Neuronal Information Processing - From Biological Data to Modelling and Applications (World Scientific, 1998).

REDUCING THE COMPLEXITY OF NEURAL NETS FOR INDUSTRIAL APPLICATIONS AND BIOLOGICAL MODELS

Gérard DREYFUS

ESPCI, Laboratoire d'Électronique (CNRS UPR 9081)

10 rue Vauquelin, 75005 Paris

email: Gerard.Dreyfus@espci.fr

ABSTRACT

The fundamental property of feedforward neural networks - parsimonious approximation - makes them excellent candidates for modeling *static* nonlinear processes from measured data. Similarly, feedback (or recurrent) neural networks have very attractive properties for the *dynamic* nonlinear modeling of artificial or natural processes; however, the design of such networks is more complex than that of feedforward neural nets, because the designer has additional degrees of freedom. In the present paper, we show that this complexity may be greatly reduced by (i) incorporating into the very structure of the network all the available mathematical knowledge about the process to be modeled, and by (ii) transforming the resulting network into a "universal" form, termed *canonical form*, which further reduces the complexity of analyzing and training dynamic neural models.

1. INTRODUCTION

For historical reasons, networks of formal neurons (hereinafter termed neural networks) have been extensively used for classification purposes, essentially in the framework of pattern recognition (Bishop, 1995). However, this is but a small fraction of the potential applications of neural nets. Actually, the latter are basically universal approximators, which can be advantageously used for statistical nonlinear data modeling (nonlinear regression), i.e. for building nonlinear models, either static or dynamic, from measured data. In the present paper, we focus on dynamic nonlinear models. We show how to go beyond nonlinear regression by incorporating prior knowledge into the structure of the neural networks, thereby making them "gray boxes" which achieve a valuable tradeoff between black-box models (designed solely from measured data) and knowledge-based models (designed solely from mathematical equations resulting from a physical - or chemical, biological, economical, etc. - analysis of the process). This approach may result in complex dynamic network structures; we further show how this complexity can be reduced by the fact that any dynamic discrete-time neural net, however complex, is amenable to a canonical form which can be analyzed and trained.

2. STATIC MODELS

2.1. The formal neuron

A neuron is a bounded, parameterized nonlinear function f of several variables x_i. The variables of the function are usually called the *inputs* of the neuron, the value of the function its *output*, and a parameter (or *weight*) c_i is attached to each variable. The most popular neuron (for reasons explained below) is a neuron whose output y is a sigmoid function of a linear combination of its inputs (a constant term equal to 1, called *bias*, is appended to the set of variables):

$$y = \tanh \left| c_0 + \sum_{i=1}^{n} c_i x_i \right|$$

where n is the number of variables. Thus, the output of the neuron is nonlinear with respect to the variables *and with respect to the parameters* c_i.

A formal neuron is usually represented graphically as shown on Figure 1.

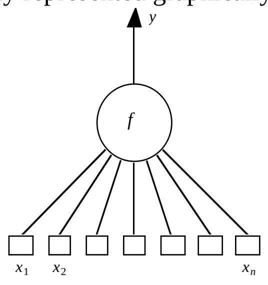


Figure 1

A formal neuron

2.2. Static modeling with feedforward neural networks

2.2.1. Feedforward neural networks

A feedforward neural network is a network whose graph of connections is *acyclic*: the output of a feedforward network is a nonlinear function of the inputs, resulting from the composition of the nonlinear functions performed by the neurons. Among the possible architectures of feedforward networks, networks having inputs, one layer of neurons, and a linear output, have the property of *parsimony* which will be described in section 2.2.2. Such an architecture, shown on Figure 2, is frequently called a "multilayer Perceptron", or MLP.

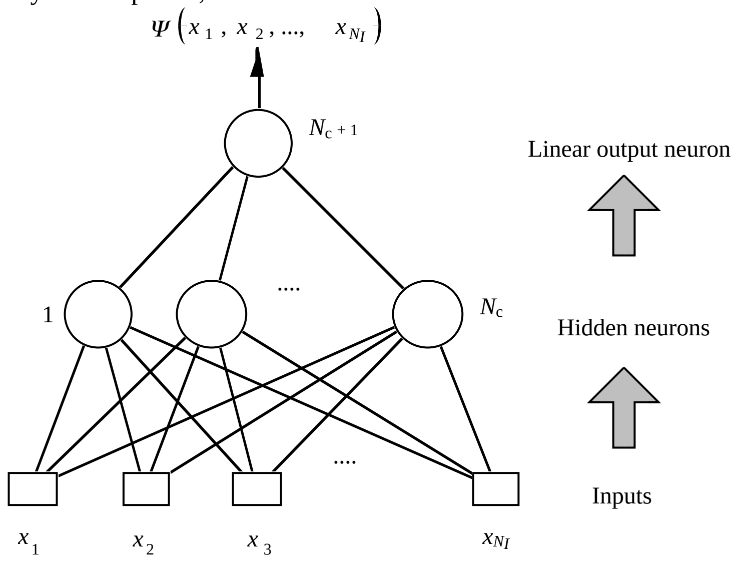


Figure 2

A multilayer Perceptron

Feedforward neural nets are static: if the inputs are constant in time, the outputs are also constant. In other words, *time* does not play any functional role in the behavior of feedforward nets.

2.2.2. Some feedforward neural networks are parsimonious universal approximators

Feedforward neural networks are universal approximators: given a (sufficiently regular) nonlinear function $\Phi(x_1, x_2, ..., x_n)$, and a given desired accuracy, there exists a neural network of the type shown on Figure 2, with a finite number of hidden neurons, that can approximate function Φ uniformly with the desired accuracy, within a domain of input space. This property is shared by a number of other approximators such as spline functions, radial functions, Fourier series, etc.

