Identification of dynamical systems with friction using neural networks

Identification of Dynamical Systems with Friction Using Neural Networks

Report no. 96075
A research report by M.M.J Houben
Tutor: dr.ir. M.J.G. v.d. Molengraft

May 1996
Division of Control Engineering
Faculty of Mechanical Engineering
Eindhoven University of Technology

Abstract

This study examines the use of neural networks for prediction of dynamical systems. After a brief introduction to neural networks and their architecture, a function was approximated by a neural network, introducing the backpropagation learning algorithm. Because of the slow convergence, this learning algorithm was substituted by the superior Levenberg-Marquardt training technique.

A neural network was used to identify a second-order linear dynamical system (a double mass-spring-damper system without friction). It was shown that this system can easily be identified with a prediction type network without hidden neurons. The linear dynamical system was transformed into a non-linear dynamical system by adding friction. This system can be identified with a neural network with only a few hidden neurons.

Finally a control application for this system has been examined. If a neural network has to estimate the force needed to realise a prescribed displacement of mass 1, this displacement is attained with a small error. However, the calculated force fluctuates between large positive and negative values. Additionally the network is able to give an estimation of the displacement of mass 2, which can be used to implement a control law.

Contents

1. An introduction to neural networks

A neural network is a parallel, interconnected network of elementary units called neurons. A feedforward neural network called perceptron is made up of layers of neurons with connections between neurons of successive layers. The intermediate layers between the output layer and the network's input are called hidden layers. All the inputs to each neuron combined determine the internal state of the neuron named activation. The combining function, which defines how to accumulate the inputs, is usually a simple addition of the inputs and a bias, the threshold of the neuron. The activation function or transfer function relates the output of the neuron with its activation.

The performance of the network can be improved by training it with new information. This is accomplished by modifying the interconnection strengths among neurons called weights. The bias of a neuron can be regarded as an input unit with a constant value of minus one together with a variable weight value. The "knowledge" of the network lies therefore in its weight values.

The next section deals with some choices that have to be made when constructing a neural network. In section 3, a neural network has been used for function approximation. At first, the backpropagation learning algorithm has been applied, but later on it was substituted by the Levenberg-Marquardt optimization algorithm. Section 4 and 5 deal with a dynamical system without and with friction, respectively.

2. The network architecture

2.1 Number of layers

Cybenko [4] has shown that a neural network with at least one hidden layer (containing sigmoidal non-linearities) can represent any arbitrary function, i.e. it can model any continues non-linear transformation. With more hidden layers the network learns faster, but training of the network becomes much more complex and therefore results in longer computation times. Moreover, too many hidden layers may produce local minima on the error surface. A reasonable choice is therefore a neural network with one hidden layer. Hence in this study the neuron structure of the network will be: input units → hidden neurons → output neurons.

2.2 Number of neurons

The nature of the application, in this case identification of a dynamical system, determines the number of output neurons. The number of input neurons depends on the system to be identified as well as the number of past outputs one wishes to use as input (which actually depends on the implemented system too). The more neurons in the hidden layer, the more likely it is that the system learns to reproduce the input-output combinations. However, the use of more hidden neurons results in longer computation times when training the network. Too few neurons may cause "underfitting". The network is only able to fit a part of the whole training set. On the other hand, too many neurons may result in "overfitting". All training combinations are fitted well, but when tested with a not-trained combination, the neural network produces an output that differs greatly from the target output.

2.3 Dynamical representation

The system to be considered is dynamical. Because of this time dependent behaviour one has to train the neural network not only with the current data but also with the past data. Therefore it will be necessary to use previous outputs as new inputs. This can be accomplished in two different ways [9], [3]:

① One possibility is to use a feedforward network and take *the past measurement outputs of the system* as inputs to the network. This is often called a "series-parallel identification model" or "prediction model".

② The other possibility is to use a recurrent network. As current inputs to the network, *the estimated outputs from the network* are taken. This is often called a "parallel identification model" or "simulation model".

These two schemes that establish the network architecture are visualised in Figure 1.

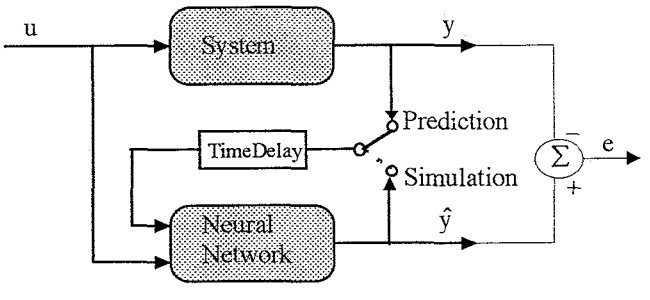

Figure 1: Prediction and simulation model of system identification.

In contrast to the prediction model, the stability of the simulation model cannot be assured. There is no guarantee that the parameters will converge or that the output error e (this is the difference between the estimated output of the neural network $\hat{y}$ and the desired or real output of the system y) will tend to zero [9]. Furthermore no previous estimated outputs are initially known.

Since no feedback loop exists in the prediction model, a less complex training technique can be used to adjust the parameters, reducing the computation time considerably compared to the simulation model. Hence in this study for neural network identification the prediction model will be used.

2.4 Training data

An input presented to the neural network results in an estimated output $\hat{y}$. A neural network "learns" by comparing this estimated output with the desired output y. On the basis of the difference e between these outputs the parameters of the network will be adapted. The input-output combinations can be presented to the network in two different ways:

① One possibility is to present the patterns (input-output combinations) one by one and to adapt the network parameters immediately after each pattern. This is called "pattern mode" or "pattern learning".

② Another possibility is to present all combinations of the training set (or epoch) as a whole batch to the neural network. The network is updated after the entire batch of data is processed. This is called "batch mode" or "batch learning". Generally a network in batch mode learns faster. Hence this mode will be used in this study.

2.5 Activation functions

By using a non-linear function as activation function (transfer function) in the hidden layer, the neural network can also represent non-linear systems. The function most often used in neural networks, due to its smoothness and differentiability, is the sigmoid function. Usually two kinds of sigmoid functions are used (in this study the second one):

① The logistic (unipolar/ binary/ logsig) transfer function with a range of 0 to 1:

$$\varphi(u) = \frac{1}{1+e^{-\alpha u}} \qquad (\alpha \text{ is the "slope parameter", frequently chosen 1})$$

② The hyperbolic tangent (bipolar/ tansig) transfer function with a range of -1 to 1:

$$\varphi(u) = \tanh(\alpha u) = \frac{1-e^{-2\alpha u}}{1+e^{-2\alpha u}} = \frac{e^{2\alpha u}-1}{e^{2\alpha u}+1}$$

By using a linear function,

$$\varphi(u) = u$$

in the output layer, the output of the neural network is not limited to absolute 1, but can equal any desired limited value. The input layer usually acts as an input data holder which distributes inputs to the hidden layer. Then the neurons in the input layer do not need an activation function.

When training the network with a specific training technique, more aspects have to be considered. For example, when using backpropagation the learning rate is an important training parameter. Some of these aspects will be discussed while explaining the training algorithm (in the appendices).

To understand a little of what is going on in neural networks it is useful trying to approximate a function with a neural network. This is the main goal of the following section.

