15th Triennial World Congress, Barcelona, Spain

SYSTEM IDENTIFICATION USING DYNAMIC NEURAL NETWORKS: TRAINING AND INITIALIZATION ASPECTS [1]

V.M. Becerra * **J.M.F. Calado** ** **P.M. Silva** ** **F. Garces** *

* *The University of Reading*
Department of Cybernetics
Whiteknights
Reading RG6 6AY, United Kingdom
E–mail: v.m.becerra@reading.ac.uk
** *ISEL- Instituto Superior de Engenharia de Lisboa*
Polytechnic Institute of Lisbon
Mechanical Engineering Studies Centre
Rua Conselheiro Emdio Navarro
1949-014 Lisboa, Portugal
E–Mail: jcalado@dem.isel.pt

Abstract: This paper explores training and initialization aspects of dynamic neural networks when applied to the nonlinear system identification problem. A well known dynamic neural network structure contains both output states and hidden states. Output states are related to the outputs of the system represented by the network. Hidden states are particularly important in allowing dynamic neural networks to approximate complex nonlinear dynamics. An optimisation based method is proposed in this paper for properly initialising the hidden states of a dynamic neural network, so as to avoid the introduction of bias in the network parameters as a result of incorrect hidden state initialisation. Furthermore, a simple optimisation based method is proposed to initialise the hidden states once the network has been trained. The methods are illustrated with experimental data taken from a laboratory scale pressure plant. Copyright © 2002 IFAC.

Keywords: system identification, nonlinear systems, neural networks

1. INTRODUCTION

Neural networks have become a standard tool for the identification of dynamic systems. Many applications use static neural networks to build nonlinear input–output models of the plant. The use of neural networks for dynamic system identification has been extensively researched in the last two decades. The attention of researchers was first focused on static networks such as multilayer perceptrons (MLPs) (Rumelhart and McClelland, 1986), (Churchland *et al.*, 1992), and radial basis functions (Broomhead and Lowe, 1988). The inputs to these static networks are usually delayed values of the inputs and outputs of the plant. The neural network is used to synthesise a nonlinear map. This approach, however, has some disadvantages:

- The input structure is not easy to choose.
- The discrete time model requires re–training when the sampling time is changed.
- The problem of discrete time non–linear control is not as well understood as that of continuous time nonlinear control.

For the purposes of nonlinear system identification dynamical systems, some of the most relevant architectures that do not suffer from the above disad-

[1] Work supported by the British Council under the Treaty of Windsor Programme

vantages are the continuous time Hopfield networks and their variations (Hopfield, 1982), (Koiran, 1994). The first dynamic neural networks (DNNs) were introduced by Hopfield in the context of associative memory (Hopfield, 1984), (Hopfield and Tank, 1985), (Hopfield and Tank, 1986), but later modifications made them capable of approximating multivariable dynamic systems. Such networks can be represented by a nonlinear state space model. The problem of nonlinear dynamic system approximation represents an extension of the problem of approximating time series and trajectories (Funahashi, 1989). Input affine DNNs can approximate autonomous nonlinear systems (Funahashi and Nakamura, 1993), (Kimura and Nakano, 1998), systems linearly coupled to the control for single-input single-output (SISO) systems (Delgado *et al.*, 1995) as well as multivariable control affine systems (Garces *et al.*, 1999), (Kambhampati *et al.*, 2000) and furthermore general nonlinear systems (Garces, 2000). Stability conditions for DNNs have also been analysed (Matsuoka, 1992), (Sanchez and Perez, 1999).

This paper explores the problems of training and initialization of dynamic neural networks. DNNs are characterised by having a number of hidden neurons, each of which possesses a dynamic state. Once the architecture of the DNN is defined in terms of number of inputs, outputs and number of states, the number of hidden states is the difference between the number of states and the number of outputs. It is shown in this paper that it is important to consider the initial values of the hidden states of the DNNs both for network training and use, and that arbitrary initialization of the DNN's hidden states during training introduces bias in the parameters. Interestingly, in the literature published so far on DNNs applied to system identification, the initialisation of the network's hidden states has typically been done using zeros.

The paper is organised as follows. Section 2 introduces dynamic neural networks. Section 3 describes the problem of dynamic neural network training and describes the effects of not including the initial states of the hidden neurons in the decision vector. Section 4 introduces a method for initialising dynamic neural networks. Section 5 describes a case study using data from a pressure pilot plant. Section 6 gives final remarks about this work.