Feedforward neural nets *which are nonlinear with respect to the parameters* (such as networks of neurons described in section 2.1) are *parsimonious*, i.e. they require a smaller number of adjustable parameters than approximators which are linear with respect to the parameters (such as polynomials for instance, or radial basis function with fixed centers and variances) (Hornik et al. 1994). Specifically, the number of parameters required increases *linearly* with the number of input variables, whereas it grows *exponentially* for approximators which are linear with respect to the parameters. Therefore, the use of neural networks (of the type defined above) allows a significant decrease of the number of parameters when the number of inputs is "large", i.e. (from the extensive experience gathered in our group), when the number of variables is larger than 2.

Qualitatively, the origin of parsimony is the following: when the output is linear with respect to the parameters, it is a linear combination of functions *whose shapes are fixed*; in contrast, the output of a multilayer Perceptron is a linear combination (with adjustable parameters which are the weights of the second layer of connections) of functions *whose shapes are adjustable* through the weights of the first layer. The additional flexibility due to the fact that the shapes of the functions to be combined are adjusted, together with the parameters of the combination themselves, is the key to the parsimony.

2.2.3. From nonlinear function approximation to nonlinear regression estimation

In practice, function approximation is of little or no interest to the modeler, whether engineer or biologist. Actually, the problem one is confronted with is the following: given a *finite* set of measurements of a quantity of interest, and of the factors that have an influence on it, find the simplest function that "best" describes the measured data, i.e. that "best" accounts for the deterministic relationship which is assumed to exist between the quantity of interest (or output of the model) and the factors (or inputs of the model); in other words, find the simplest function that "best" approximates the regression function of the quantity of interest. If the amount of available data was infinite, the regression function would be fully determined as the expectation value of the quantity of interest. Since the amount of data is finite, the best one can hope for is to find a satisfactory approximation of the regression function. In order to do this, one chooses a family of parameterized functions (polynomials, neural nets, radial functions, wavelets, etc.) and estimates the parameters of the chosen function that provide the best fit, i.e. estimates the parameters that minimize the sum of the squared errors over the set of measurements (differences between the measured values of the quantity of interest and the corresponding values estimated by the model).

Hence, neural networks appear as a family of parameterized functions that are attractive candidates for computing a good approximation, in the least squares sense, of the - forever unknown - regression function. The parameters are estimated from the available measurements through the so-called *training phase*, which is nothing but a numerical process whereby the sum of the squared errors is minimized with respect to the parameters.

The price that one has to pay for the parsimony is apparent during this phase: since the output of the model of a parsimonious neural network is nonlinear with respect to the parameters, the least-squares cost function is not a quadratic function of the parameters. Therefore, the standard, fast and simple, least-squares procedures are not applicable: one has to resort to nonlinear minimization methods, which are iterative gradient techniques.

Despite this shortcoming, the advantages of parsimony for nonlinear regression are definitely worth the price: since the number of measurements required to perform a meaningful estimation of the parameters is roughly proportional to the number of parameters, the parsimony allows the neural network designer to get, from a given amount of data, a better approximation of the regression than if he used a non-parsimonious approximator

2.2.4. Summary

Feedforward neural networks having one layer of hidden neurons with sigmoid nonlinearity are parsimonious parameterized approximators of nonlinear functions. They can be advantageously used for estimating regression functions from measured data. Their parsimony allows them to make a better use of the available data than non-parsimonious approximators (such as polynomials) do.

3. DYNAMIC NEURAL MODELS

3.1. Dynamic models

The previous section introduced *static* models, which are algebraic equations which account for the relations between the inputs and the outputs of static processes (or of dynamic processes in their steady state). In the rest of the paper, we focus on *dynamic* models, which are expressed as a set of differential equations and algebraic equations where *time* is one of the variables. Because of the pervasive use of computers both for process simulation and for process control, in which the inputs, outputs, and other relevant variables, are measured and/or computed at discrete instants of time, we focus on *discrete-time models*, which are governed by *difference equations* (also termed *recurrent equations*).

3.2. Feedback (or recurrent) networks

3.2.1. Definitions

A feedback network is a network whose graph of connections has cycles; each edge of the graph is assigned a *delay*, which is a non-negative integer indicating the time necessary for the information to be transferred along that edge, expressed in a time unit which is the sampling period of the process. For a neural network to be causal, the sum of delays of the edges pertaining to any cycle must be non-zero. Figure 3 shows a feedback network (the numbers within squares are the delays associated to each connection).

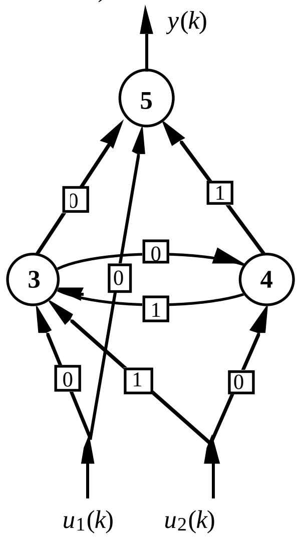


Figure 3
A dynamic network
(numbers within squares are delays)

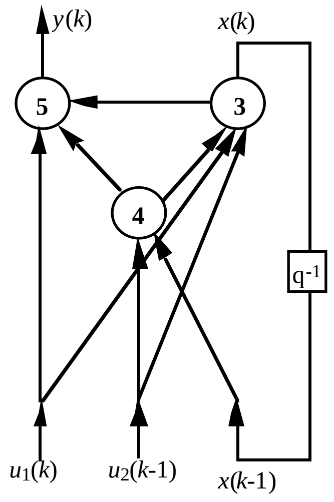


Figure 4
The canonical form of the network of Figure 3
(q^{-1} is a standard symbol for a unit delay)

Since non-zero delays are necessary in a feedback network, such a network is dynamic: if the inputs are constant, the output(s) may vary in time until a stable state (if any) is reached. Therefore, feedback neural networks are excellent candidates for modeling *dynamic* processes, just as feedforward nets are excellent candidates for modeling *static* data.