3. Function approximation

A multilayer neural network may be viewed as a function approximator. An input to the neural network results in a specific output depending on the network architecture and weight values. Given a function to be approximated, multiple desired input-output pairs can be constructed. For every input a known output target exists. The neural network simply has to learn to successfully associate the input with the target. This learning means that the neural network has to find such weights (and biases) for which the output errors for al input-output pairs are minimal, or at least acceptably small.

3.1 Least squares minimization

When an input pattern is presented to the neural network, it propagates forward through the successive layers. Each neuron computes its output value according to the equation

$$x_j^l(k) = \varphi^l\left\{v_j^l(k)\right\} = \varphi^l\left\{\sum_{i=1}^{N_{l-1}} w_{ji}^l x_i^{l-1}(k) + b_j^l\right\}$$

where $x_j^l(k)$ denotes the kth output of neuron j in the lth layer and w_{ji}^l the weight of the connection between the ith neuron in the $(l\text{-}1)$th layer and the jth neuron in the lth layer, b_j^l is the threshold or bias of the jth neuron in the lth layer and N_{l-1} is the number of neurons in the $(l\text{-}1)$th layer. The function $\varphi^l\{\cdot\}$ is the activation function of the lth layer (described in paragraph 0) and $v_j^l(k)$ is the activation of neuron j in layer l.

Regarding the bias as a weight together with an input equal to minus one, this equation can be written as

$$x_j^l(k) = \varphi^l\left\{\sum_{i=0}^{N_{l-1}} w_{ji}^l x_i^{l-1}(k)\right\}$$

where the weight w_{j0}^l (corresponding to the fixed input $x_0^{l-1} = -1$) equals the bias b_j^l applied to neuron j in layer l.

For a network with one hidden layer and a linear activation function in the output layer the kth value of output j of the network is given by

$$\hat{y}_j(k) = \sum_{i=0}^{N_1} w_{ji}^2 \varphi^1\left\{\sum_{h=0}^{N_0} w_{ih}^1 x_h^0(k)\right\}$$

The numbers 0,1 and 2 as well as the dummy variables h, i and j refer to the input layer, hidden layer and output layer respectively. Thus N_0 refers to the total number of network inputs (excluding the threshold) and $x_h^0(k)$ is the kth value of the hth input to the network.

The error signal at the jth output neuron when presenting the neural network with the kth input-output combination is defined by

$$e_j(k) = y_j(k) - \hat{y}_j(k)$$

The error criterion which is generally used in the training of neural networks is the sum of squared errors. For a network presented with K pairs of desired input-output combinations and N_2 outputs, the instantaneous error function for the kth input-output combination is

$$E(k) = \tfrac{1}{2}\sum_{j=1}^{N_2}\{e_j(k)\}^2 = \tfrac{1}{2}\sum_{j=1}^{N_2}\{y_j(k) - \hat{y}_j(k)\}^2$$

or in vector notation,

$$E(k) = \tfrac{1}{2}\|\underline{e}(k)\|^2 = \tfrac{1}{2}\|\underline{y}(k) - \underline{\hat{y}}(k)\|^2$$

where

$\underline{e}(k)$ is the kth output error vector (belonging to the kth input-output combination) with components $(e_1(k),\cdots,e_{N_2}(k))$

$\underline{y}(k)$ is the kth desired output vector (the output of the real system) with components $(y_1(k),\cdots,y_{N_2}(k))$

$\underline{\hat{y}}(k)$ is the kth estimated output vector (the output of the neural network) with components $(\hat{y}_1(k),\cdots,\hat{y}_{N_2}(k))$.

$\|\cdot\|$ is the Euclidean norm; $\|\underline{x}\|^2 = x_1^2 + \cdots + x_n^2$.

The average squared error is obtained by summing $E(k)$ over all k's (the entire training set with K combinations) and then normalizing with respect to the set size K, as shown in vector notation by

$$E_{\text{av}} = \tfrac{1}{K}\sum_{k=1}^{K} E(k) = \tfrac{1}{2K}\sum_{k=1}^{K}\|\underline{e}(k)\|^2$$

This error criterion is a function of all the free parameters (i.e., weights and biases) of the network. The objective of the learning process is to adjust these free parameters so as to minimize E_{av}. This minimization can be done with several optimization techniques.

3.2 Backpropagation

An often used optimization technique is "backpropagation", see appendix A. The function to be approximated with the backpropagation learning algorithm is

$$y = \tfrac{1}{2}x^2 - 2x + 5$$

Together with the choices made earlier, this leads to a neural network with three layers. An input layer with one input unit (receiving x) and no activation function, a hidden layer with tangent sigmoid activation functions and an output layer with one output neuron (giving y) and a linear activation function. The number of neurons in the hidden layer will be varied. When using only the simple form of backpropagation, the network learning is very slow. Using backpropagation with adaptive learning rate and addition of a momentum term the network learns less slowly but still the computation time is very long. Figure 2 shows the results of training the neural network with the improved backpropagation, after presenting it 1000 times (number of iterations or epochs is 1000) with the whole training set consisting of 21 input-output targets. The cross signs represent the target output, the different line types

represent different numbers of hidden neurons, as indicated by the legend. The improvement obtained by using more hidden neurons (but not too much) can, in this particular case, also be obtained by increasing the number of training epochs. In general, a minimal number of hidden neurons are required.

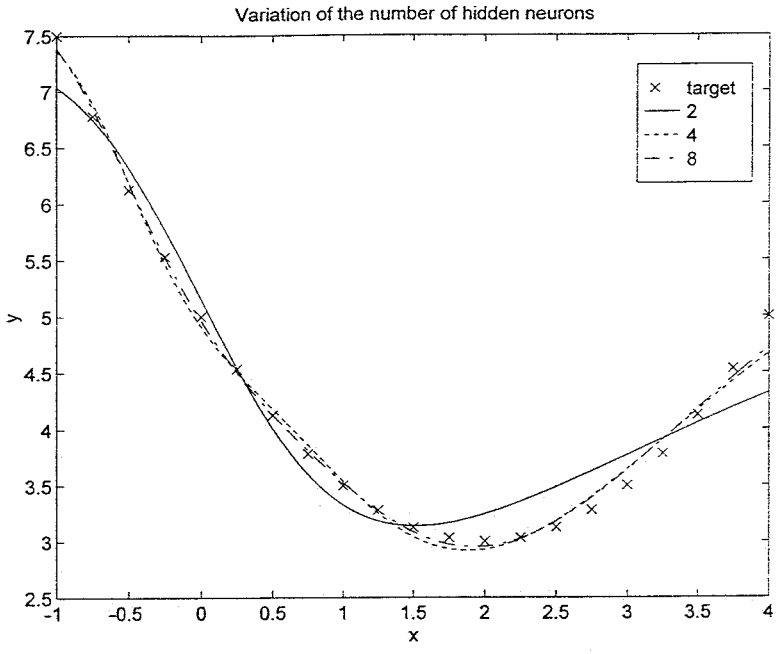

Figure 2: Function approximation using the backpropagation algorithm.

The average squared error E_{av}, displayed in Figure 3, decreases slowly, illustrating the slow convergence of the backpropagation algorithm. Furthermore the backpropagation algorithm may become trapped at local minima and can be very sensitive to user selectable parameters.