2. DYNAMIC NEURAL NETWORKS

The introduction of feedback into a feedforward neural network architecture produces a state space dynamic model. Given suitable parameters for the network the orbits generated by the states converge to fixed equilibrium points.

Originally, recurrent networks were introduced in the context of associative or content addressable memory (CAM) problems (Kohonen, 1989) and (Hopfield,

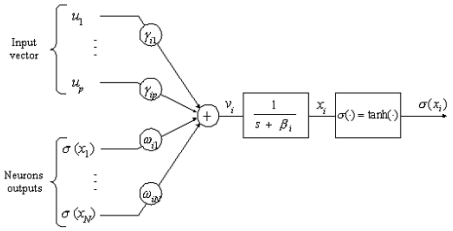

Fig. 1. Diagram of a dynamic neuron

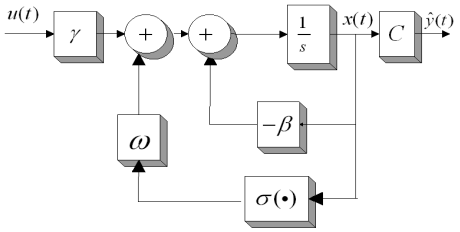

Fig. 2. Dynamic neural network

1984). The uncorrupted pattern is used as a stable equilibrium point and its noisy versions should lie in its basin of attraction. In this way, a dynamical system associated with a set of patterns is created. If such a CAM correctly partitions the whole working space, then any initial condition (corresponding to a sample pattern) should have a steady-state solution corresponding to the uncorrupted pattern. The dynamics of such a classifier serve as a filter.

The model is defined by a one-dimensional array of neurons; each unit can be described as follows,

$$\dot{x}_i = -\beta_i x_i + \sum_{j=1}^{n} \omega_{ij}\sigma(x_j) + \sum_{j=1}^{p} \gamma_{ij} u_j \qquad (1)$$

where β_i, ω_{ij} and γ_{ij} are adjustable weights, with $1/\beta_i$ as positive time constant and $p \leq n$, x_i the activation state of unit i, and $u_1, \ldots, u_m$ the input signals as seen in Figure 1. The function $\sigma(\cdot)$ is typically a nonlinear sigmoid-type function like the hyperbolic tangent function. The DNN is formed by a single layer of n units as in Equation (1). For convenience, the output of the network is often taken as the first p units of the state vector x, leaving $n-p$ units as hidden neurons. The network is defined in Equation (2) by the vectorised expression of Equation (1),

$$\begin{aligned} \dot{x} &= -\beta x + \omega\sigma(x) + \gamma u \\ \hat{y} &= C_n x \end{aligned} \qquad (2)$$

where $x \in \Re^n$ is a state vector, $u \in \Re^m$ is the input vector, $\hat{y} \in \Re^p$ is the output vector, $\omega \in \Re^{n\times n}$, $\sigma(x) = [\sigma(x_1), \ldots, \sigma(x_n)]^T$, $\gamma \in \Re^{n\times m}$, $C_n = [I_{p\times p} \emptyset_{p\times(n-p)}]$, and $\beta \in \Re^{n\times n}$ is a diagonal matrix with elements $[\beta_1, \ldots, \beta_n]$ in the diagonal.

It is assumed that $n > p$, so that the network has $n-p$ hidden units. Hidden units are used to increase the dynamic mapping potential of the network. Their dynamics allow DNN's to discover and exploit regularities in the system, such as symmetries or replicated structure (Hinton, 1986) and (Sejnowski *et al.*, 1986).

The state vector of the DNN can thus be partitioned into the output states x_o and the hidden states x_h:

$$x = \begin{bmatrix} x_o \\ x_h \end{bmatrix} \tag{3}$$

where $x_o \in \Re^p$ and $x_h \in \Re^{n-p}$.

3. TRAINING DYNAMIC NEURAL NETWORKS

Suppose that data have been collected from a real system that is to be modelled by means of a dynamic neural network. Consider a training data set with N input–output pairs and sampling time T_s:

$$Z_N = [y(t_k), u(t_k)]_{k=1,N} \tag{4}$$

where $y \in \Re^p$ is the measured output, $u \in \Re^m$ is the input variable, and k is a sampling index. Then the problem of training the DNN to learn the dynamics from data set Z_N may be written as an optimisation problem.