3.2.2. The canonical form of feedback neural networks

The most general form of feedback networks, called the *canonical form*, is the following

$$\begin{aligned} x(k+1) &= \varphi\,[x(k), u(k)] \\ y(k+1) &= \psi\,[x(k+1)] \end{aligned} \qquad (1)$$

where φ and ψ are two non linear functions, and $x(k)$ is a vector called *vector of state variables*; it is the vector, *having the smallest number of components*, which must be known at time kT in order to be able to predict the state vector and the output vector at time $(k+1)T$ if the inputs are known at kT (T is the sampling period).

The canonical form is thus composed of two feedforward nets implementing functions φ and ψ, the outputs of the first feedforward net (the state variables) being fed back to its inputs through unit delays. In general, the canonical form is made of a single neural net whose outputs are the output vector of the model and the state variables (the latter being fed back to the input). Note that all, or some, output variables may be state variables. Figure 5 shows the most general canonical form.

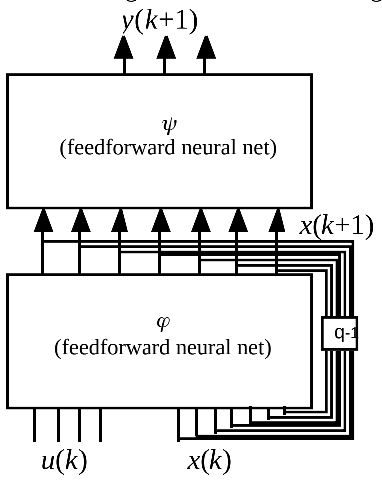


Figure 5

In practice, the two feedforward neural nets are generally lumped into a single network whose inputs are the state inputs $x(k)$ and external inputs $u(k)$, and whose outputs are the outputs of the model $y(k+1)$ and the state outputs $x(k+1)$. It will be shown in section 4, that *any neural network, however complex, can be transformed into a canonical form.*

Figure 4 shows the canonical form of the network of Figure 3.

3.3. The challenge of dynamic modeling

A process has

- *control inputs*, represented, at time kT (where T is the sampling period), by a vector $u(k)$
- *outputs*, represented, at time kT, by a vector $y(k)$,
- *unmeasured disturbances* which may be deterministic or random, such as disturbances of the state of the process (e.g. a cold object of unknown temperature introduced at some unpredictable time into an oven), or of its output (such as noise on the measurement of the output).

The objective of dynamic modeling is the following: find a model such that its response to a control signal be the same as the response that the process would exhibit *in the absence of disturbances*.

3.4. Issues in the design of a dynamic neural model

The basic issue in the design of a static model is the choice of the number of hidden neurons, which determines the tradeoff between the accuracy of the fit to the available data and the interpolation capability (this tradeoff is sometimes called the *bias-variance dilemma*). The design of dynamic models requires additional decisions concerning the architecture of the network.

3.4.1. *Input-output* vs. *state-space representations*

The first issue in the design of a dynamic model, whether linear or not, is that of representation. Two types of representations can be considered:

- in an input-output representation, the output of the model is a function of m past values of the external inputs and of n past values of the outputs of the model: $y(k+1) = F\,[y(k), y(k-1), ..., y(k-n), u(k), u(k-1), ..., u(k-m)]$;
- in a state-space representation, the output of the model is a function of intermediate variables (termed *state variables*), which are functions of past values of the inputs and of past values of the state variables:

  $x(k+1) = \varphi\,[x(k), u(k)]$ (state equation)

  $y(k+1) = \psi[x(k+1)]$ (output equation) where x is the vector of state variables and u is the vector of external inputs. The *order* of the model is the dimension of vector x, i.e. the number of state variables. Note that this form is identical to the canonical form defined in section 3.2.2.

For linear models, the input-output and state-space forms are strictly equivalent. For nonlinear models, this is not the case: state-space models have been shown to be more general and more parsimonious than input-output models (Levin, 1992). An input-output model is actually a special case of a state-space model, where the state variables are the past values of the output.

3.4.2. Choice of the order of the model

In the context of black-box modeling, the order of the model (i.e. the number of state variables x_i) must be chosen much in the same way as the number of hidden neurons must be chosen, either heuristically, or by making use of statistical tests.

3.4.3. Summary

The design of a dynamic neural model involves a larger number of degrees of freedom than the design of a static neural model. Therefore, in order to restrict the huge space of possible dynamic models, it is important to take guidance, whenever possible, from domain knowledge. This approach is explained in the next section.

3.5. Reducing design complexity by knowledge-based neural modeling

In contrast to black-box models, which are derived from measurements only, knowledge-based models are derived from the analysis of the process from a physical, chemical, economical, etc., point of view. Very often, a state-space model of a complex process exists and is reasonably accurate, but the approximations made in the derivation of the equations, the insufficient knowledge of some phenomena, or the uncertainties on the numerical values of the parameters of the model, make it unsuitable for the purpose that it should serve. The key idea of knowledge-based neural modeling is that, despite its shortcomings, such a model can be used as a basis for designing an accurate neural model.

The design of a knowledge-based neural model consists in:

- building and training (from simulated data) a recurrent network which obeys the same equations as the knowledge-based model,

- adding black-box neural nets wherever necessary, in order to take care of unmodeled dynamics,
- training the complete network from real data.