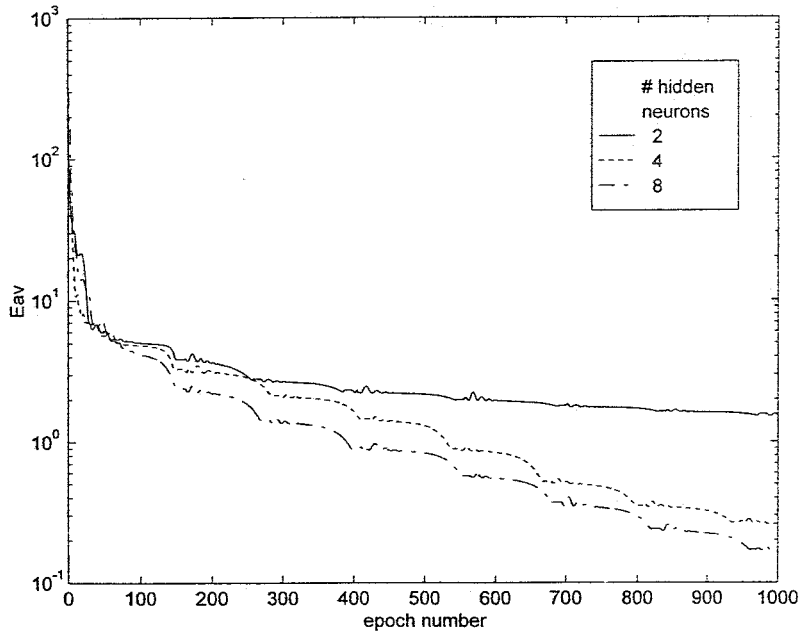

Figure 3: Training error (backpropagation).

3.3 Levenberg-Marquardt optimization

A better learning algorithm is the Levenberg-Marquardt optimization technique, see appendix B. Another attempt to approximate the function $y = .5\,x^2 - 2\,x + 5$ has been carried out with this Levenberg-Marquardt training algorithm. Figure 4 shows the results of a neural network with two hidden neurons. The training took 24 seconds for about 120 epochs. Apparently the network is appropriate to approximate this function. The network can even approximate more complicated functions (i.e. functions with more non-linearities). Figure 5 shows the decreasing average squared error during training. The Levenberg-Marquardt algorithm has superior convergence properties compared with backpropagation and will be used to train the networks of the remaining part of the present study.

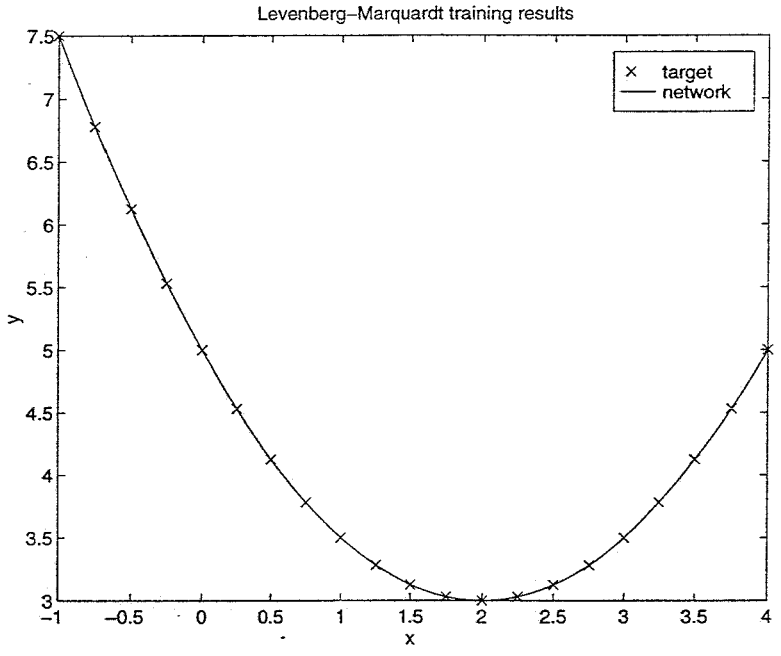

Figure 4: Function approximation using the Levenberg-Marquardt algorithm.

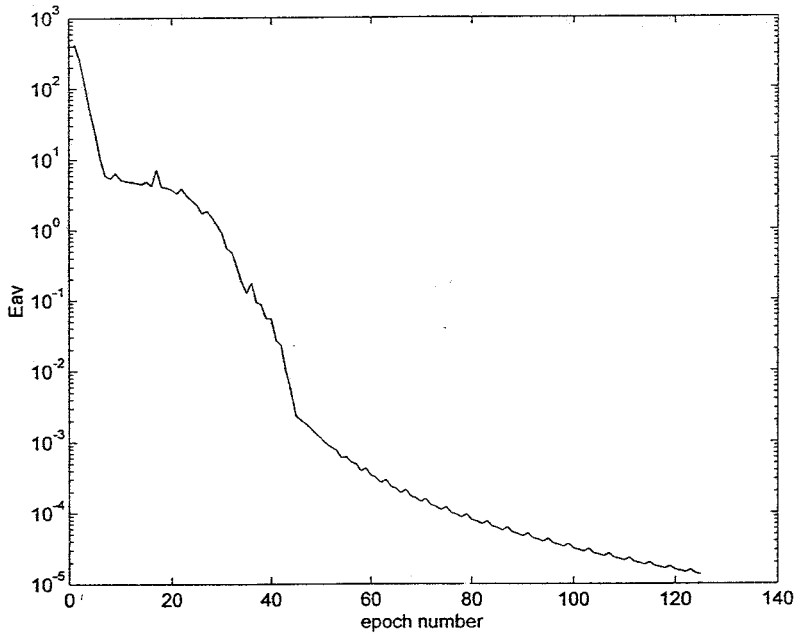

Figure 5: Training error (Levenberg-Marquardt algorithm).

4. Second-order linear dynamical system

4.1 System description

A double mass-spring-damper system, as represented in Figure 6, is to be identified with a neural network. The force $F(t)$ is the input $u(t)$ of the system. The left and right mass equal m_1 and m_2, respectively. The output of the system consists of both the displacement $x_1(t)$ of mass m_1 and the displacement $x_2(t)$ of mass m_2. The spring constants are k_1 and k_2. The damper constants are b_1 and b_2.

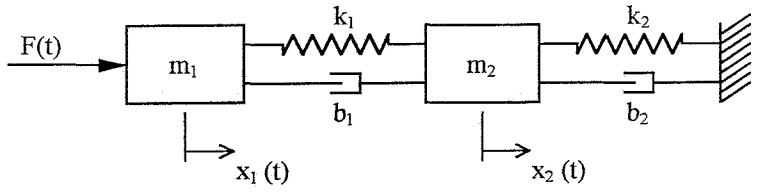

Figure 6: Double mass-spring-damper system.

The equations of motion resulting from force balance are:

$$F - k_1(x_1 - x_2) - b_1(\dot{x}_1 - \dot{x}_2) = m_1\ddot{x}_1$$

$$k_1(x_1 - x_2) + b_1(\dot{x}_1 - \dot{x}_2) - k_2x_2 - b_2\dot{x}_2 = m_2\ddot{x}_2$$

Written in matrix notation:

$$\begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix}\begin{bmatrix} \ddot{x}_1 \\ \ddot{x}_2 \end{bmatrix} + \begin{bmatrix} b_1 & -b_1 \\ -b_1 & b_1 + b_2 \end{bmatrix}\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} + \begin{bmatrix} k_1 & -k_1 \\ -k_1 & k_1 + k_2 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} F \\ 0 \end{bmatrix}$$

with

$$\begin{array}{lll} m_1 = 5 & k_1 = 25 & b_1 = 5 \\ m_2 = 3 & k_2 = 100 & b_2 = 5 \end{array}$$

The eigen frequencies of this system are:

$$f_1 = 0.32 \text{ Hz}$$
$$f_2 = 1.04 \text{ Hz}$$

4.2 The discrete form of the equations of motion

The derivative of a function can be written in various ways:

$$\dot{x} = \frac{dx}{dt} = \lim_{\Delta t \to \infty} \frac{x(t + \Delta t) - x(t)}{\Delta t} = \lim_{\Delta t \to \infty} \frac{x(t) - x(t - \Delta t)}{\Delta t} = \lim_{\Delta t \to \infty} \frac{x(t + \Delta t) - x(t - \Delta t)}{2\Delta t}$$

With omission of the limit ($\Delta t \to \infty$) the last three terms are called forward, backward, and central divided differentiation, respectively.

