
dim(W1)= [(# of hidden units) * (nx+inputs+outputs+1)]
dim(W2)=[nx * (# of hidden units+1)]
If they are passed as [] they are initialized automatically.
obsidx: Pseudo-observability indices. Their sum must equal nx!
If passed as [] a particular set of indices is selected.
trparms: Data structure containing parameters associated with the training algorithm (optional). Use the function SETTRAIN if you do not want to use the default values.

Description

Determines a nonlinear state space model of a dynamic system:

$$\hat{x}(t+1,\theta) = g\big(\hat{x}(t,\theta), u(t-1), \varepsilon(t|\theta)\big)$$

$$\hat{y}(t) = C\hat{x}(t,\theta)$$

The neural network is trained with the Levenberg-Marquardt method. The function can handle multi-input multi-output systems (MIMO).

The function does not work for time series.

Examples

```
>> load spmdata
>> NetDef = ['HHHH';'LL--'];
>> trparms=settrain;
>> trparms=settrain(trparms,'maxiter',300,'D',1e-3,'skip',10);
>> [W1,W2,obsidx,critvec,iter,lambda] =...
                    nnssif(NetDef,2,[],[],[],trparms,y1,u1);
```

>> *[yhat,NSSE]=ifvalid(NetDef,2,W1,W2,obsidx,y2,u2);*

Algorithm

The name NNSSIF has been chosen because the regressors equal those of a linear state space innovations form (the Kalman filter).

See Ljung (1987) for an explanation of overlapping parametrizations, and for a definition of the pseudo-observability indices.

See Also

IFVALID.

Reference

L. Ljung: *"System Identification - Theory for the User,"* Prentice-Hall, 1987.

M. Nørgaard, O. Ravn, N. K. Poulsen, L. K. Hansen: *"Neural networks for Modelling and Control of Dynamic Systems,"* Springer-Verlag, London, 2000.

nnvalid

Purpose

Validate neural network input-output models of dynamic systems.

Synopsis

Network generated by NNARX (or NNRARX):
[Yhat,NSSE] = nnvalid('nnarx',NetDef,NN,W1,W2,Y,U)

Network generated by NNARMAX1 (or NNRARMX1):
[Yhat,NSSE] = nnvalid('nnarmax1',NetDef,NN,W1,W2,C,Y,U)

Network generated by NNARMAX2 (or NNRARMX2):
[Yhat,NSSE] = nnvalid('nnarmax2',NetDef,NN,W1,W2,Y,U)

Network generated by NNOE:
[Yhat,NSSE] = nnvalid('nnoe',NetDef,NN,W1,W2,Y,U)

Network generated by NNARXM:
[Yhat,NSSE] = nnvalid('nnarxm',NetDef,NN,W1,W2,Gamma,Y,U)

Input

See the function used for generating the model.
For time series the argument U is simply left out.

Output

Yhat:	Network predictions.
NSSE:	Normalized sum of squared errors.

Description

The function validate models that have been generated by one of the functions NNARX(M), NNRARX, NNARMAX1+2, NNRARMX1+2, or NNOE.

The following plots are produced:
- Observed output together with predicted output.
- Prediction error.
- Auto correlation function of prediction error and cross-correlation between prediction error and input.
- A histogram showing the distribution of the prediction errors.
- Coefficients of extracted linear models.

Example

```
>> load spmdata
>> NetDef = ['HHHH';'L---'];
>> NN=[2 2 1];
>> [W1,W2,critvec,iter,lambda]=nnarx(NetDef,NN,[],[],[],y1,u1);
>> [yhat,NSSE]=nnvalid('nnarx',NetDef,NN,W1,W2,y2,u2);
```

obdprune

Purpose

Prune ordinary feedforward networks with Optimal Brain Damage (OBD).

Synopsis

[thd,NSSEvec,FPEvec,NSSEtestvec,deff_vec,pvec]=...
obdprune(NetDef,W1,W2,PHI,Y,trparms,prparms,PHI2,Y2)

Input

NetDef, W1, W2,	
PHI, Y, trparms:	See for example the function MARQ.
PHI2,Y2 (optional):	Test data. This can be used as an indicator for pointing out the optimal network architecture.
prparms:	Parameters associated with the pruning session. prparms = [iter RePercent] iter: Max. number of retraining iterations. RePercent : Prune 'RePercent' percent of the remaining weights (0 = prune one weight at a time). If passed as [] prparms will be set [50 0].