Assume that the state-space model is of the form:

$$\frac{dx}{dt} = f\left[x(t), u(t)\right]$$

$$y(t) = g\left[x(t)\right]$$

where f and g are known analytically, possibly with a few parameters. In a typical chemical engineering unit for instance, the number of state variables would be on the order of one (or a few) hundred.

The state equations can be discretized to

$$x(k+1) = x(k) + f\left[x(k), u(k)\right]$$

$$y\left(k+1\right) = g\left[x\left(k+1\right)\right]$$

by Euler's method (other discretization techniques can be used as well). If two feedforward neural networks can be trained to approximate functions f and g, then a network such as shown on Figure 6 obeys the same discrete-time equation as the model.

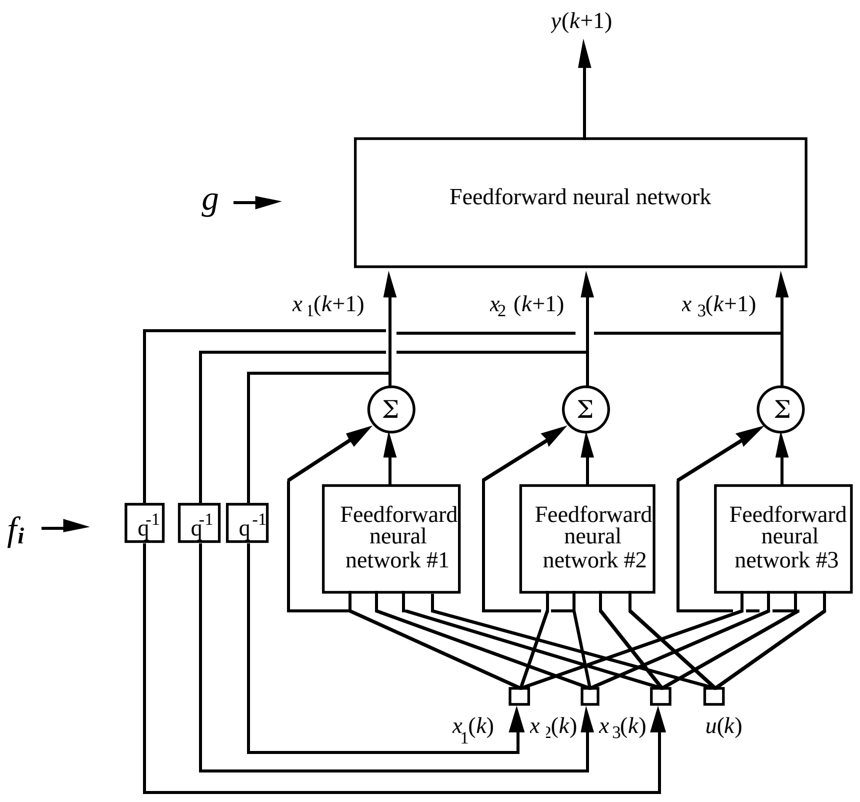
Since $x(k)$ is a vector, f is a vector too; therefore, instead of using a single network for approximating the whole vector f, it is generally advantageous to use different networks for different components f_i of f. Several situations may arise:

- function f_i (known analytically) can readily be computed: it can therefore be put simply into a "neural" form; this is a purely formal step, which is just intended to ease the implementation of the whole model as a neural network ;
- the computation of function f_i (known analytically) is time-consuming: for instance, one has

  $$\frac{\mathrm{d}x_i}{\mathrm{d}t} = f_i\left[x(k), T\left[x(k), u(k)\right], u(k)\right]$$

  where T is a solution of a nonlinear equation $\Gamma\left\{x(k), T[x(k), u(k)], u(k)\right\} = 0$; the computation of the value of $x_i(k+1)$ requires solving the second equation at each time step. In such a case, it may be advantageous to generate a set of representative sequences by solving numerically the above equations, and to use these sequences for training and testing neural network #i. Since a properly designed neural network uses a very small number of neurons, the time necessary for a trained network to compute $x_i(k+1)$ can be smaller than the time necessary for solving the above equations by several orders of magnitude. The same considerations apply to function *g*.

- function f_i is known with very poor accuracy: it can be implemented purely in a black-box fashion.



$y(k+1)$
g
Feedforward neural network
$x_1(k+1)$
$x_2(k+1)$
$x_3(k+1)$
Σ
f_i
q^{-1}
Feedforward neural network #1
Feedforward neural network #2
Feedforward neural network #3
$x_1(k)$
$x_2(k)$
$x_3(k)$
$u(k)$

Figure 6

At the end of this step, the neural network, trained from simulated data, performs exactly as well - or as poorly - as the knowledge-based model.

In the final step, the knowledge-based neural model is trained with sequences measured on the process itself. In this step, not all weights are adjustable: since most weights of the network have a physical meaning, those which are known to be accurate and not to require any adjustment are kept fixed during training. The only adjustable weights are the weights of the black-box networks (if any), and the weights whose values are not known accurately from theory.

This design technique has been successfully applied to a variety of problems such as the modeling of an industrial distillation process (Ploix et al., 1997), the automatic piloting of a four-wheel drive vehicle (Rivals et al., 1994), and the generation of animated synthetic images.

3.6. Summary: from black-box to gray-box dynamic modeling

We have shown that the design of a dynamic model of a process, whether artificial or biological, involves several issues in addition to those encountered in the design of static models (feedforward neural networks). In order to overcome this additional complexity, it is highly desirable to take advantage of whatever mathematical knowledge is available on the process, resulting from a physical, chemical, biological, ..., analysis of the process. We have described the technique of knowledge-based neural modeling, whereby domain knowledge can be built into the structure of the neural network. This procedure has several benefits:

- many weights of the resulting network have a physical meaning, so that all of them need not be trained, and the values taken on by those which are trained may give insight into the process,
- computation times may be cut by orders of magnitude, with respect to conventional differential equation solvers,
- the resulting model has the flexibility due to training, while retaining the intelligibility of knowledge-based models.