With discrete time k and sample frequency $f_s = 1/\Delta t$ the first and second derivatives can be approximated by

$$\dot{x}(k) = f_s\{x(k) - x(k-1)\}$$
(a backward divided differentiation)

$$\ddot{x}(k) = f_s\{\dot{x}(k+1) - \dot{x}(k)\} = f_s^2\{x(k+1) - 2x(k) + x(k-1)\}$$
(a forward differentiation on x followed by a backward differentiation on the derivative)

Combining these equations with the equations of motion leads to

$$q(k+1) = \text{function}\{q(k), q(k-1), u(k)\}$$

with $q(k) = [x_1(k) \;\; x_2(k)]^T$
and $u(k) = F(k)$

So it is shown that to be able to approximate the second-order dynamical system, the input to the neural network has to consist of the force $u(k)$ and the actual as well as the previous positions, $q(k)$ and $q(k-1)$ respectively. The output is an estimation of the next positions $q(k+1)$.

$$\text{input} = [\, u(k);\, x_1(k);\, x_2(k);\, x_1(k-1);\, x_2(k-1)\,]$$

$$\text{output} = [\hat{x}_1(k+1); \hat{x}_2(k+1)]$$

4.3 Training & testing

The used training algorithm is the Levenberg-Marquardt optimization technique described in appendix B. The neural network will be divided into two parts, one part per output. Consequently not every hidden neuron is connected with both outputs, but with only one of them. Since both outputs are indirectly connected to all the input neurons, this does not matter for the functioning of the neural network. The motive for this division is twofold. Firstly the Levenberg-Marquardt optimization programme (in Matlab) that was available is limited to only one output. Secondly this separation reduces the computation time. In Figure 7 the neural network architecture is visualised with two times three hidden neurons.

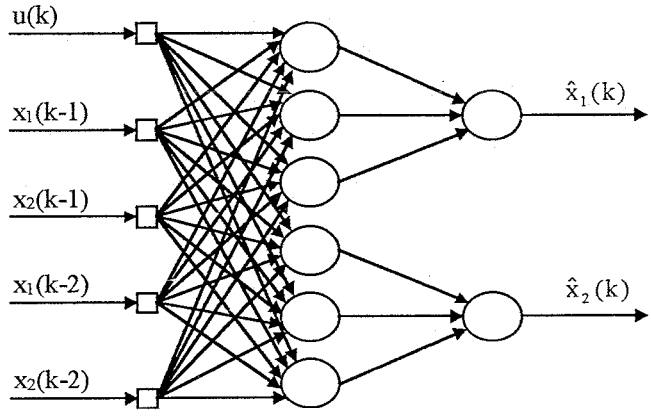

Figure 7: Neural network architecture.

For the sampling frequency f_s a value of 20 Hz has been chosen. The fold frequency (f_s / 2) has to be greater than the maximum eigen frequency f_{max} of the system (sampling theorem of Shannon). The chosen sample frequency satisfies the usually applied criterion of $f_s > 10 * f_{max}$. The number of training epochs is 400. The applied force used for training is an accumulation of several sinusoidals (see Figure 8, upper part), with the maximum frequency (= 8 / $2\pi \approx 1.3$ Hz) smaller than the sampling frequency:

$$u(k) = 10\{\sin(k) + \sin(2k) + \sin(3k) + \sin(6.5k) + \sin(8k)\}$$

The eigen frequencies of the simulated system, 0.32 Hz and 1.04 Hz, match sin(2k) and sin (6.5k), respectively. So the input signal contains all frequencies that are relevant for this system. The input u used for testing, is a block form with amplitude 10 [N] and period 10 [s] (see Figure 8, lowest part).

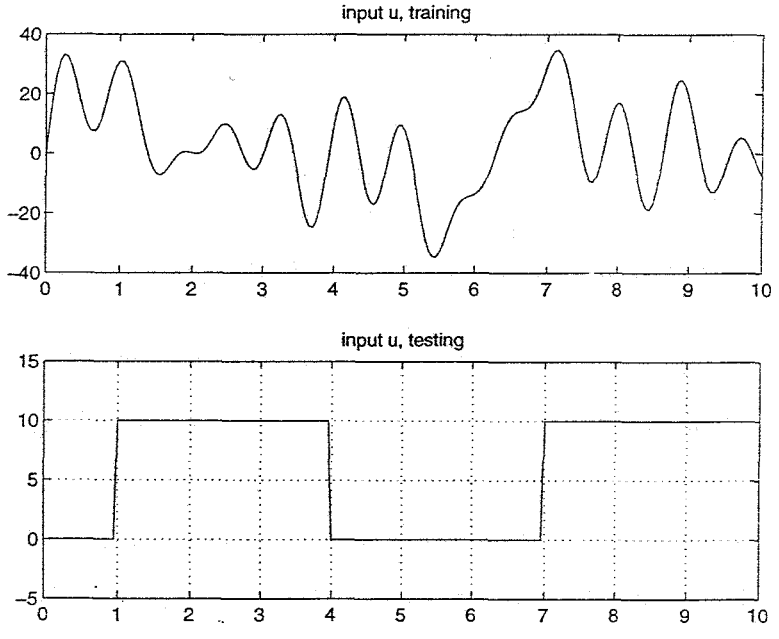

Figure 8: Input for training (upper part) and input for testing (lowest part).

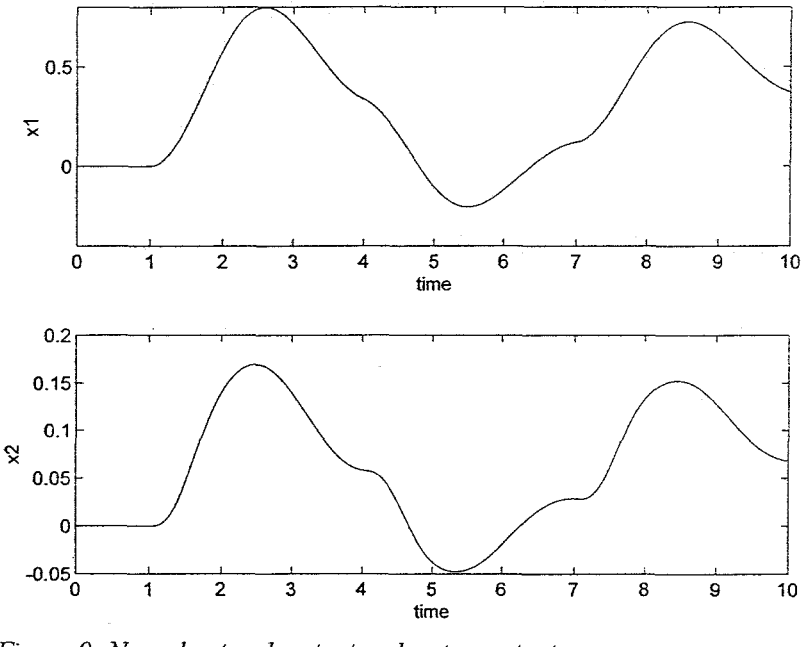

Figure 9: Neural network output and system output.