Output

thd:	Matrix containing all the parameter vectors
NSSEvec:	Vector containing normalized sum of squared errors (SSE/2N), the *training error*, after each weight elimination.
FPEvec:	Contains the FPE estimate of the average generalization error
NSSEtestvec:	Contains the test error (SSE/2N for the test set).
deff_vec:	Contains the “effective” number of weights.
pvec:	Index into the above vectors.

Description

This function applies the Optimal Brain Damage (OBD) strategy for pruning feed-forward neural networks. Two different prucedures are possible:

- Eliminate one weight, retrain, eliminate one weight, retrain, ...
- Eliminate 5% (or some other percentage) of the remaining weights, retrain, eliminate 5% of the remaning weights,retrain, ...

The retraining is done with the Levenberg-Marquardt method in MARQ.

The function will return a matrix containing the parameter vectors (a vector containing all weights), obtained after each retraining. The optimal parameter vector is then chosen afterwards. For example as the one representing the network leading to the smallest FPE or the one leading to the smallest test error (if a test set is available). After having determined the optimal number of weights, the weight matrices are extracted from the thd-matrix with the function NETSTRUC.

Example

Prune network with OBD

```
>> [thd,tre,fpevec,tee,deff,pvec]=...
         obdprune(NetDef,W1,W2,PHI1,Y1,trparms,[50 5],PHI2,Y2)
```

Find index to minimum FPE

```
>> [minfpe,index] = min(fpevec(pvec));
>> index = pvec(index);
```

Extract weights from matrix of parameter vectors

```
>> [W1,W2] = netstruc(NetDef,thd,index);
>> drawnet(W1,W2,eps)
```

See Also

NETSTRUC, OBSPRUNE, NNPRUNE.

References

Y. Le Cun, J:S. Denker, S.A Solla: *"Optimal Brain Damage,"* Advances in Neural Information Processing Systems, Denver 1989, ed. D. Touretzsky, Morgan Kaufmann, pp. 598-605.

C. Svarer, L.K. Hansen, J. Larsen: *"On Design and evaluation of Tapped-Delay Neural Network Architectures,"* The 1993 IEEE Int. Conf. on Neural networks, San Francisco, Eds. H.R. Berenji et al., pp. 45-51.

obsprune

Purpose

Prune ordinary feedforward networks with Optimal Brain Surgeon (OBS).

Synopsis

[thd,NSSEvec,FPEvec,NSSEtestvec,deff_vec,pvec]=...
obsprune(NetDef,W1,W2,PHI,Y,trparms,prparms,PHI2,Y2)

Input

NetDef, W1, W2, PHI, Y, trparms:	See for example the function MARQ.
PHI2,Y2 (optional):	Test data. This can be used as an indicator for pointing out the optimal network architecture.
prparms:	Parameters associated with the pruning session. prparms = [iter RePercent] iter: Max. number of retraining iterations. RePercent : Prune 'RePercent' percent of the remaining weights (0 = prune one weight at a time). If passed as [] prparms will be set to [50 0].

Output

thd:	Matrix containing all the parameter vectors
NSSEvec:	Vector containing normalized sum of squared errors (SSE/2N), the *training error*, after each weight elimination.
FPEvec:	Contains the FPE estimate of the average generalization error
NSSEtestvec:	Contains the test error (SSE/2N for the test set).
deff_vec:	Contains the "effective" number of weights.
pvec:	Index into the above vectors.

Description

This function applies the Optimal Brain Surgeon (OBS) strategy for pruning feed forward neural networks. Two different procedures are possible:

- Eliminate one weight, retrain, eliminate one weight, retrain, ...
- Eliminate 5% (or some other percentage) of the remaining weights, retrain, eliminate 5% of the remaning weights,retrain, ...

The retraining is done with the Levenberg-Marquardt method in MARQ.