The price to be paid for these benefits is the fact that the structure resulting from the above procedure may be quite intricate, with several entangled feedback loops, which may make its training, or the analysis of its dynamics, quite difficult. This complexity may be alleviated to a large extent by transforming the network into a canonical form. This final step in complexity reduction is described in the next section.

4. COMPLEXITY REDUCTION BY CANONICAL FORM TRANSFORMATION

In this section, we summarize a procedure, presented in detail in (Dreyfus et Idan., 1998), which allows to transform any feasible dynamic neural network into a canonical form consisting (as defined in section 3.2.2) of a feedforward network whose inputs are the state inputs $x(k)$ and the external inputs $u(k)$, and whose outputs are the model outputs $y(k+1)$ and the state outputs $x(k+1)$.

We consider a discrete-time model consisting of a set of N equations of the form:

$$x_i(k+1) = \Psi_i\left(\left\{x_j\,(k - \tau_{ij,h} + 1)\right\}, \left\{u_l(k - \tau_{il,h} + 1\right\}\right) \ ,$$
$$i, j = 1, ..., N, \quad l = N+1, ..., N+N', \quad h > 0 \qquad (2)$$

where Ψ_i is an arbitrary function, $\tau_{ij,h}$ is a positive integer denoting the delay of the h-th delayed value of variable x_j used for the computation of $x_i(k+1)$, and where u_l denotes an external input. The above relation expresses the fact that the value of variable x_i at time $k+1$ may be a nonlinear function of (i) all past variables x_j (including x_i itself) and present variables (excluding x_i itself), and of (ii) all external inputs at time $k+1$ or at previous times. These equations are usually complemented by an output (or observation) equation expressing the relations between the outputs and the variables of the model.

In the context of neural networks, equations (1) may be considered as the description of a recurrent network where x_i is the output of neuron i, or the output of a feedforward neural network i, and Ψ_i is the activation function of neuron i, or the function implemented by the feedforward network i.

As a didactic example, consider a process described by the following model :

$$\ddot{x}_1 = f_1\left(x_1, x_2, x_3, u\right)$$
$$x_2 = f_2\left(x_1, x_3\right)$$
$$\ddot{x}_3 = f_3\left(x_1, \dot{x}_2\right)$$
$$y = x_3$$

where f_1, f_2 and f_3 are nonlinear functions. After discretization (by Euler's method for instance), these equations have the following form:

$$x_1(k+1) = \Psi_1\left[x_1(k), x_1(k\text{-}1), x_2(k\text{-}1), x_3(k\text{-}1), u_4(k\text{-}1)\right],$$
$$x_2(k+1) = \Psi_2\left[x_1(k+1), x_3(k+1)\right], \qquad (3)$$
$$x_3(k+1) = \Psi_3\left[x_3(k), x_3(k\text{-}1), x_1(k\text{-}1), x_2(k), x_2(k\text{-}1)\right]$$

$$y(k+1) = x_3(k+1)$$

Thus, referring to relations (1), one has $N = 3$, $N' = 1$, $\tau_{11,1} = 1$, $\tau_{11,2} = 2$, $\tau_{12,1} = 2$, $\tau_{13,1} = 2$, $\tau_{14,1} = 2$, $\tau_{21,1} = 0$, $\tau_{23,1} = 0$, $\tau_{33,1} = 1$, $\tau_{33,2} = 2$, $\tau_{31,1} = 2$, $\tau_{32,1} = 1$, $\tau_{32,2} = 2$.

The transformation of such a system to a canonical form is performed in three steps:
(i) find the order ν of the system, i.e. find the minimum number of variables $\{z_i\}$ which describe completely the model at time k if their initial values are known, and if the values of the external inputs $\{u_l\}$ are known at times 0 to k;
(ii) find a state vector, i.e. a set of ν state variables,
(iii) transform equations (2) into the state equations which govern the state variables derived in (ii).

In the above example, the equations (3) are not in a canonical form; however, a canonical form is readily derived by substituting the expression of $x_2(k+1)$ into the equations giving $x_1(k+1)$ and $x_3(k+1)$ (or, in the continuous-time model, substituting $x_2(t)$ into the expressions of the second derivatives of $x_1(t)$ and $x_3(t)$): the order of the model is 4, and the state variables are $x_1(k)$, $x_1(k-1)$, $x_3(k)$, $x_3(k-1)$ (or $x_1(t)$, $x_3(t)$ and their first derivatives). It can be proved that, for discrete-time models, these derivations and substitutions can be viewed as a sequence of graph transformations which can be performed on a computer in polynomial time. In the following, we first define the graph representation of a discrete-time model, and we subsequently summarize the steps leading to the canonical form

4.1. Graph representation of a discrete-time dynamic model

We define the graph representation of a model as a finite directed graph $G(E, V)$ consisting of a set of edges E and of a set of vertices V. Each vertex v_i represents a variable x_i. A directed edge e_{ij} from vertex v_j to vertex v_i represents a non-zero term on the right-hand side of equation i of the system of equations (3). The length of each edge is the associated delay $\tau_{ij,h}$: the number of parallel edges from v_j to v_i is equal to the number of different delays $\tau_{ij,h}$. A directed edge from v_j to v_i of length τ is denoted by $e_{ij}{}^{\tau}$ (however, for simplicity, the superscript τ will be omitted whenever the context makes it unnecessary); $\{R_i\}$ denotes the set of outgoing edges from vertex v_i, and the length of the incoming edge to v_i of maximal length is denoted by M_i. $c(v_i)$ is the number of cycles (i.e. the number of paths that start and end at the same vertex) which include vertex v_i; $c(\mathrm{e}_{ij})$ is the number of cycles which include edge e_{ij}; A_{ji} is the number of edges e_{ij} from vertex v_j to vertex v_i. Note that the dynamic system is causal if and only if the graph $G(E, V)$ does not contain any cycle of length zero. Figure 7 shows the graph representation of model (3).

