Proceedings of the 1996 IEEE International Symposium
on Computer-Aided Control System Design
Dearborn, MI • September 15-18, 1996

The NNSYSID Toolbox

- A MATLAB® Toolbox for System Identification with Neural Networks

M. Nørgaard*, O. Ravn*, L.K. Hansen**, N.K. Poulsen**

*Department of Automation, building 326. pmn, or@iau.dtu.dk
**Department of Mathematical Modelling, building 321. nkp, lkh@imm.dtu.dk
Technical University of Denmark (DTU), 2800 Lyngby, Denmark

Abstract

To assist the identification of nonlinear dynamic systems, a set of tools has been developed for the MATLAB® environment. The tools include a number of different model structures, highly effective training algorithms, functions for validating trained networks, and pruning algorithms for determination of optimal network architectures. The toolbox should be regarded as a nonlinear extension to the System Identification Toolbox provided by The MathWorks, Inc [9]. This paper gives a brief overview of the entire collection of toolbox functions.

1. Introduction

Inferring models of dynamic systems from a set of experimental data is a task which relates to a variety of areas. Technical as well as non-technical. If the system to be identified can be described by a linear model quite standardized methods exist for approaching the problem. Furthermore, a number of highly advanced tools are available which offer assistance in solving the problem [9]. When it is unreasonable to assume linearity and if the physical insight into the system dynamics is too limited to propose a suitable nonlinear model structure, the problem becomes relatively complex. In this case some kind of generic nonlinear model structure is required. A large number of such model structures exist, each characterized by having different advantages and disadvantages. The multilayer perceptron neural network [7] has proven to be one of the most powerful tools in practice and thus it has been selected as the key technology in our work. The attention has been restricted to networks with a single hidden layer of *tanh* (or linear) units since these offer a satisfying flexibility for most practical problems. MATLAB 4.2 has been chosen as the environment in which to operate due to its popularity, the simple user-interface, and its excellent data visualization features. The MathWorks, Inc already offers a neural network toolbox [2]. Although a small overlap with this has been unavoidable the two toolboxes are, however, fundamentally different. While the Neural Network toolbox has been designed for covering a variety of network architectures and for solving many different types of problems, the NNSYSID toolbox is specialized to solve system identification problems. Because the overlap is limited we have therefore chosen to develop NNSYSID completely independent of the Neural Network toolbox.

All toolbox functions have been written as "m-functions," but some CMEX dublicates have been coded for speeding up the most time consuming functions. The only official MathWorks toolbox required is the Signal Processing Toolbox.

The NNSYSID toolbox has primarily been designed from a control engineering perspective, but can be used for many other applications. For example is time series analysis also supported (i.e. no exogenous variable/control signal). The toolbox is mainly created for handling Multi-Input-Single-Output systems (with possible time delays of different order). Identification of multi-output systems is only supported for the most common model structures.

A number of demonstation programs have been implemented with the GUI facilities of MATLAB 4.2. These are designed to give a quick introduction to the toolbox and demonstrates most of the functions. The toolbox is also accompanied by a "MATLAB style" manual, which fully documents the entire collection of functions [11].

The outline of the paper is as follows: first a brief recapitulation of the basic identification procedure is given and subsequently the toolbox functions are presented by category in accordance with the overall identification procedure.

0-7803-3032-3/96/$5.00 © 1996 IEEE

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on March 02,2010 at 07:30:10 EST from IEEE Xplore. Restrictions apply.

2. The Basic Procedure

Fig. 1 shows the procedure usually followed when identifying a dynamic system. Prior to the execution of the procedure two issues should be considered: what *a priori knowledge* about the system is available and what is the *purpose* (i.e., the intended application of the model)?. Typically, these issues will have a strong impact on the entire procedure.