The Prediction Error Method (Ljung, 1999) attempts to find the estimated parameter vector $\theta \in \Re^{n_\theta}$ such that a loss function (typically the mean square error) is minimised:

$$V_N(\theta, Z_N) = \frac{1}{2N} \sum_{k=1}^{N} [y(t_k) - \hat{y}(t_k|\theta)]^2 \tag{5}$$

where $y(t_k|\theta)$ is the output vector of the network (2) at time t_k given the decision vector θ.

If the sampling time is short compared with the dynamics of the system that generated the data to be fitted, then the loss function V_N may be written as follows:

$$V_N(\theta, Z_N) = \frac{1}{2Nh} \int_{t_1}^{t_N} [y(t) - \hat{y}(t|\theta)]^2 dt \tag{6}$$

where the integral sign denotes numerical quadrature using a fixed step size algorithm, and $h = T_s$ is the integration step used.

A nonlinear identification problem can be cast as a nonlinear unconstrained optimisation problem:

$$\min_{\theta} \quad V_N(\theta, Z_N)$$

The optimisation is typically carried out using unconstrained Quasi–Newton methods or genetic algorithms. The optimisation problem associated with training usually exhibits local minima, so several runs from different (possibly random) initial decision variables should be made, noting that genetic algorithms are less sensitive to the initial values of the decision variables than Quasi–Newton methods, at the expense of higher computational requirements.

For training purposes, the decision vector that has typically been reported in the literature is based on the matrix coefficients of the DNN (2):

$$\theta = \begin{bmatrix} \beta_d \\ \text{vec}(\omega) \\ \text{vec}(\gamma) \end{bmatrix} \tag{7}$$

where β_d is a vector with the diagonal elements of β and $\text{vec}(\cdot)$ is a vector created with the stacked columns of an argument matrix $(\cdot)$.

It is proposed in this paper that the decision vector θ be augmented to include the initial values of the hidden states of the DNN, $x_h(t_1)$:

$$\theta = \begin{bmatrix} \beta_d \\ \text{vec}(\omega) \\ \text{vec}(\gamma) \\ x_h(t_1) \end{bmatrix} \tag{8}$$

The training procedure would be as follows:

Procedure 1. (DNN Training).

- Step 1: Initialize the output states as follows:

$$x_o(t_1) = y(t_1) \tag{9}$$

- Step 2: Initialize the values of β_d, ω, γ and $x_h(t_1)$ with random values. Form the initial decision vector $\theta^{(0)}$ according to Equation (8).
- Step 3: Compute the decision vector $\hat{\theta}$ by solving the associated optimisation problem, with $V_n(\theta, Z_N)$ given by either Equation (5) or (6):

$$\hat{\theta} = \arg\min_{\theta} \quad V_N(\theta, Z_N) \tag{10}$$

Notice that the output states are normally initialized with the values of the output variables. Also notice that several runs of Procedure 1 may be required to check for local minima.

If the initial values of the hidden states are not included in the vector of decision variables, then it is likely that the resulting model parameters β, ω and γ will be biased. The explanation of this is as follows. Although the initial values of the hidden states $x_h(t_1)$ do not affect the initial values of the output states $x_o(t_1)$ (which are typically set to match the real system's output), the initial values of the hidden states do affect the initial time derivatives of the output states $\dot{x}_o(t_1)$. This can be easily inferred by looking at the following form of Equation (2) at time t_1 and recalling that ω is a full matrix:

$$\frac{d}{dt}\begin{bmatrix} x_o(t_1) \\ x_h(t_1) \end{bmatrix} = -\beta \begin{bmatrix} x_o(t_1) \\ x_h(t_1) \end{bmatrix} + \omega\sigma\left(\begin{bmatrix} x_o(t_1) \\ x_h(t_1) \end{bmatrix}\right) + \gamma u(t_1) \tag{11}$$

So, by fixing the initial values of the hidden states to zero for training purposes, the initial time derivatives of the DNN outputs may have different values than the time derivatives of the data that the network is required to learn. This will force the training algorithm to adjust the network parameters β, ω and γ to compensate

for the initial incorrect values in the derivatives, so introducing a bias in the values of these parameters.