4.2. Computation of the order of the model

The first step in the determination of the canonical form of the network consists in finding which variables of the model will give rise to state variables, i.e. will appear as components of the state vector $x(k)$ (for instance, in the above example, $x(k) = [x_1(k), x_1(k\text{-}1), x_3(k), x_3(k\text{-}1)]^T$: only x_1 and x_3 give rise to state variables). Therefore, we want to reduce the initial graph G_0 of the model to a simpler graph G_1 which contains only the vertices that give rise to state variables (vertices v_1 and v_3 in the above example), and which has the same number of state variables (but not necessarily the same state variables) as the model described by G_0. From this simplified graph we will be able to compute the order of the model.

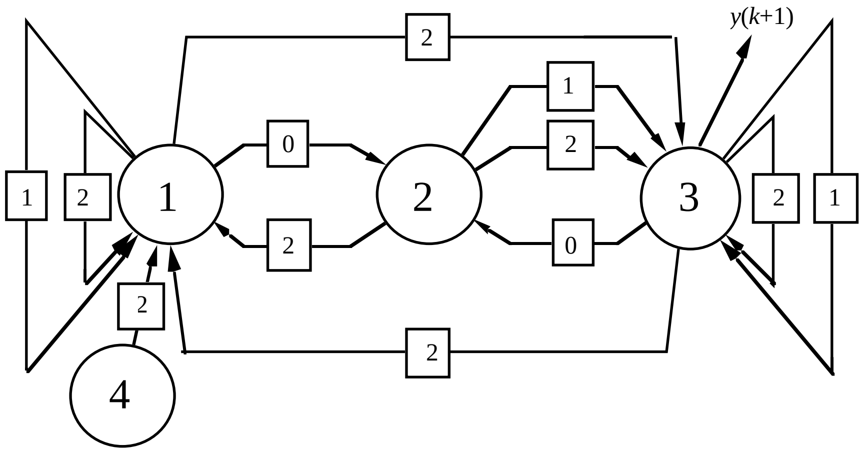


Figure 7

Graph representation G_0 of the model described by equations (3)

The simplifications of graph G_0 are the following, where G denotes the current state of the graph before and/or after the considered transformation is performed:

I. Delete all edges of G_0 that do not belong to any cycle

$$G \leftarrow G_0 - \left\{ e_{ij} \,\middle|\, c(e_{ij}) = 0 \right\}$$

and delete all isolated vertices (vertices without incoming nor outgoing edge) that may result. In principle, the present transformation is sufficient for the determination of the order. The transformations of steps II result in further simplifications of the graph, hence of the computations using the information of the graph. It can be proved (Dreyfus et al. 1998) that the present transformations can be performed in polynomial time.

II. Iterate until no change is possible:

II.1 Delete vertices whose incoming edges are all of zero length, and recombine their incoming and outgoing edges

$$\forall\ v_j, e_{ji}^{0}, e_{kj}^{\tau} \mid M_j = 0 \quad G \leftarrow G - v_j - e_{ji}^{0} - e_{kj}^{\tau} + e_{ki}^{\tau}$$

II.2 Iterate until no change is possible: if a vertex has one incoming edge only (or one set of parallel incoming edges only) and one outgoing edge only (or one set of parallel outgoing edges only) delete the vertex; if there is a single incoming and a single outgoing edge, merge the edges into a single edge whose length is the sum of the lengths of the merged edges; if there is a set of parallel incoming edges and a set of parallel outgoing edges, merge each pair of one incoming and one outgoing edges into a single edge whose length is the sum of the lengths of the merged edges

$$\forall\ v_j, e_{ij}^{\tau_1}, e_{jk}^{\tau_2} \mid\ A_{ji} \geq 1, A_{jl} = 0\ \forall l \neq i, A_{kj} \geq 1, A_{lj} = 0\ \forall l \neq k$$

$$G \leftarrow G - v_j - e_{ij}^{\tau_1} - e_{jk}^{\tau_2} + e_{ik}^{\tau_1 + \tau_2}$$

II.3 Iterate until no change is possible: if several parallel edges between two vertices exist, delete all but the edge of maximum length.

$$\forall\ v_j, e_{ij}^{\tau_1}, e_{ij}^{\tau_2} \quad G \leftarrow G - e_{ij}^{min(\tau_1,\ \tau_2)}$$

When no further change is possible, the resulting graph G_1 may be a non-connected graph.

From graph G_1, the order of the network is easily derived: the order ν of the model is given by $\nu = \sum_i \omega_i$ where

$$\forall\ v_i \in G_1 \quad \omega_i = \begin{cases} M_i \pm \min_{e_{ji} \in \{R_i\}} \left(M_j \pm \tau_{ji}\right) & if\ M_i \pm \min_{e_{ji} \in \{R_i\}} \left(M_j \pm \tau_{ji}\right) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Figure 8 shows the graph G_1 derived from the graph G_0 of Figure 7. Following the above procedure, edge e_{14}^{2} is deleted, and the output edge from vertex 3 is deleted, since they do not belong to any cycle; then, vertex 2 is deleted since all its incoming edges have zero length, edge e_{13}^{2} and two edges e_{31}^{2} and e_{31}^{1} are generated; finally, parallel edges are deleted iteratively until only edges of maximum length are left. The order of the model is easily derived: one has $M_1 = 2$, $M_3 = 2$, $\omega_1 = 2$, $\omega_3 = 2$, hence $\nu = 4$.