The function will return a matrix containing the parameter vectors (a vector containing all weights) obtained after each retraining. The optimal parameter vector is then chosen afterwards. For example as the one representing the network leading to the smallest FPE or the one leading to the smallest test error (if a test set is available). After having determined the optimal number of weights, the weight matrices are extracted from the thd-matrix with the function NETSTRUC.

Examples

Prune network with OBS
>> *[thd,tre,fpevec,tee,deff,pvec]=...*
obsprune(NetDef,W1,W2,PHI1,Y1,trparms,[50 5],PHI2,Y2)

Find index to minimum FPE
>> *[minfpe,index] = min(fpevec(pvec));*
>> *index = pvec(index);*

Extract weights from matrix of parameter vectors
>> *[W1,W2] = netstruc(NetDef,thd,index);*
>> *drawnet(W1,W2,eps)*

Algorithm

If the network has been trained without regularization (weight decay), the basic OBS scheme of Hassibi and Stork is used. To avoid numerical problems, the inverse (Gauss-Newton) Hessian is approximated by the recursive method described in the paper (see also RPE). If regularization was used when training the network, the saliences are calculated as the prediced increase in the training error as described by Hansen & Pedersen. If more than one weight is eliminated between each retraining the inverse Hessian is calculated after each weight elimination as the Schur complement of the previous inverse Hessian (see Pedersen et al.).

The OBS-scheme has been implemented so that it is impossible to have hidden units without weights leading to as well as from them. If a hidden unit has only one weight connecting it to the input or one weight connecting it to the output layer, the saliency for removing the entire unit is calculated. If the entire unit-saliency is smaller than any of the other saliencies, the entire unit will be removed.

See Also

NETSTRUC, OBDPRUNE, NNPRUNE.

References

L.K. Hansen & M. W. Pedersen: *"Controlled Growth of Cascade Correlation Nets,"* Proc. ICANN '94, Sorrento, Italy, 1994, Eds. M. Marinaro & P.G. Morasso, pp. 797-800.

B. Hassibi, D.G. Stork: *"Second Order Derivatives for Network Pruning: Optimal Brain Surgeon,"* NIPS 5, Eds. S.J. Hanson et al., 164, San Mateo, Morgan Kaufmann, 1993.

M.W. Pedersen, L.K. Hansen, J. Larsen: *"Pruning With Generalization Based Weight Saliences: γOBD, γOBS,"* 1995.

pmntanh

Purpose

Fast hyperbolic tangent function.

Synopsis

y=pmntanh(x)

Description

The function replaces the TANH function provided by MATLAB to increase speed. This is particularly relevant for older versions of MATLAB where the implementation of *tanh* is relatively slow.

rpe

Purpose

Recursive prediction error method.

Synopsis

[w1,w2,critvec,iter]=rpe(NetDef,W1,W2,PHI,Y,trparms)

Input

NetDef:	Network definition.
W1:	Input-to-hidden layer weights dim(W1)=[(# of hidden units) * (inputs + 1)] (1 is due to the bias) Use [] for a random initialization.
W2:	Hidden-to-output layer weights dim(W2)=[(outputs) * (# of hidden units + 1)] Use [] for a random initialization.
PHI:	Input data. dim(PHI)=[(inputs) * (# of data)]
Y:	Output data. dim(Y)=[(outputs) * (# of data)]
trparms:	Data structure containing parameters associated with the training algorithm (optional). Use the function SETTRAIN if you do not want to use the default values.

Output

w1, w2:	Weight matrices obtained by training.
critvec:	Vector containing the criterion after each iteration.
iter:	# of iterations.

Description

Given a set of corresponding input-output pairs and an initial network, a two layer neural network is trained with the recursive prediction error method ("recursive Gauss-Newton"). Also pruned, i.e., not fully connected, networks can be trained. Most often the disadvantages of a recursive method are overwhelming compared with a batch method. The recursive methods may, however, be relevant for very large networks+data sets where lack of memory is a problem or when there is a high degree of redundancy in the data set. Different methods have been implemented with inspiration from adaptive control: exponential forgetting, constant trace and the so-called exponential forgetting and resetting algorithm (EFRA). The *method* field in *trparms* selects one of the three schemes. The default method is the exponential forgetting factor algorithm. See SETTRAIN for details.