Figure 9 shows that the output of the neural network virtually equals the output of the system (the output error e is approximately zero).

Figure 10 and Figure 11 reveal that the hidden neurons that have a small input, and therefore function in their linear region, mainly establish the output of the neural network. The hidden neurons with a large input barely influence the output of the neural network. This is to be expected, because the considered discrete system is linear. A very simple network, with one or even none hidden neurons is sufficient, but only if the force u applied during testing is of the same order of magnitude as applied during training. When applying a force ten times larger than during training, the output of the neural network is structurally smaller than the real output of the system. This is a consequence of using a sigmoid (tansig) function as a linear function. Large inputs to the neural network reveal large inputs to the hidden neurons too. At these large inputs the tansig function diverges from the linear function. When using linear activation functions in the hidden layer, the output error e is approximately zero, as expected.

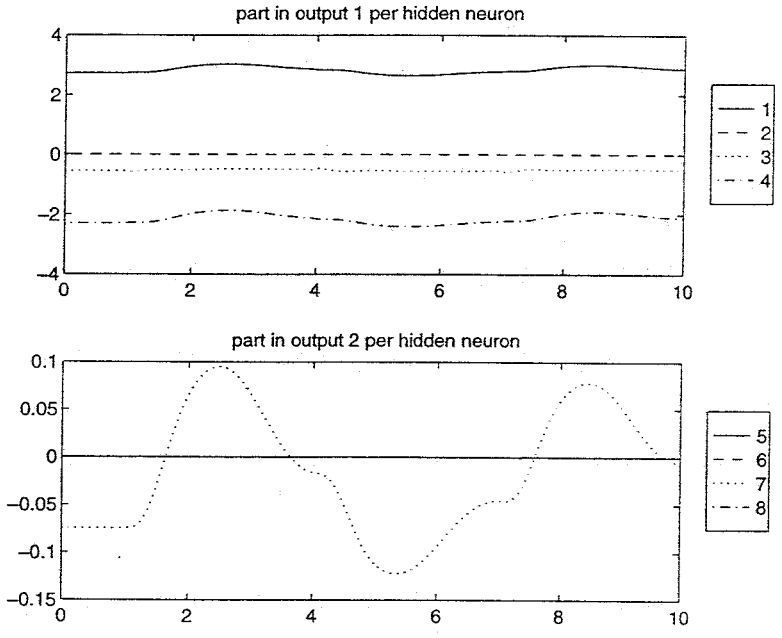

Figure 10: Part in output per hidden neuron.

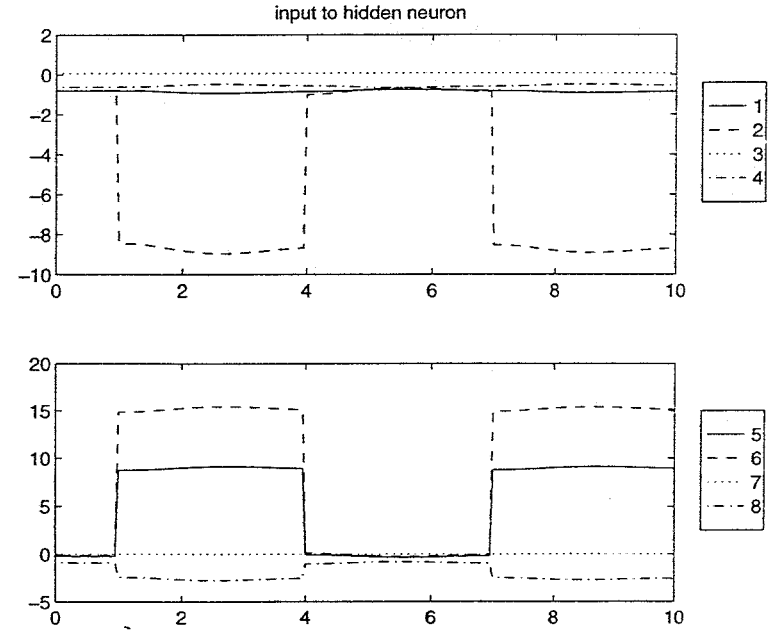

Figure 11: Input to the hidden neurons.

5. Second-order system with friction

5.1 System description

The system that will be used in this section is the same as used in section 4 (represented in Figure 6) but with the addition of friction. The friction forces working on mass m_1 and m_2 are F_{f1} and F_{f2}, respectively, see Figure 12. Taking friction into account, the system becomes non-linear. In this section the performance of the neural network identifying this non-linear system will be examined.

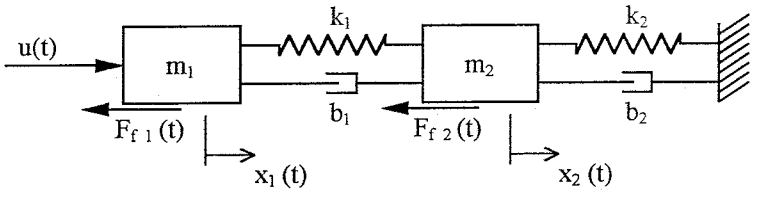

Figure 12: Double mass-spring-damper system with friction.

The equations of motion resulting from force balance are:

$$u - F_{f\,1} - k_1(x_1 - x_2) - b_1(\dot{x}_1 - \dot{x}_2) = m_1\ddot{x}_1$$

$$-F_{f\,2} + k_1(x_1 - x_2) + b_1(\dot{x}_1 - \dot{x}_2) - k_2 x_2 - b_2\dot{x}_2 = m_2\ddot{x}_2$$

For the friction force a sign function can be used:

$$F_{f\,1} = c_1 \cdot \text{sign}(\dot{x}_1) \quad \text{with } c_1 = \mu_k m_1 g$$

$$F_{f\,2} = c_2 \cdot \text{sign}(\dot{x}_2) \quad \text{with } c_2 = \mu_k m_2 g$$

with μ_k the kinetic friction coefficient (taken 0.7) and g the gravitation acceleration (taken 10 [m/s^2]). This friction model however is not easy to implement. In the static case, if the velocity is zero, the friction force equals the force acting upon the mass. In this case the friction force can be any value between plus and minus the maximum friction force $\max|F_f|$ (a point anywhere on the vertical dotted line in Figure 13) but cannot be simply calculated by solving the equations of motions. It is easier to use the hyperbolic tangent (tanh or tansig) function instead of the sign function (see Figure 13):