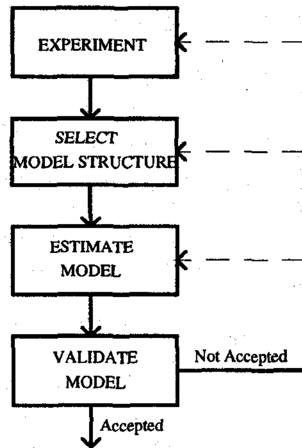

***Figure 1.** The system identification procedure.*

It is assumed that experimental data describing the underlying system in its entire operating range has been obtained in advance with a proper choice of sampling frequency:

$$Z^N = \{[u(t), y(t)] \,|\, t = 1,\ldots,N\}$$

$u(t)$ specifies the input to the system while $y(t)$ specifies the output.

The toolbox is designed to cover the remaining three stages as well as the paths leading from validation and back to previous stages. The following chapters will briefly describe the functions contained in the toolbox.

3. Selecting a Model Structure

Assuming that a data set has been acquired the next step is to select a set of candidate models. Unfortunately, this is much more difficult in the nonlinear case than in the linear. Not only is it necessary to choose a set of regressors, but a network architecture is required as well. The implemented model structures more or less follow the suggestions given in [15]. The idea is to select the regressors as for the conventional linear model structures and then, afterwards, determine the best possible neural network achitecture with the selected regressors as inputs.

The toolbox provides the six model structures listed below. $\varphi(t)$ is the regression vector, θ is the parameter vector containing the weights, g is the function realized by the neural network, and $\hat{y}(t|\theta) = \hat{y}(t|t-1,\theta)$ denotes the one-step ahead prediction of the output.

- *NNARX* structure. $\hat{y}(t|\theta) = g(\varphi(t),\theta)$ and

$$\varphi(t) = [y(t-1) \;\ldots\; y(t-n_a) \quad u(t-n_k) \;\ldots\; u(t-n_b-n_k+1)]^T$$

- *NNOE* structure. $\hat{y}(t|\theta) = g(\varphi(t),\theta)$ and

$$\varphi(t) = [\hat{y}(t-1|\theta) \;\ldots\; \hat{y}(t-n_a|\theta) \quad u(t-n_k) \;\ldots\; u(t-n_b-n_k+1)]^T$$

- *NNARMAX1* structure. $\hat{y}(t|\theta) = g(\varphi_1(t),\theta) + C(q^{-1})\varepsilon(t)$

$$\begin{aligned}\varphi(t) &= [\varphi_1^T(t) \quad \varepsilon(t-1) \;\ldots\; \varepsilon(t-n_c)]^T \\ &= [y(t-1) \;\ldots\; y(t-n_a) \\ &\qquad u(t-n_k) \;\ldots\; u(t-n_b-n_k+1) \\ &\qquad \varepsilon(t-1) \;\ldots\; \varepsilon(t-n_c)]^T\end{aligned}$$

  $\varepsilon(t)$ is the prediction error: $\varepsilon(t) = y(t) - \hat{y}(t|\theta)$ and C is a polynomial in the delay operator:
  $C(q^{-1}) = 1 + c_1 q^{-1} + \cdots + c_{n_c} q^{-n_c}$

- *NNARMAX2* structure. $\hat{y}(t|\theta) = g(\varphi(t),\theta)$ and

$$\begin{aligned}\varphi(t) = [&y(t-1) \;\ldots\; y(t-n_a) \\ &u(t-n_k) \;\ldots\; u(t-n_b-n_k+1) \\ &\varepsilon(t-1) \;\ldots\; \varepsilon(t-n_c)]^T\end{aligned}$$

- *NNSSIF* structure (state space innovations form). Predictor:

$$\hat{x}(t+1) = g(\varphi(t),\theta)$$
$$\hat{y}(t|\theta) = C(\theta)\hat{x}(t)$$

with

$$\varphi(t) = [\hat{x}^T(t) \quad u^T(t) \quad \varepsilon^T(t)]^T$$

To obtain an an observable model structure, a set of *pseudo-observability indices* must be specified just as in [9]. For supplementary information see chapter 4, appendix A in [8].