4.3. Determination of a state vector

The order of the model having been computed as shown above, the determination of the *graph of time constraints* leads to the determination of a state vector. The graph of time constraints G_2 is derived from the model graph G_0 by the

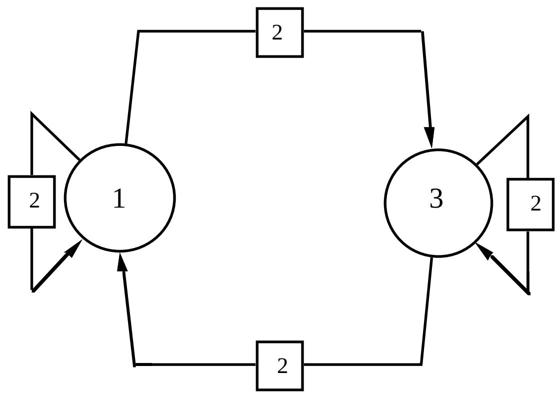


Figure 8
Graph G_1 of the model described by equations (3)

following sequence of graph transformations, which can be performed in polynomial time.

I. Initialize G to G_0. Iterate until no change is possible:

I.1 Delete all vertices whose incoming edges are all of zero length, and recombine their incoming and outgoing edges

$$\forall\ v_j, e_{ji}^{0}, e_{kj}^{\tau} \mid M_j = 0 \quad G \leftarrow G - v_j - e_{ji}^{0} - e_{kj}^{\tau} + e_{ki}^{\tau}$$

I.2 Iterate until no change is possible: if a vertex has one incoming edge only (or one set of parallel incoming edges only) and one outgoing edge only (or one set of parallel outgoing edges only) delete the vertex and merge the edges into a single edge whose length is the sum of the lengths of the merged edges

$$\forall\ v_j, e_{ij}^{\tau_1}, e_{jk}^{\tau_2} \mid\ A_{ji} \geq 1, A_{jl} = 0\ \forall l \neq i, A_{kj} \geq 1, A_{lj} = 0\ \forall l \neq k\ ,$$
$$G \leftarrow G - v_j - e_{ij}^{\tau_1} - e_{jk}^{\tau_2} + e_{ik}^{\tau_1 + \tau_2}$$

I.3 Iterate until no change is possible: if several parallel edges between two vertices exist, delete all but the edge of maximum length.

$$\forall\ v_j, e_{ij}^{\tau_1}, e_{ij}^{\tau_2}\ \ G \leftarrow G - e_{ij}^{\min(\tau_1,\, \tau_2)}$$

II. Delete all edges that do not have both vertices belonging to at least one cycle:

$$\forall\, e_{ij} \in G \mid c(v_i) = 0, c(v_j) = 0, \quad G \leftarrow G - e_{ij}$$

The variables of the model which are represented by the vertices of the resulting graph G_2 are the state variables: vertex i gives rise to state variables $x_i(k\text{-}k_i)$, $x_i(k\text{-}k_i\ \text{-}1, ..., x_i(k\text{-}k_i\ \text{-}w_i)$. Thus, two integers k_i and w_i ($k_i \geq 0$, $w_i > 0$) are associated to each vertex v_i; the computation of this set of integers is the final step of the determination of the state vector.

We denote by N_E the number of edges in the graph of time constraints. Consider an edge e_{ji} of G_2 of length τ_{ji} (Figure 9): from the very definition of the state vector, and from the construction of the graph of time constraints, it must be possible to compute $x_j(k\text{-}k_j\text{+}1)$ from one of the state variables, arising from vertex v_i, which are available at vertex v_j at time $k\text{-}k_j\text{+}1$; these variables must have been computed at vertex v_i at time $k\text{-}k_j\text{+}1\text{-}\tau_{ji}$. Therefore, the following relations must hold if $\tau_{ji} \neq 0$:

$k - k_i - w_i + 1 + \tau_{ji} \leq k - k_j + 1 \leq k - k_i + \tau_{ji}$

or equivalently

$k_j - w_i + \tau_{ji} \leq k_i \leq k_j + \tau_{ji} - 1$ (4)

Hence, a set of $2N_E$ such inequalities with $2N_V$ integer variables must be satisfied.

Thus, the problem of finding the state variables and the state equations is amenable to the following linear optimization problem in integer numbers: find the set of integers $\{w_i\}$ such that $\sum_i w_i$ is minimum (since the state representation is the *smallest* set of variables that describe the model), under the set of constraints expressed by the inequalities (4). In addition, the value of the minimum is known to be equal to ν, whose value is derived as shown in section 4.2.

Note that there is a trivial solution to the set of inequalities (4): $k_i = 0$, $w_i = \max_j \tau_{ji}$. This solution is valid if $\sum_i w_i = \nu$. Otherwise, a solution that satisfies all constraints can be found by linear optimization methods, such as the simplex (Dantzig 1963). The minimized objective function is $\sum_i w_i$ and at least one solution with $\sum_i w_i = \nu$ is known to exist. It has been proved in (Dreyfus et Idan. 1998) that the algorithm (Kuenzi et al. 1971, Press et al. 1992) converges to a solution *with integer values*, which is precisely what is needed. The solution may not be unique.