$$\max|F_{f\,1}| = c_1 = \mu_k m_1 g = 35 \text{ [N]}$$

$$\max|F_{f\,2}| = c_2 = \mu_k m_2 g = 21 \text{ [N]}$$

$$F_{f\,1} = 35 \cdot \text{tansig}(\dot{x}_1 \cdot 10^8)$$

$$F_{f\,2} = 21 \cdot \text{tansig}(\dot{x}_2 \cdot 10^8)$$

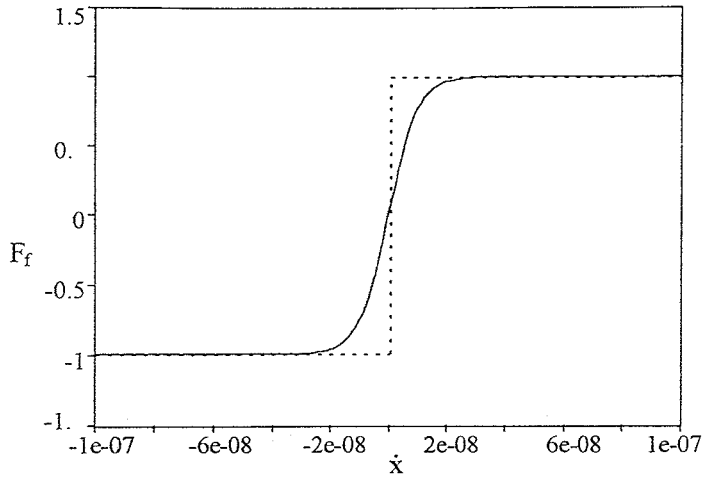

Figure 13: Friction force (normalized at maximum force) as function of velocity; implemented with $F_f = \text{tansig}(\dot{x} \cdot 10^8)$ *{solid line} and* $F_f = \text{sign}(\dot{x})$ *{dotted line}.*

When a small force acts upon the mass, it will get a very small velocity. The mass will not displace noticeably but the friction force F_f increases. F_f will equal the force exerted upon the mass. Because if F_f is smaller, $\dot{x}$ increases and therefore F_f too, until F_f balances the forces exerted upon the mass. So the net force equals zero. Only if F_f has reached his maximum $\max|F_f|$, a net force not equal to zero can be present.

Figure 14 shows the positions x_1 and x_2 resulting when an increasing force u (represented in the upper part of the figure) is applied to the system without (represented in the middle part) and with friction (lowest part of the figure). Comparison of the middle and lowest part of the figure shows the eminent influence of the implemented friction on the output of the system.

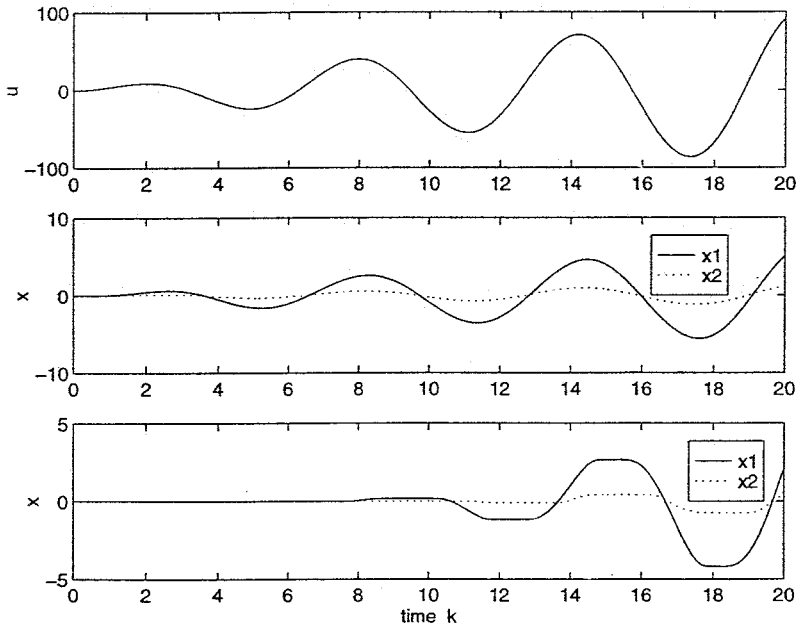

Figure 14: Influence of an increasing force (upper part) applied to the system without (middle part) and with friction (lowest part).

5.2 Training & testing

The network architecture is the same as used in section 4, dealing with a system without friction. The used training data are represented in Figure 15. The input u to the system increases with time and is an accumulation of sinusoidals:

$$u = 5k\{\sin(k) + \sin(3k) + \sin(6k) + \sin(8k)\}$$

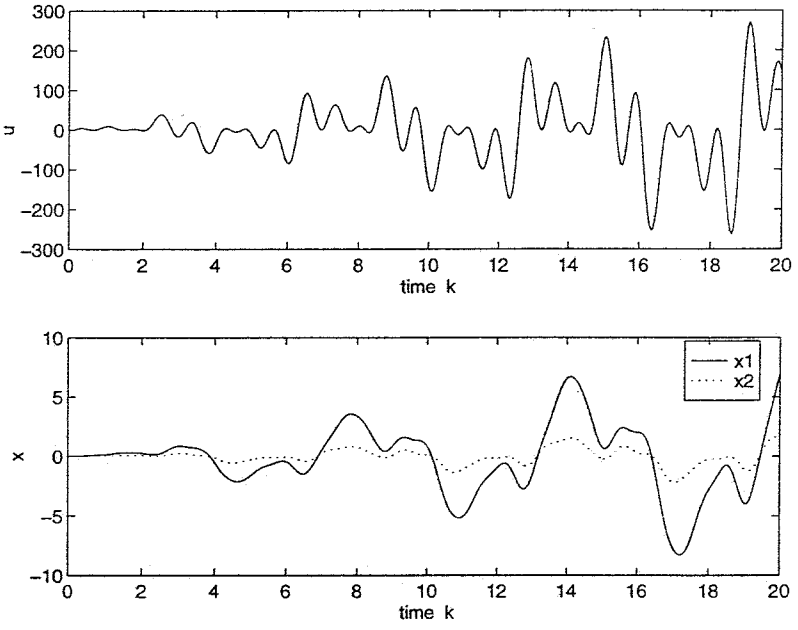

Figure 15: Input (u) to and output (x_1 and x_2) of the system with friction.

Figure 16 shows the results of testing the network with 2 times 10 hidden neurons and with the input $u = 25\{\sin(1.5k) + \cos(3k) + 1.5\sin(3.2k + 2) - 1 + k/10\}$. The network imitates the system very well, indicating that the considerable amount of friction causes no problem. Even a network with only a few hidden neurons generates a relatively small output error. This good performance is a result of the used model, namely a prediction model. The network input consists of the real positions x_1 and x_2 and only has to estimate these positions at one time step later.

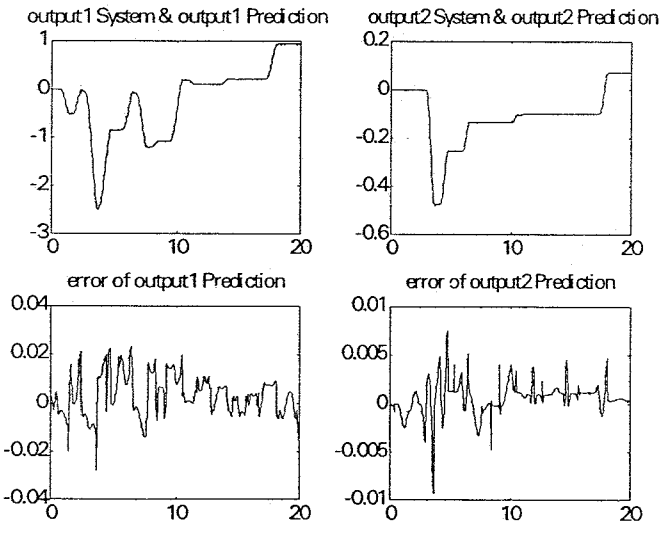

Figure 16: Neural network output (prediction model) and system output.

