Using Neural Networks to find the best test signal for system identification.

Joan Codina, Josep M. Fuertes.
Automatic Control and Computer Engineering Department
Email: JCodina@esaii.upc.es
Universitat Politècnica de Catalunya
Pau Gargallo, 5 E-08028 Barcelona - Spain

Abstract. In non linear systems identification or modeling there is always the problem of finding the best test signal to be used in order to make the identification in a correct way. It's obvious that in linear systems a signal rich in frequencies is good enough but this is not the case of non linear systems. In this case the behavior of the systems is not invariant to frequency or additions of different sines or cosines. We can state that best test signal will be only the one that is able to make the system evolve through the hole space of states, or at least a reasonable area of it. In this paper is presented a method that using artificial neural networks to identify non linear systems is able to optimize the test signal to be used for system identification. As soon as the definition of the test signal can not be done without a knowledge of the system behavior, an iterative process is done in order to use the already learned system dynamics to improve the test signal. The same neural network that identifies the system will be used to generate the test signal for the next training phase. At every iteration the area correctly identified grows till it reaches the space to be identified.

1. Introduction

One of the problems in the identification of non linear, time invariant systems is related with the choose of the correct input signal $u(t)$ necessary for a correct identification not only for the training signals but also all the operating situations. This problem is difficult to solve because in non linear systems there is no signal as $\delta(t)$ which is enough as input signal to identify any linear system. The choose of the correct signal is related with the systems dynamics and non lineairties, this means that if we don't know the system dynamics we can not find the optimal input, and if we don't know the optimal input we can not find the optimal model. But what it arises from the previous affirmation is that from any input signal we can find some model of the system and from some model of the system we could extract some information to improve the test signal.

The research in the field of systems identification has developed some models based on feed-forward neural networks. Such models usually take the form given by (1) and use external delay blocks for the inputs and outputs to accomplish the dynamic behavior:

$$x(k) = F \left[ u(k), u(k-1), u(k-2), \ldots, x(k), x(k-1), x(k-2), \ldots \right]$$ (1)

In Narendra[1] it is presented a very extensive work with this methodology in which different configurations are studied. The neuron activation function used in such work are of sigmoidal type and it is obtained the input-output simulation of some proposed dynamic systems.

The use of dynamic neurons or the Hopfield network [2] increases the order of the system to the number of neurons, obstructing the study of the network dynamics.

To allow simpler models for MIMO systems (Multiple Inputs, Multiple Outputs), and to allow the application of classical techniques, we have developed an ANN architecture based on the state space representation of dynamic systems, Fig 1. Such structure allows the application of modern control theory to SISO and MIMO systems. Beginning with linear discrete-time systems, we obtained a neural network in Codina[3], able to learn the matrices of the system state representation from pairs of input-output vector signals. The model was a neural network with three layers: input, state and output. The state and the output layers are connected to the input layer, composed by the actual inputs and state of the system. It was used the "back-propagation through time" learning algorithm.
Werbos[6], where the error is back-propagated from the actual state to the previous state. With this methodology, and using linear neurons, we can obtain from the system, the state space matrices $A$, $B$, $C$ and $D$:

$$x(k+1) = A \ x(k) + B \ u(k)$$
$$y(k) = C \ x(k) + D \ u(k) \quad (2)$$

This model was expanded in order to deal with non linear systems. But expanding the linear model using sigmoids impedes a theoretical study of the previous model. To solve this drawback we have used a structured neural network based on the Fourier theory in order to approximate any non linear function, within a bounded interval, by means of a weighted sum of sines and cosines, Codina[4]. In Fig. 2 we can see a neural network that calculates the Fourier coefficient of a single function.

A non linear discrete-time system can be expressed by the following difference equations:

$$x(k+1) = F( x(k), u(k))$$
$$y(k) = G( x(k), u(k)) \quad (4)$$

where $F$ and $G$ can be approximated in a bounded interval by a Fourier series.

The use of neural networks with space state description of systems allows to make an exhaustive search through the state space. But we need the neural network to do it find the input signals to do so.

**The optimal input signal.**

Before beginning to optimize the input signal, we must know which is the optimal, and after this to search the way to find it.