In linear state space modelling, the effect of the initial states depends on the stability of the model and can be separated from the influence of the external input. If the linear model is asymptotically stable, then the effect of the initial states decays exponentially with time, and if the system dynamics are fast, then the effect of the initial states disappears quickly (Ljung, 1999).

On the other hand, in nonlinear state space modelling, the effect of the initial states is richer: firstly, the effect of the initial states cannot be in general separated from the effect of the external input. Secondly, even if the external input is kept constant, the system may exhibit, for example, multiple equilibrium points, instability, limit cycles or chaotic behaviour, depending on the values of the initial states (Khalil, 1992).

4. INITIALIZING DYNAMIC NEURAL NETWORKS

Once a DNN has been trained, it is likely that the designer will use it on a different data set for validation or simulation purposes, or it may even be used on–line as part of a monitoring or nonlinear control scheme.

Denote $y_v(t_1)$ the value of the output vector at time t_1 and $u_v(t_1)$ as the value of the input vector at time t_1.

A simple two step procedure can be used to initialize the DNN at time t_1

Procedure 2. (DNN Initialisation).

- Step 1: Initialize the output states $x_o(t_1)$ with the values of the system's output at time t_1:

$$x_o(t_1) = y(t_1) \tag{12}$$

- Step 2: Initialize the hidden states $x_h(t_1)$ by solving the following optimisation problem:

$$x_h(t_1) = \arg\min_{x_h(t_1)} (\dot{y}(t_1) - \dot{x}_o(t_1))^2 \tag{13}$$

  where $\dot{y}(t_1)$ can be calculated from known data using finite differences and $\dot{x}_o(t_1)$ can be calculated from Equation (11), given the values of $x_o(t_1)$, $x_h(t_1)$, β, ω and γ.

If the DNN is being used for validation purposes, then $\dot{y}(t_1)$ may be calculated using the validation data set. On the other hand, if the DNN is being used in an on–line application, then $\dot{y}(t_1)$ can be calculated using the recent history of $y(t)$.

By using Procedure 2, it is ensured that the initial value of the hidden state vector $x_h(t_1)$ is such that the initial output derivative of the DNN, $\dot{x}_o(t_1)$, is as close as possible to the required value $\dot{y}(t_1)$.

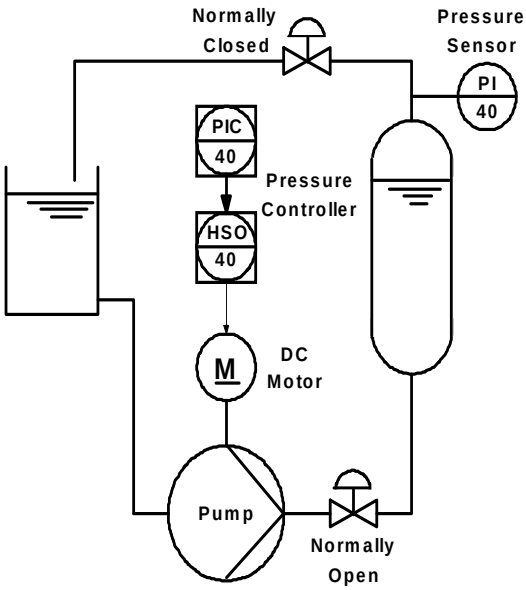

Fig. 3. Schematic diagram of the pilot plant

Notice that in general the number of hidden states is different from the number of outputs. Therefore, formulating the DNN initialization as an optimisation problem is more general than formulating it as a nonlinear equation solution problem.

5. CASE STUDY

To illustrate the concepts presented above, a case study has been carried out using data from a pressure pilot plant, which is described below.

5.1 *The pressure pilot plant*

The pressure pilot plant used in this case study is illustrated in Figure 3. It consists of a pressure vessel containing air and water. The air pressure is measured at the top of the vessel by means of a pressure transducer. A hydraulic pump is used to create a water flow that enters the vessel through an inlet pipe and so decreases the air volume, thus increasing its pressure. For a given pump rotation speed the system reaches an equilibrium point where no extra water enters the vessel. Furthermore, the direction of flow can be reversed such that the level decreases and so does the air pressure. The input signal, with a range of 0-10 V, is the voltage applied to the power amplifier that drives the DC motor that operates the hydraulic pump. The signals are sent and acquired by a supervisory PC via a Profibus network. The pressure signal ranges between 0 an 100 mBar. The sampling time used was 0.165 s.