- *NNIOL* structure (Input-Output Linearization).

$$\begin{aligned}\hat{y}(t|\theta) &= f(y(t-1),\ldots,y(t-n_a),u(t-2),\ldots,u(t-n_b),\theta_f) \\ &\quad + g(y(t-1),\ldots,y(t-n_a),u(t-2),\ldots,u(t-n_b),\theta_g)u(t-1)\end{aligned}$$

f and g are two separate networks. This structure differs from the previous ones in that it is not motivated by a linear

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on March 02,2010 at 07:30:10 EST from IEEE Xplore. Restrictions apply.

model structure. NNIOL models are particularly interesting for control by *input-output linearization*.

For *NNARX* and *NNIOL* models there is an algebraic relationship between prediction and past data. The remaining models are more complicated since they all contain a feedback from network output (the prediction) to network input. In the neural network terminology these are called *recurrent* networks. The feedback may lead to instability in certain regimes of the system's operating range which can be very problematic. This will typically happen if either the model structure or the data material is insufficiant. The NN*ARMAX1* structure has been constructed to overcome this by using a linear moving average filter on the past prediction errors.

When a particular class of model structure has been selected, the next choice to be made is the number of past signals used as regressors (i.e. "the model order" or the "lag space"). It is desireable that the user has sufficient physical insight to choose these properly. However, the toolbox provides a function which occasional may come in handy. It implements a method based on so-called "lipschitz coefficients" which has been proposed in [6]. It is restricted to the determinstic case (or when signal-to-noise ratio is high). The MATLAB call is:

>> *OrderIndexMat = lipschit(U,Y,nb,na)*

where
U: input sequence.
Y: outputs sequence
nu: vector specifying input lags to be investigated.
ny: vector specifying output lags to be investigated.

4. Estimate a Model

The model estimation stage includes choosing a criterion of fit and an iterative search algorithm for finding the minimum of the criterion (i.e., training the network). The only type of criterion implemented is a regularized mean square error type criterion:

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

The matrix D is a diagonal matrix which is usually set to $D=\alpha I$. For a discussion of regularization by simple weight decay, see for example [10] and [16]. The toolbox provides the following four possibilities: no regularization, one common weight decay coefficient, one weight decay for the input-to-hidden layer and one for the hidden-to-output layer, and individual weight decay for all weights.

For multi-output systems it is possible to train NNARX and state-space models according to the criterion:

$$W_N(\theta, Z^N) = \frac{1}{2N}\sum_{t=1}^{N}\left(y(t) - \hat{y}(t|\theta)\right)^T \Lambda\left(y(t) - \hat{y}(t|\theta)\right) + \frac{1}{2N}\theta^T D\theta$$

The function *nnigls* implements the iterated generalized least squares procedure for iterative estimation of network weights and noise covariance matrix. The inverse of the estimated covariance matrix is in this case used as weight matrix in the criterion.

The main engine for solving the optimization problem is a version of the Levenberg-Marquardt method [3]. This is a batch method providing a very robust and rapid convergence. Moreover, it does not require a number of exotic design parameters which makes it very easy to use. In addition it is for some of the model structures also possible to train the network with a recursive prediction error algorithm [8]. This may have some advantages over batch algorithms when networks are trained on very large data sets. Either due to redundancy in the data set or because lack of computer storage is a problem. The recursive algorithm has been implemented with three different types of forgetting: Exponential forgetting, constant trace, and the EFRA algorithm [14].

The toolbox contains the following functions for generating models from a specified model structure:

1: Nonlinear System Identification	
nnarmax1	Identify a neural network ARMAX (or ARMA) model (linear noise filter).
nnarmax2	Identify a neural network ARMAX (or ARMA) model.
nnarx	Identify a neural network ARX (or AR) model.
nnigls	IGLS procedure for multi-output systems.
nnarxm	Identify a multi-output NNARX model.
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 model.
nnrarmx1	Recursive counterpart to NNARMAX1.
nnrarmx2	Recursive counterpart to NNARMAX2.
nnrarx	Recursive counterpart to NNARX.