When using the simulation model for testing, the neural network performance is not good at all (see Figure 17), even when the input u is the same as used for training. But that is not surprising since the neural network has not been trained for simulation.

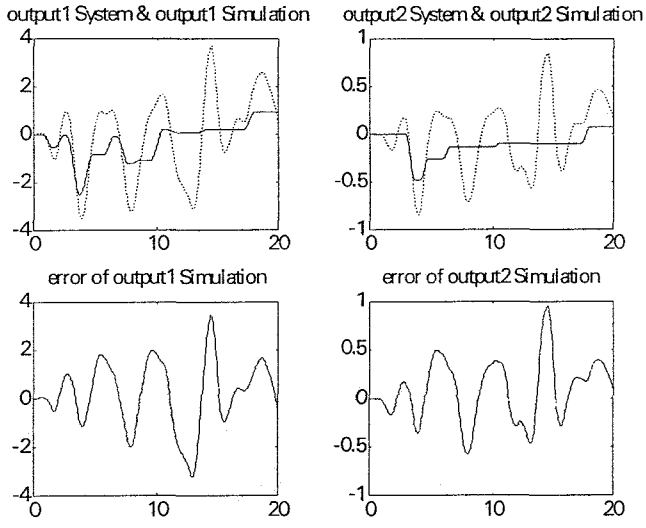

Figure 17: Neural network output (simulation model) and system output, represented by a dotted and a solid line, respectively.

5.3 A control application

An interesting application of a neural network concerning the double mass-spring-damper system with friction (represented in Figure 12) is to prescribe the position x_1 of mass m_1, x_{1d}. The neural network input consists of the present as well as the previous positions of both masses and the desired position of mass m_1 at one time step later. The network has to estimate the force u needed to realise this desired position. It is useful to construct a network that additionally estimates the position of mass m_2 at one time step later, $\hat{x}_2$. This estimated position can be regarded as the desired position of mass m_2 while mass m_1 follows a prescribed path. The reason for this additional part is the use of a control law, to be explained later.

The input to and output of the neural network are (see Figure 18):

input = [$x_{1d}(k+1)$; $x_1(k)$; $x_2(k)$; $x_1(k-1)$; $x_2(k-1)$]

output = [$u(k)$;$\hat{x}_2(k+1)$]

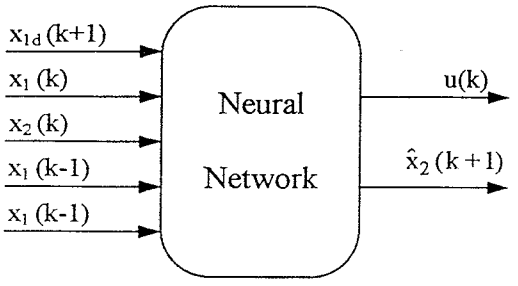

Figure 18: Neural network for control.

The neural network has been trained with two times five hidden neurons, $f_s = 20$ [Hz] and 100 iterations. The training data are obtained by computing the positions of mass 1 and 2 when a force $u = 100\{\sin(1.5k) + \cos(3k) + 1.5\sin(3.2k + 2) - 1 + k / 10\}$ is applied to the dynamical system (from $k = 0$ to $k = 20$ [s]). The resulting position of mass m_1 has been used as input to the network, the resulting position of mass m_2 and the applied force u as target output.

For testing, the sinus function $x_{1d} = 10 \sin(2k)$ has been used. Figure 19 shows the test results. The real position differs only slightly from the desired position, the error is small, except in the beginning of the test because of inaccurate initial conditions. The estimated position of mass 2 agrees very well with the real position. From this, one may conclude that the neural network functions properly. But when observing output 1 of the network (the needed force u to accomplish the desired position) it appears that the network behaves quite peculiarly. The computed force alternates between large positive and negative values, see Figure 20. The activations of some hidden neurons show the same extreme behaviour. This implies that these hidden neurons work in their saturation region (with very large absolute inputs).

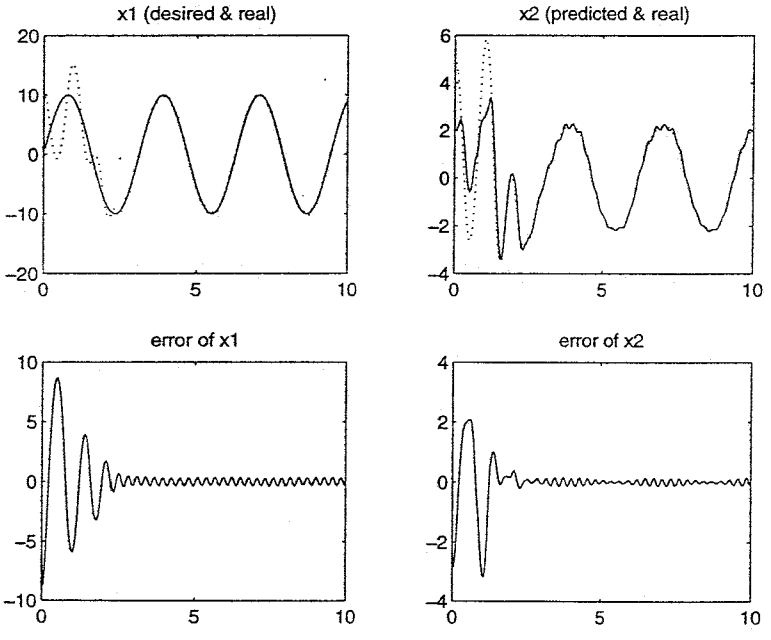

Figure 19: Desired position of mass 1, estimated position of mass 2 and real position of both masses.

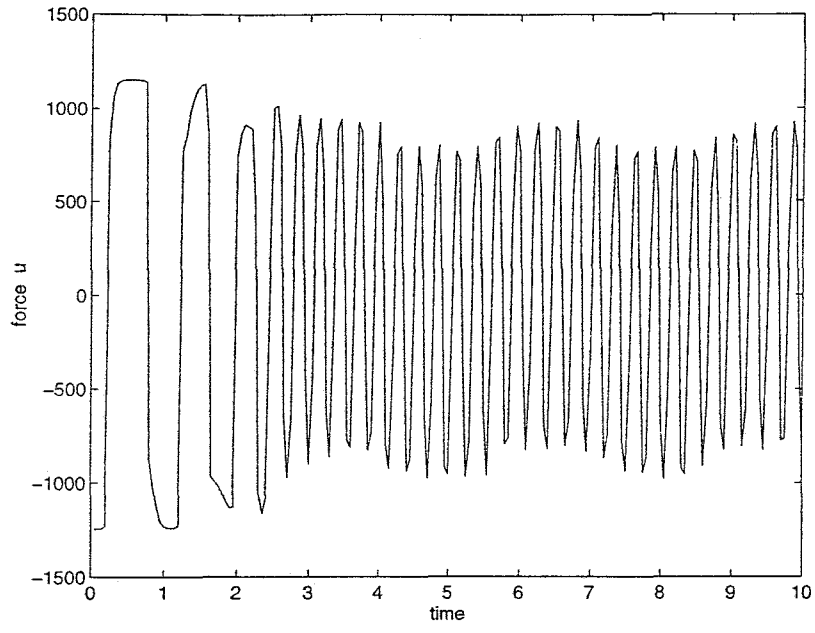

Figure 20: Needed force to realise the desired position of mass 1.