The activation functions can be either *linear* or *tanh*. The network architecture is defined by the matrix 'NetDef' which has two rows. The first row specifies the hidden layer while the second specifies the output layer.

E.g.:
```
NetDef =  ['LHHHH'
           'LL---']
```
(L = linear, H = tanh)
Notice that the bias is included as an extra column in the weight matrices and that a weight is eliminated (i.e. 0 and not updated) by setting it to zero.

Example

Generate data as sinusiodal+noise
>> *PHI = 2*pi*rand(1,300);*
>> *Y = sin(PHI) + 0.2*randn(1,300);*
>> *plot(PHI,Y,'+');*

Initialize Network. 5 tanh hidden units, 1 linear output
>> *NetDef = ['HHHHH';'L----'];*
>> *[W1,W2,critvec,iter]=rpe(NetDef,[],[],PHI,Y);*
>> *drawnet(W1,W2,eps)*

Plot criterion evaluated after each iteration
>> *semilogy(critvec); grid;*
>> *xlabel('Iteration');*
>> *ylabel('Training error')*

Algorithm

Be careful not to select the forgetting factor too small in the forgetting factor method. Due to the large number of weights usually present in a network eigenvalues in the covariance matrix ("the inverse Hessian") might grow uncontrollably.

See Also

MARQ, BATBP, INCBP.

References

L. Ljung: *"System Identification - Theory for the User,"* Prentice-Hall, 1987.

J.E. Parkum: *"Recursive Identification of Time-Varying Systems,"* Ph.D. thesis, IMM, Technical University of Denmark, 1992.

M.E. Salgado, G. Goodwin, R.H. Middleton: *“Modified Least Squares Algorithm Incorporating Exponential Forgetting And Resetting,”* Int. J. Control, 47(2), 1988, pp. 477-491.

M. Nørgaard, O. Ravn, N. K. Poulsen, L. K. Hansen: *”Neural networks for Modelling and Control of Dynamic Systems,”* Springer-Verlag, London, 2000.

settrain

Purpose

Set parameters for a training algorithm.

Synopsis

trparms =settrain;

Set all parameters to default values.

settrain(trparms)

List all parameters.

trparms = settrain(trparms,'field1',value1,'field2',value2,...)

Set specific parameters

trparms.field1 = value1;

trparms.filed2 = value2;

etc.

If *value = 'default'* the parameter is set to its default value.

The following fields are valid:

Information displayed during training

infolevel - Display little information (0) or much (1).

Stopping criteria (all algorithms, see note below)

maxiter - Maximum iterations.

critmin - Stop if criterion is below this value.

critterm - Stop if change in criterion is below this value.

gradterm - Stop if largest element in gradient is below this value.

paramterm - Stop if largest parameter change is below this value.

NB: critterm, gradterm and paramterm must all be satisfied.

Weight decay (all algorithms trained with the Levenberg-Marquardt alg.)

D - Row vector containing the weight decay parameters. If D has one element a scalar weight decay will be used. If D has two elements, the first element will be used as weight decay for the hidden-to-output layer while second will be used for the input-to-hidden layer weights. For individual weight decays, D must contain as many elements as there are weights in the network.

Levenberg-Marquardt parameters

lambda - Initial Levenberg-Marquardt parameter.

Back-propagation parameters
eta - Step size.
alph - Momentum.

RPE parameters
method - Training method ('ff', 'ct', 'efra').

Forgetting factor (method='ff')
fflambda - Forgetting factor.
p0 - Covariance matrix is initialized to p0*I.

Constant trace (method='ct')
ctlambda - Forgetting factor.
alpha_min - Min. eigenvalue of P matrix.
alpha_max- Max. eigenvalue of P matrix.

EFRA (method='efra')
eflambda - Forgetting factor.
alpha - EFRA parameter.
beta - EFRA parameter.
delta - EFRA parameter.

For recurrent nets
skip - Do not use the first 'skip' samples for updating the weights.

For multi-output networks
repeat - Number of times the IGLS procedure should be repeated.

Remarks on the Stopping Criteria

The stopping criterion is not the same for all training algorithms. The batch algorithms use all the parameters; the recursive algorithms use only *maxiter*, *critmin* and *critterm*.