From the space state equations we take only the part referring to the calculation of the next state (4). And if we apply it recursively to an input function $u(k)$ then we can define $\Phi(k)$ as the output after the first $k$ terms of $u(k)$, assuming initial state $x(0)$:

$$x(k+1) = \Phi(x(0),u(0)\ldots u(k)) = F((\ldots(x(0),u(0))\ldots,u(1))\ldots,u(k)) \quad (5)$$

Then in a system with $n$ states and $m$ inputs, the optimal input function is the one that in a bounded "working" space $C$ of dimension $n+m$, makes the system evolve through it describing a curve that takes all the possible values in $C$ once and only one:
\[ u(k) \] so that \( \forall [x(k_0), u(k_0)] \in C \exists k' \text{ / } [x(k_0 + 1)] = \Phi x(k_0), u(k_0) \cdot \]

being \( x(k_0) = x_a \) and \( u(k_0) = u_a \)

This is the optimum but is impossible to have an input function to do so. What is more feasible is to have a set of points \( P \) being a regular sampling over \( C \), constructed by the cartesian product of \( P_x \) a regular formatting over the states and \( P_u \) a regular sampling over the input sub space:

\[ u(k) \] so that \( \forall [x(k_0), u(k_0)] \in P \exists k' \text{ / } [x(k_0 + 1)] = \Phi x(k_0), u(k_0) \]

being \( x(k_0) = x_a \) and \( u(k_0) = u_a \)

It is a very hard condition that the only points the systems goes through to be in \( P \) and to force the pass through the points once and only once. This condition is very important to guarantee that the learning procedure will have the same behavior all over \( C \). To make feasible the search of \( u(k) \) we could redefine the problem in order to find a set of functions that in a fixed number of steps (controllability is then assumed) brings the system from the initial state to a desired state belonging to a set of points \( P_x \) that is a regular sampling over \( C_x \) the sub space of dimension \( n \) of \( C \) that ignores the inputs.

\[ \forall \xi_a \in P \exists \xi(0), u_a(0) \text{ / } \xi(0) = \Phi x(0), u_a(0) \cdot \]

Then we can add to each \( u_a(k_0) \) all the values belonging to \( P_u \) so that we can have the output obtained when the input and state is any point belonging to \( P \):

\[ y_{a,b}(k_0 + 1) = G \Phi [x(0), u_a(0), \xi_a(k_0)]_{u_a} \bigcap u \in P_u, b \in P_u \]

Now \( y_{a,b}(k_0) \) can be used in order to train the network. But instead of training the network for all \( k \) we only do it for \( k = k_0 + 1 \), the last value. Doing so we are training the network for a set of points sampled regularly over the \( C \) space. If we use a network with a structure inspired on the Fourier Series Codina[4] then we can assume that the net is calculating Fourier coefficients corresponding to the Function \( F \), for the first \( N \) coefficients stating we have sampled \( 2N \) points for each dimension in \( C \).

**Obtention of \( y_{a,b} \)**

The problem is now to find \( u_a(k) \) for all the states in \( P_x \) without knowing \( F \). And of course this is not possible. But what's possible is to have and apporximation of \( F \) obtained training the neural network with a pair of input-output signals. Then we obtain an estimation of the state space and we use this one to obtain the approximations to \( u_a(k) \).

So let’s imagine that we have trained the network with a pair of input-output signals, and that the network has learnt the system behavior. If we now suppose that we have a first order, one input system, with \( F \) being known. We can draw the state trajectory due to the input \( u(k) \) , Fig 3. And we
assume that if the system has learned to simulate this input-output pair of signals then it approximates the function $F$ in an area surrounding the trajectory, we call $P_0^i$ this first approximation. If we now use $P_0^i$ to calculate $i_u(k)$ and then we use the resulting outputs of the system to train the network. We obtain the new function $P_1^i$, which approximates the new $F$ in a wider area, because the $i_u(k)$ are good in all the area were $F$ was correctly approximated.

**Obtention of $i_u(k)$**

The problem is now to find $i_u(k)$ for all the states in $P_X$. Without knowing $F$. Of course this is not possible. But what's possible is to have an approximation of $F$, obtained training the neural network with a pair of input-output signals. Then we obtain an estimation of the state space transfer function, inside the neural network used to learn the system dynamics. If we now take the neural network who has been trained and learned $P_0^i$, to find the sequence of inputs $i_u(k)$ that brings the state to $x_a$, to be precise to the neural network estimated state $x^e_a$.

The way to do it is building the structure of neural networks shown in fig. 4. Where NetSim is the system simulator that implements the function $P_0^i$, and is used to train the network NetImp. This network is composed by a linear neuron with $k_0$ inputs, and at each time $k$ only the $k$th input is activated with a value of 1.0. After $k_0$ sample times, the state is compared with the desired state and the error is backpropagated. We use the backpropagation through time algorithm to find the way that each of the inputs influence the output through the state:

**Example**
**Conclusions**

In the identification of non-linear, dynamic, time invariant systems, the choice of the right input-output pair of signals to use in the process of identification is crucial. As soon there is no prior knowledge of the system to identify, there is no possibility to calculate, a priori, the best test signal to be used. A good test signal is that one who makes the system state evolve through all the state space. The use of ANN to optimize the input signal to improve the identification on non-linear, dynamic, time invariant systems, is a new feature to be explored. In this paper the basic methodology to be used is presented. The use of ANN has been showed through an example with a first order non-linear system. The results encourage us for the search of most theoretical studies on that subject to study the properties of the methodology, as convergence or number of steps, and also the kind of systems is applicability is feasible.

**References**