To exemplify how these functions are called from MATLAB consider the *nnarx* function:
>> *[W1,W2,crit,iter]=*
nnarx(NetDef,NN,w1,w2,trparms,Y,U)

NetDef: A "string matrix" defining the network architecture.

```
NetDef=['HHHHHH'
        'L-----'];
```


Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on March 02,2010 at 07:30:10 EST from IEEE Xplore. Restrictions apply.

	specifies that the network has 6 *tanh* hidden units and 1 linear output.
NN:	*NN=[n_a n_b n_k]* defines the regression vector. n_a specifies past outputs, n_b past inputs, and n_k the time delay. For multi-input systems, n_b and n_k are row vectors.
w1, w2:	Matrices containing initial weights. If passed as [] they are initialized automatically
U:	Vector (Matrix) containing the input(s). This is left out for time series.
Y:	Vector containing the desired outputs.
trparms:	Vector containing different parameters associated with the training (max. # of iterations, error bound, weight decay). Can be passed as [].
W1, W2:	Weights after training.
crit:	Vector containing the criterion evaluated after each iteration.
iter:	Iterations executed before termination.

The functions for identifying models based on recurrent networks furthermore requires the parameter *skip*. This is used for preventing transient effects from corrupting the training (the '*skip*' first samples are not used for updating the weights):

>>[W1,W2,crit,iter]=...
nnoe(NetDef,NN,w1,w2,trparms,skip,Y,U)

5. Validation and Model Comparison

When a network has been trained, the next step is, according the procedure, to validate it. Table 2 shows the functions associated with this stage of the identification procedure.

2: Evaluation of Trained Networks	
ifvalid	Validation of models generated by NNSSIF.
ioleval	Validation of models generated by NNIOL.
kpredict	Compute and plot *k*-step ahead predictions.
nnfpe	FPE-estimate for I-O models of dynamic systems.
nloo	Leave-one-out estimate for NNARX models.
nnsimul	Simulate model of dynamic system.
nnvalid	Validation of I-O models of dynamic systems.
xcorrel	Display diffent cross-correlation functions.

The most common method of validation is to investigate predictions and prediction errors (residuals) by cross-validation on a fresh set of data, a *test set*. The functions *nnvalid, ioleval, ifvalid* assist such an investigation. This includes a comparison of the actual outputs and the predicted outputs, a histogram showing the distribution of the residuals, and the auto-correlation function of the residuals. A linear model is extracted from the network at each sampling instant to provide an impression of the "degree of nonlinearity" (see [17]). *xcorrel* computes a series of cross-correlation functions to check that the residuals are independent of (past) inputs and outputs [1].

As an example, the function *nnvalid*, which handles the validation for most of the model types, is called as follows if *nnarx* was used for generating the model:

>> [Yhat,V]=nnvalid('nnarx',NetDef,NN,W1,W2,y,u)

u and *y* specify the test set input and output signals (for multi-output systems only one output at a time is considered). *Yhat* contains the one-step ahead predictions produced by the network while *V* is the normalized sum of squared errors:

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

evaluated on the test set (the so-called *test error*). *V* is an important quantity since it can be regarded as an estimate of the generalization error. This should not be too large compared to the training error, in which case one can suspect that the network is overfitting the training data. If a test set is not available, the average generalization error:

$$J\equiv E\left\{\overline{V}(\hat{\theta})\right\},\ \overline{V}(\hat{\theta})=\lim_{N\to\infty}E\left\{V_N(\hat{\theta},N)\right\}$$

can be estimated from the training set alone by Akaike's final prediction error (FPE) estimate. Although a test set is available, the FPE estimate might still offer some valuable insights and in particular it can be quite useful for model comparison. For the basic unregularized criterion the estimate reads [8]:

$$\hat{J}_{FPE}=\frac{N+d}{N-d}V_N(\hat{\theta},Z^N)$$

d denotes the number of weights in the network. When the regularized criterion is used, the expression becomes more complex [10]:

$$\hat{J}_{FPE}(\mathbf{M})=\frac{N+\gamma_1}{\left(N+\gamma_1-2\gamma_2\right)}V_N(\hat{\theta},Z^N)$$

where

$$\gamma_1=tr\left[R(\hat{\theta})(R(\hat{\theta})+\tfrac{1}{N}D)^{-1}R(\hat{\theta})(R(\hat{\theta})+\tfrac{1}{N}D)^{-1}\right]$$

and

$$\gamma_2=tr\left[R(\hat{\theta})(R(\hat{\theta})+\tfrac{1}{N}D)^{-1}\right]$$

R is the Gauss-Newton Hessian evaluated in the minimum and γ_1 $(\approx\gamma_2)$ specifies the so-called *effective* number of weights in the network. The function *nnfpe* computes the FPE estimate and is for *NNARX* models called by:

>> [FPE,deff]=...
nnfpe('nnarx',NetDef, W1,W2,U,Y,NN,trparms);

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on March 02,2010 at 07:30:10 EST from IEEE Xplore. Restrictions apply.

In addition to the FPE estimate the effective number of weights in the network, *deff*, is also returned.

For NNARX models it is also possible to compute the so-called leave-one-out estimate of the average generalization error (*nnloo*). Due to the nature of this estimate it cannot be calculated for model structures based on recurrent networks.

6. The "Feedback" Paths

In fig. 1 a number of paths leading from validation back to the previous stages are shown. The path from validation to training symbolizes that it might be possible to obtain a better model if the network is trained with a different weight decay or if the weights are initialized differently. Since it is likely that the training algorithm ends up in a non-global minimum, the network should be trained a couple of times with different initializations of the weights. Regularization by weight decay has a smoothing effect on the criterion and several of the local minima are often removed when this is used.

Another path leads back to model structure selection. Because the model structure selection problem has been divided into two seperate subproblems, this can mean two things. Namely, "try another set of regressors" or "try another network architecture." While the regressors typically have to be chosen on a trial-and-error basis, it is to some extent possible to automate the network architecture selection. The most commonly used method is to prune a very large network until the optimal architecture is reached. The toolbox provides the so-called *Optimal Brain Surgeon (OBS)* algorithm for pruning the networks. OBS was originally proposed in [5], but in [4] it is modified to cover networks trained according to a regularized criterion.

3: 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).

If a model has been generated by *nnarx*, the OBS function is called as follows:

```
>> [thd,tr_err,FPE,te_err,deff,pvec] =nnprune('nnarx', ...
      NetDef,W1,W2,U,Y,NN,trparms,prparms,u,y);
```

U,Y and *u,y* are input-output sequences for training and test set, respectively.

prparms specifies how often the network is retrained. To run a maximum of 30 iterations each time 2% of the weights have been eliminated, set *prparms=[30 2]*. *prparms=[]* gives the default *[50 5]*.

tr_err, FPE, and *te_err* are training error, FPE estimate, and test error, respectively. These are all plotted while pruning and can be used for pointing out the optimal network architecture.

thd is a matrix containing the parameter vectors, θ, after each weight elimination. The last column of *thd* contains the weights for the initial network. The next-to-last column contains the weights for the network appearing after elimination of one weight, and so forth. To extract the weight matrices from *thd*, the function *netstuc* has been implemented. If, for example, the network containing 25 weights is the optimal, the weights are retrieved by:

```
>> [W1,W2] = netstruc(NetDef,thd,25);
```

7. Additional Functions

The toolbox contains a number of additional functions which did not fit directly into any of the groups mentioned above. In relation to system identification the functions in table 4 are often relevant.

4: Miscellanous Utilities	
drawnet	Draws a two-layer feedforward network.
dscale	Scale data to zero mean and variance 1.
getgrad	Derivative of network outputs w.r.t. weights.
wrescale	Rescale weights of trained network.

A number of functions for training and evaluation of ordinary feedforward networks for many other purposes than system identification (e.g., curve fitting) are provided as well. These are listed in table 5.

5: Functions for ordinary feedforward networks	
batbp	Batch version of the back-propagation algorithm.
fpe	FPE estimate of generalization error.
igls	IGLS estimation for multi-output networks.
incbp	Recursive (incremental) back-prop. algorithm.
loo	Leave-One-Out estimate of generalization error.
marq	Levenberg-Marquardt method.
marqlm	Memory saving implementation of L-M method.
nneval	Validation of feed-forward networks.
obdprune	Prune with Optimal Brain Damage (OBD).
obsprune	Prune with Optimal Brain Surgeon (OBS).
rpe	Recursive prediction error method.