5.2 *Results*

Both the training and validation data sets were obtained experimentally and had 500 input–output samples each. The optimisation was carried out using a

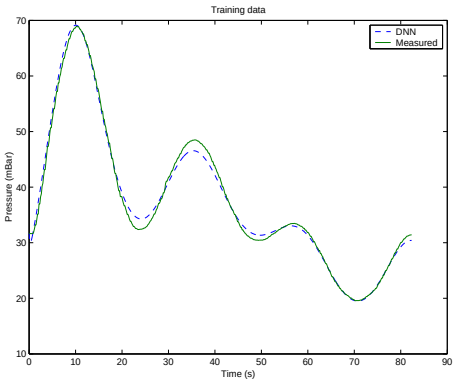

Fig. 4. Training trajectories with optimised initial hidden states

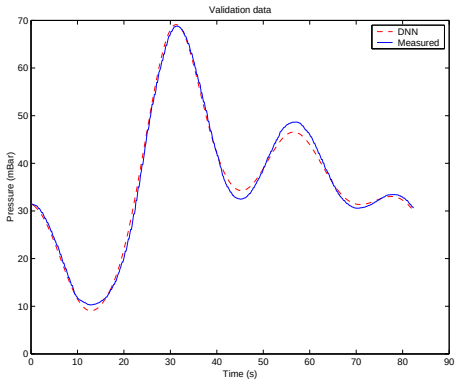

Fig. 5. Validation trajectories with optimised initial hidden states

Quasi-Newton unconstrained optimisation algorithm implemented in function *fminunc*, which is part of the Optimisation Toolbox in MATLAB version 6.1. The DNN used had a total two states: one output state and one hidden state. In order to make a fair comparison, the training runs were carried out from the same initial values of the parameters β, ω and γ. This case study compares the training and validation performance of the models calculated with and without the optimisation of the hidden states.

For training purposes, all signals were scaled and translated such that their range was in the interval [-1,1]. Figures 4 and 5 show the training and validation data, respectively, with optimised initial hidden states, as described in Sections 3 and 4. Figures 6 and 7 show the training and validation data, respectively, with the initial hidden state set to zero. Notice the differences between the results with and without optimised initial hidden state.

The values of the parameters when the initial hidden state was optimised were:

$$\beta_d = \begin{bmatrix} 0.2479 \\ 5.6164 \end{bmatrix} \tag{14}$$

$$\omega = \begin{bmatrix} -2.1166 & -7.2336 \\ -1.1385 & 2.1893 \end{bmatrix} \tag{15}$$

$$\gamma = \begin{bmatrix} 5.3782 \\ 2.7541 \end{bmatrix} \tag{16}$$

The values of the parameters when the initial hidden state was set to zero were:

$$\beta_d = \begin{bmatrix} 0.2166 \\ 23.7131 \end{bmatrix} \tag{17}$$

$$\omega = \begin{bmatrix} -2.7336 & -6.9816 \\ -5.8830 & 10.6528 \end{bmatrix} \tag{18}$$

$$\gamma = \begin{bmatrix} 5.6761 \\ 11.7509 \end{bmatrix} \tag{19}$$

Notice the differences in the parameter values in both cases, which indicate the presence of bias when the initial hidden state was not optimised.