If it is important that training is continued until the weights are extremely close to the minimizing values, one should reduce *critterm*, *gradterm*, and *paramterm* (or at least one of them).

wrescale

Purpose

Rescale the weights of the trained network model if the training data was scaled with DSCALE prior to the training.

Synopsis

[w1,w2]=wrescale(method,W1,W2,Uscale,Yscale,NN)

Input

method	The function applied for generating the model. For example method='nnarx' or method='nnoe'. Use method='inverse' for inverse models (see the NNCTRL toolkit).
W1:	Input-to-hidden weights of network trained on scaled data.
W2:	Hidden-to-output weights.
Uscale:	Matrix containing the sample mean and standard deviation for each input. For time series an empty matrix, [], is passed.
Yscale:	Matrix containing mean and std's for each output.
NN:	Vector containing lag spaces, i.e., the number of past signals used as input to the network (see nnarx, nnarmax, nnoe ..). For ordinary feedforward networks ("function fitting" type networks) NN is left out.

Output

w1, w2:	Scaled weight matrices.

Description

WRESCALE rescales the weights for networks with LINEAR OUTPUT UNITS. Don't use it for networks with *tanh* output units! The function works for feedforward networks as well as for input-output models of dynamic systems (i.e. NNAR(X), NNARMA(X) and NNOE type models). If the function DSCALE was used for scaling the data to zero mean and unity variance before training, the weights should be rescaled after training so that the network can work on unscaled data. Notice that when the function is used on a pruned network, it is likely to reintroduce biases removed in the pruning session.

See Also

DSCALE.

xcorrel

Purpose

Calculate high-order cross-correlation functions for input-output models of dynamic systems.

Synopsis

Network generated by NNARX (or NNRARX):
xcorrel('nnarx',NetDef,NN,W1,W2,Y,U)

Network generated by NNARMAX1 (or NNRARMX1):
xcorrel('nnarmax1',NetDef,NN,W1,W2,C,Y,U)

Network generated by NNARMAX2 (or NNRARMX2):
xcorrel('nnarmax2',NetDef,NN,W1,W2,Y,U)

Network generated by nnoe:
xcorrel('nnoe',NetDef,NN,W1,W2,Y,U)

Input

See the function used for generating the model.
For time series the argument U is left out.

Description

The function calculates a number of high order cross-correlation functions for models that have been generated by one of the functions NNARX, NNRARX, NNARMAX1+2, NNRARMX1+2, or NNOE.

Ideally, the prediction errors from the trained neural network model should be unpredictable from all combinations of past inputs and outputs. A complete check for statistical independence is obviously not feasible so instead it is common to investigate a few "wisely" chosen correlation functions.

Plots of the following six (normalized) correlation functions are produced:

$$\hat{r}_{\varepsilon\varepsilon}(\tau) = \frac{\sum_{t=1}^{N-\tau}(\varepsilon(t,\hat{\theta}) - \bar{\varepsilon})(\varepsilon(t-\tau,\hat{\theta}) - \bar{\varepsilon})}{\sum_{t=1}^{N}(\varepsilon(t,\hat{\theta}) - \bar{\varepsilon})^2} = \begin{cases} 1, & \tau = 0 \\ 0, & \tau \neq 0 \end{cases}$$

$$\hat{r}_{u\varepsilon}(\tau) = \frac{\sum_{t=1}^{N-\tau}(u(t)-\overline{u})(\varepsilon(t-\tau,\hat{\theta})-\overline{\varepsilon})}{\left(\sum_{t=1}^{N}(u(t)-\overline{u})^2\right)^{1/2}\left(\sum_{t=1}^{N}(\varepsilon(t,\hat{\theta})-\overline{\varepsilon})^2\right)^{1/2}} = 0,\ \forall\tau$$

$$\hat{r}_{u^2\varepsilon^2}(\tau) = \frac{\sum_{t=1}^{N-\tau}(u^2(t)-\overline{u^2})(\varepsilon^2(t-\tau,\hat{\theta})-\overline{\varepsilon^2})}{\left(\sum_{t=1}^{N}(u^2(t)-\overline{u^2})^2\right)^{1/2}\left(\sum_{t=1}^{N}(\varepsilon^2(t,\hat{\theta})-\overline{\varepsilon^2})^2\right)^{1/2}} = 0,\ \forall\tau$$