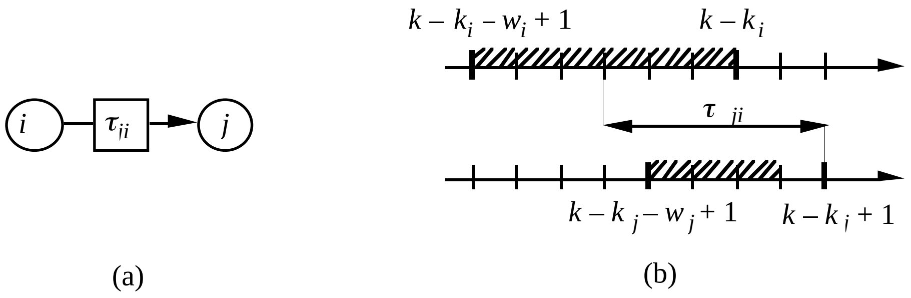


Figure 9

Once the pairs $\{k_i, w_i\}$ have been determined, the canonical network can easily be constructed. The canonical form of the network of Figure 7 is shown on Figure 10.

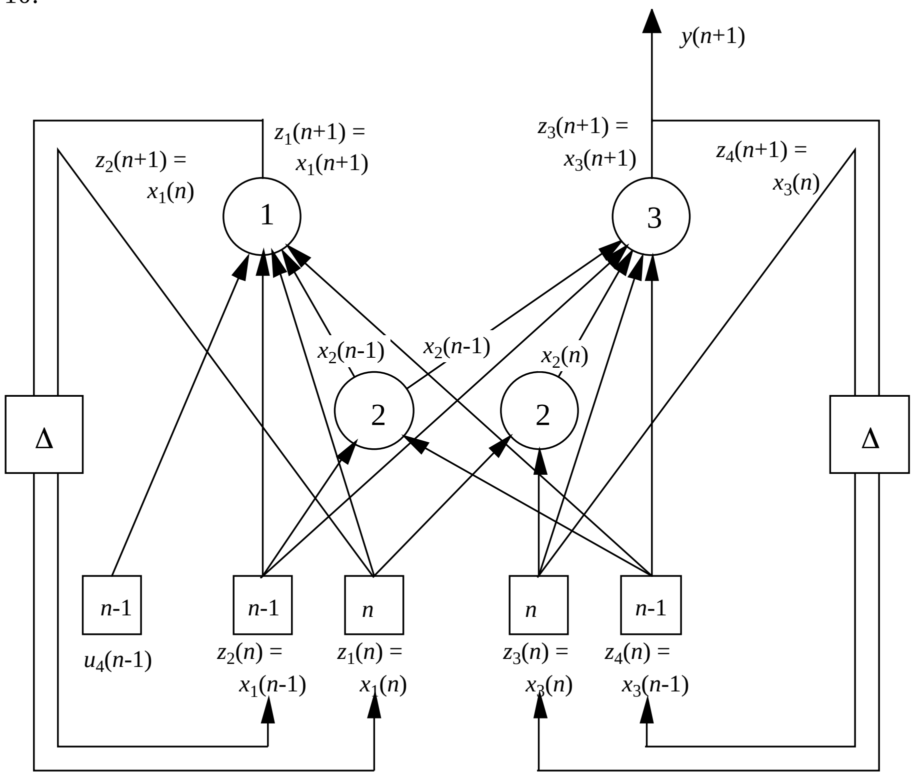


Figure 10

5. CONCLUSION

Nonlinear dynamic models become increasingly important in the field of biological modeling as well as for engineers. Feedback neural networks are excellent candidates for performing such tasks, but their design is more complex than the design of feedforward nets, since it involves a larger number of degrees of freedom. In order to reduce this complexity, two techniques are very helpful:

- the number of degrees of freedom may be decreased by making use of the available mathematical knowledge of the process to be modeled ("knowledge-based neural modeling"); this technique combines the intelligibility of knowledge-based models with the flexibility of black-box models;
- the complexity of the resulting model may be further reduced by casting it into a canonical form, which greatly facilitates the training and the analysis of he dynamics of the model.

These techniques have been proved to be useful in industrial applications, and they may also be useful in the design and analysis of models of biological neural networks such as those presented in (Quenet et al., 1998).

REFERENCES

Bishop, C. (1995). *Neural networks for pattern recognition*. Oxford University Press.

Dantzig, G.B. (1963). *Linear programming and extensions*. Princeton University Press.

Dreyfus, G. and Ydan, I. (1988). The Canonical Form of Nonlinear Discrete-Time Models, *Neural Computation, 10,* 133-164.

Hornik, K., Stinchcombe, M., White, H., Auer, P. (1994). Degree of approximation results for feedforward networks approximating unknown mappings and their derivatives. *Neural Computation 6*, 1262-1275.

Kuenzi, H.P., Tzschach, H.G., and Zehnder, C.A. (1971). *Numerical methods of mathematical optimization*. Academic Press

Levin, A. U. (1992). Neural networks in dynamical systems; a system theoretic approach, *PhD Thesis*, Yale University.

Ploix, J.L. and Dreyfus, G. (1997). Knowledge-based Neural Modeling: Principles and Industrial Applications. In F. Fogelman and P. Gallinari, eds., *Industrial Applications of Neural Networks.* World Scientific**.**

Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P. (1992). *Numerical Recipes in C : the Art of Scientific Computing,* Cambridge University Press.

Quenet, B., Dreyfus, G., Masson, C. (1998). From Complex Signals to Adapted Behaviour; a Theoretical Approach of the Honeybee Olfactory Pathway. *This volume.*

Rivals, I., Canas, D., Personnaz, L. and Dreyfus, G. (1994). Modeling and Control of Mobile Robots and Intelligent Vehicles by Neural Networks, IEEE Conference on Intelligent Vehicles (Paris).