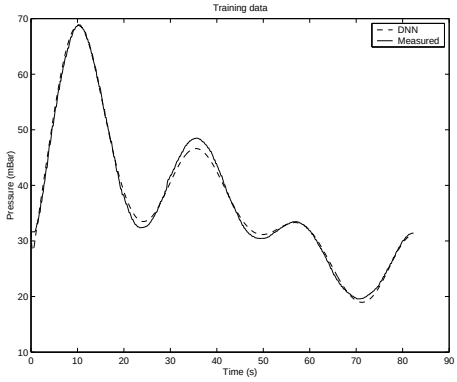

Fig. 6. Training trajectories with zero initial hidden states

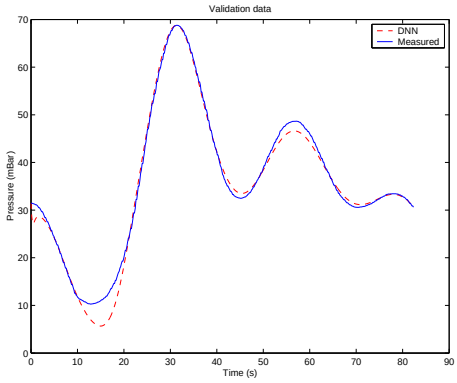

Fig. 7. Validation trajectories with zero initial hidden states

6. CONCLUSIONS

An optimisation based method has been proposed in this paper for properly initialising the hidden states of a dynamic neural network, so as to avoid the introduction of bias in the network parameters as a result of incorrect hidden state initialisation. Furthermore, a simple optimisation based method has been proposed to initialise the hidden states once the network has been trained. The methods are illustrated with experimental data taken from a laboratory scale pressure plant.

7. REFERENCES

Broomhead, D. S. and D. Lowe (1988). Multi-variable functional interpolation and adaptive networks. *Complex Systems* **2**, 321–355.

Churchland, P. S., T.J. Sejnowski and T.J. Seynowski (1992). *The computational brain*. MIT Press. Cambridge, Mass.

Delgado, A., C. Kambhampati and K. Warwick (1995). Dynamic recurrent neural-network for system-identification and control. *IEE Proceedings – Control Theory and Applications* **142**(4), 307–314.

Funahashi, K. (1989). On the approximate realization of continuous-mappings by neural networks. *Neural Networks* **2**(3), 183–192.

Funahashi, K. and Y. Nakamura (1993). Approximation of dynamical-systems by continuous-time recurrent neural networks. *Neural Networks* **6**(6), 801–806.

Garces, F. (2000). Dynamic neural networks for approximate input-output linearisation-decoupling of dynamic systems. PhD Thesis. University of Reading.

Garces, F., C. Kambhampati and K. Warwick (1999). Dynamic recurrent neural networks for identification of a multivariable nonlinear evaporator system. In: *International Conference on Dynamic Control Systems, DYCONS99*. World Scientific. Ottawa.

Hinton, G. E. (1986). Learning distributed representations of concepts. In: *Eighth Annual Conference of the Cognitive Science Society*. Amherst, Mass.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America-Biological Sciences* **79**(8), 2554–2558.

Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of 2-state neurons. *Proceedings of the National Academy of Sciences of the United States of America-Biological Sciences* **81**(10), 3088–3092.

Hopfield, J. J. and D. W. Tank (1985). Neural computation of decisions in optimization problems. *Biological Cybernetics* **52**(3), 141–152.

Hopfield, J. J. and D. W. Tank (1986). Computing with neural circuits - a model. *Science* **233**(4764), 625–633.

Kambhampati, C., F. Garces and K. Warwick (2000). Approximation of non-autonomous dynamic systems by continuous time recurrent neural networks. In: *Neural networks* (S. I. Amari, Ed.). IEEE International Conference on Neural Networks. Como, Italy. pp. I–64–I–69.

Khalil, Hassan K. (1992). *Nonlinear systems*. Macmillan Pub. Co.. New York.

Kimura, M. and R. Nakano (1998). Learning dynamical systems by recurrent neural networks from orbits. *Neural Networks* **11**(9), 1589–1599.

Kohonen, T. (1989). *Self-organization and associative memory*. 3rd ed.. Springer-Verlag. Berlin ; New York.

Koiran, P. (1994). Dynamics of discrete-time, continuous state hopfield networks. *Neural Computation* **6**(3), 459–468.

Ljung, Lennart (1999). *System identification : theory for the user*. 2nd ed.. Prentice Hall PTR. Upper Saddle River, NJ.

Matsuoka, K. (1992). Stability conditions for nonlinear continuous neural networks with asymmetric connection weights. *Neural Networks* **5**(3), 495–500.

Rumelhart, D. E. and J. L. McClelland (1986). *Parallel distributed processing : explorations in the microstructure of cognition*. MIT Press. Cambridge, Mass.

Sanchez, E. N. and J. P. Perez (1999). Input-to-state stability (ISS) analysis for dynamic neural networks. *IEEE Transactions on Circuits and Systems I-Fundamental Theory and Applications* **46**(11), 1395–1398.

Sejnowski, T. J., P. K. Kienker and G. E. Hinton (1986). Learning symmetry groups with hidden units - beyond the perceptron. *Physica D* **22**(1-3), 260–275.