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on March 02,2010 at 07:30:10 EST from IEEE Xplore. Restrictions apply.

To demonstrate the toolbox, a number of demonstration examples are provided. Together these demonstrates most of the functions in the toolbox. See table 6.

6: Demonstration Examples	
test1	Demonstrates different training methods.
test2	Demonstrates the *nnarx* function.
test3	Demonstrates the *nnarmax2* function.
test4	Demonstrates the *nnssif* function.
test5	Demonstrates the *nnoe* function.
test6	Demonstrates the regularization.
test7	Demonstrates pruning by OBS on the sunspot benchmark problem.

8. Conclusions

A variety of neural network architectures and training schemes have been proposed through time. The NNSYSID toolbox has been implemented under the philosophy: "always try simple things first" and is regarded as the subsequent step if one fails in identifying a linear model. It has been a key issue that the basis was a relatively simple type of neural network, and that training, evaluation, and architecture determination was made as automatic as possible. The toolbox has succesfully been used in a number of practical applications.

In [12] an add-on toolkit for control engineers is presented. This toolkit can be used for construction and simulation of a number of control systems based on neural networks.

The NNSYSID toolbox is available from the web-site at The Department of Automation, DTU. The address is: http://www.iau.dtu.dk/Projects/proj/nnsysid.html

References

[1] S.A. Billings, H.B., Jamaluddin, S. Chen, "Properties of Neural Networks With Applications to Modelling non-linear Dynamical Systems," Int. J. Control, Vol. 55, No 1, pp. 193-224, 1992.

[2] H. Demuth & M. Beale, "Neural Network Toolbox," The MathWorks Inc., 1993.

[3] R. Fletcher, "Practical Methods of Optimization," Wiley, 1987.

[4] 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, 1994

[5] 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.

[6] 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.

[7] J. Hertz, A. Krogh & R.G. Palmer, "Introduction to the Theory of Neural Computation," Addison-Wesley, 1991.

[8] L. Ljung, "System Identification - Theory for the User," Prentice-Hall, 1987.

[9] L. Ljung, "System Identification Toolbox User's Guide," The MathWorks Inc., 1991

[10] 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.

[11] M. Nørgaard, "Neural Network Based System Identification Toolbox," Tech. report 95-E-773, Department of Automation, Technical University of Denmark, 1995.

[12] M. Nørgaard, O. Ravn, N.K. Poulsen, L.K. Hansen, "NNCTRL - A CANCSD toolkit for MATLAB," accepted for the 1996 IEEE Symposium on Computer-Aided Control System Design, Dearborn, Michegan, USA.

[13] M.W. Pedersen, L.K. Hansen, J. Larsen, "Pruning With Generalization Based Weight Saliences: γOBD, γOBS," Proceedings of the Neural Informatioon Systems 8, 1995.

[14] M.E. Salgado, G. Goodwin, R.H. Middleton, "Modified Least Squares Algorithm Incorporating Exponential Forgetting And Resetting," Int. J. Control, 47, pp. 477-491, 1988.

[15] J. Sjöberg, H. Hjalmerson, L. Ljung, "Neural Networks in System Identification," Preprints 10th IFAC symposium on SYSID, Copenhagen, Denmark. Vol.2, pp. 49-71, 1994.

[16] J. Sjöberg & L. Ljung, "Overtraining, Regularization, and Searching for Minimum in Neural Networks," Preprint IFAC Symp. on Adaptive Systems in Control and Signal Processing, Grenoble, France. pp. 669-674, 1992.

[17] O. Sørensen, "Neural Networks in Control Applications," Ph.D. Thesis, Aalborg University, Department of Control Engineering, 1994.

Authorized licensed use limited to: Danmarks Tekniske Informationscenter. Downloaded on March 02,2010 at 07:30:10 EST from IEEE Xplore. Restrictions apply.