$$\hat{r}_{u^2\varepsilon}(\tau) = \frac{\sum_{t=1}^{N-\tau}(u^2(t)-\overline{u^2})(\varepsilon(t-\tau,\hat{\theta})-\overline{\varepsilon})}{\left(\sum_{t=1}^{N}(u^2(t)-\overline{u^2})^2\right)^{1/2}\left(\sum_{t=1}^{N}(\varepsilon(t,\hat{\theta})-\overline{\varepsilon})^2\right)^{1/2}} = 0,\ \forall\tau$$

$$\hat{r}_{\varepsilon\beta}(\tau) = \frac{\sum_{t=1}^{N-\tau}(\varepsilon(t,\hat{\theta})-\overline{\varepsilon})(\beta(t-\tau-1)-\overline{\beta})}{\left(\sum_{t=1}^{N}(\varepsilon(t,\hat{\theta})-\overline{\varepsilon})^2\right)^{1/2}\left(\sum_{t=1}^{N}(\beta(t)-\overline{\beta})^2\right)^{1/2}} = 0,\ \tau\geq 0$$

$$\hat{r}_{\alpha\varepsilon^2}(\tau) = \frac{\sum_{t=1}^{N-\tau}(\alpha(t)-\overline{\alpha})(\varepsilon^2(t-\tau,\hat{\theta})-\overline{\varepsilon^2})}{\left(\sum_{t=1}^{N}(\alpha(t)-\overline{\alpha})^2\right)^{1/2}\left(\sum_{t=1}^{N}(\varepsilon^2(t,\hat{\theta})-\overline{\varepsilon^2})^2\right)^{1/2}} = \begin{cases} k,\ \tau=0 \\ 0,\ \tau\neq 0\end{cases}$$

$$\hat{r}_{\alpha u^2}(\tau) = \frac{\sum_{t=1}^{N-\tau}(\alpha(t)-\overline{\alpha})(u^2(t-\tau)-\overline{u^2})}{\left(\sum_{t=1}^{N}(\alpha(t)-\overline{\alpha})^2\right)^{1/2}\left(\sum_{t=1}^{N}(u^2(t,\hat{\theta})-\overline{u^2})^2\right)^{1/2}} = 0,\ \forall\tau$$

where

$$\beta(t) = u(t)\varepsilon(t,\hat{\theta})$$

$$\alpha(t) = y(t)\varepsilon(t,\hat{\theta})$$

$$k_2 = \frac{\left(\sum_{t=1}^{N}(\varepsilon^2(t,\hat{\theta})-\overline{\varepsilon^2})\right)^{1/2}}{\left(\sum_{t=1}^{N}(\alpha(t)-\overline{\alpha})^2\right)^{1/2}}$$

The overbar denotes the average of a signal

$$\bar{x} = \frac{1}{N}\sum_{t=1}^{N} x(t)$$

The normalized correlation functions (sometimes called the correlation coefficients) are displayed along with their 95% confidence interval.

Notice that NNVALID calculates the autocorrelation function of the prediction error.

Example

>> *load spmdata*
>> *NetDef = ['HHHH';'L---'];*
>> *NN=[2 2 1];*
>> *trparms=settrain;*
>> *trparms=settrain(trparms,'maxiter',300,'D',1e-3);*
>> [W1,W2,critvec,iter,lambda]=nnarx(NetDef,NN,[],[],trparms,y1,u1);
>> *xcorrel('nnarx',NetDef,NN,W1,W2,y2,u2);*

See Also

NNVALID.

Reference

S.A. Billings, Q.M. Zhu: *Nonlinear model validation using correlation tests*, International Journal of Control,Vol. 60, no. 6, pp. 1107-1120, 1994.

S.A. Billings, H.B. Jamaluddin, S. Chen: *Properties of neural networks with applications to modelling non-linear dynamical systems*, International Journal of Control,Vol. 55, no. 1, pp. 193-224, 1992.