When the initial conditions of the system deviate from the desired initial conditions, a control mechanism can be applied to quickly eliminate the differences. A suitable control law is the PD-controller, implemented by

$$u(k) = u_d(k) + P_1\{x_{1d}(k+1) - x_1(k)\} + P_2\{\hat{x}_2(k+1) - x_2(k)\}$$
$$+D_1\{\dot{x}_{1d}(k+1) - \dot{x}_1(k)\} + D_2\{\hat{\dot{x}}_2(k+1) - \dot{x}_2(k)\}$$

where u_d is the needed force, to obtain the desired position of mass 1, computed by the neural network. In the previous test the initial positions were $x_1(0) = 10$ and $x_2(0) = 5$ instead of the desired $x_1 = x_2 = 0$. Therefore the desired and real positions differed in the beginning. Using the PD-control law with all coefficients equal to ten, the error in the positions disappears much faster, as shown by Figure 21.

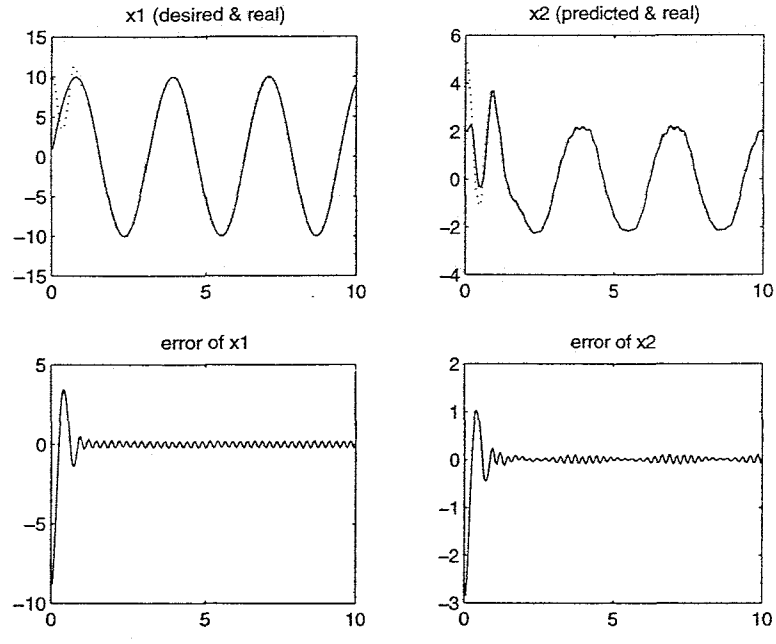

Figure 21: Desired, estimated and real positions when using a PD-control law.

6. Discussion & conclusion

For the training of a neural network, several optimization techniques are available. The technique used has much influence on the performance of the neural network. Because, dependent on the chosen training technique, the training may stop in a local minimum with a large average squared output error, or continue till a small output error is attained. An often used algorithm is backpropagation, but because of its slow convergence and its high chance to become trapped in a local point on the error surface, other training algorithms may be more satisfying and less irritating. A good alternative is the Levenberg-Marquardt optimization technique. Perhaps some modification to this technique can improve its performance even more.

A second-order linear dynamical system (a double mass-spring-damper system without friction, with as input a force working on one mass and as output the positions of both masses) can easily be identified with a prediction type network without hidden neurons. A non-linear dynamical system (the same system but with the addition of friction) can be identified with a neural network with only a few hidden neurons. When testing with a simulation model instead of a prediction model, the network output deviates strongly from the desired output. This is not surprising, because the network has not been trained for simulation. Creating a simulation type network that performs well, will be difficult, as the input to the network is not the real but the (by the network) estimated output of the system. A deviation of the network output will affect the next output. So the output error can increase with time resulting in an unstable system. In view of this problem, a neural network with a simulation model is a challenge for further research.

If the neural network has to estimate the force needed to realise a prescribed displacement of mass 1, this displacement is attained with a small error. However, the calculated force fluctuates between large positive and negative values, for some unsolved reason. Additionally the network is able to give an estimation of the displacement of mass 2. This can be used to implement a control law to eliminate deviations in the initial conditions.

In this numerical study a simulated dynamical system provided the data for training and testing the neural network. A logical continuation on this research would be to train and implement a neural network on the basis of a real system. Data obtained from real measurements introduce additional complexities like noise and measuring faults.

References

[1] Adby, P.R. and Dempster, M.A.H. *Introduction to Optimization Methods.* Chapman And Hall, London, 1974

[2] Billings, S.A., Jamaluddin, H.B. and Chen, S. Properties of Neural Networks with Applications to Modelling Non-linear Dynamical Systems. *International Journal on Control,* vol. 55, no. 1, p193-224, 1992

[3] Bulsari, A. and Saxén, H. Non-linear Time Series Analysis by Neural Networks: An Example. *IEEE Proceedings of the International Joint Conference on Neural Networks,* vol. 1, p995-998, 1993

[4] Cybenko, G. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Controls, Signals and Systems,* vol. 2, p303-314, 1989

[5] Haykin, S. *Neural Networks, a Comprehensive Foundation.* Maxwell Macmillan, 1994

[6] Fausett, L. *Fundamentals of Neural Networks. Architectures, Algorithms and Applications.* Prentice Hall, 1994

[7] Freeman, J.A. and Skapura, D.M. *Neural Networks: Algorithms, Applications and Programming Techniques.* Addison-Wesley, 1992.

[8] Kollons, S. and Anastassiou, D. An Adaptive Squares Algorithm for the Efficient Training of Artificial Neural Networks. *IEEE Transactions on Circuits and Systems,* vol. 36, no. 8, p1092-1101, 1989

[9] Narendra, K.S. and Parthasarathy, K. Identification and Control of Dynamical Systems Using Neural Networks. *IEEE Transactions on Neural Networks,* vol.1, p4-27, 1990

[10] Ojala, P., Saarinen, J. and Kaski, K. Device Modelling for VLSI Circuit Design with Technology Independent Neural Network Interface. *IEEE Midwest Symposium on Circuits and Systems,* vol. 1, p688-693, 1995

[11] Pal, C., Hagiwara, I., Kayaba, N. and Morishita, S. Dynamic System Identification by Neural Network: A New, Fast Learning Method Based on Error Back Propagation. *Journal of Intelligent Material Systems and Structures,* vol. 5, p127-135, 1994

[12] Polyak, B.T. *Introduction to Optimization.* Optimization Software, New York, 1987

[13] Qin, S.Z., Su, H.T. and McAvoy, T.J. Comparison of Four Neural Net Learning Methos for Dynamic System Identification. *IEEE Transactions on Neural Networks,* vol. 3, no.1, p122-130 , 1992

Appendix A: Backpropagation

In backpropagation the "gradient method" is used for minimizing the output error. The gradient method is an optimization technique based on linear approximation of the objective function. Say one wishes to find the w for which the function $E(w)$ is minimal. Then the gradient method gives a simple iteration formula:

$$\underline{w}(n+1) = \underline{w}(n) - \eta_n \nabla E(\underline{w}(n))$$

where

n	is the iteration number
η_n	is the stepsize (≥ 0), often $\eta_n = \eta$ for all iteration steps n
$\nabla E(\underline{w}(n))$	is the gradient of $E(\underline{w}(n))$ at a point $\underline{w}(n)$, a vector with components $\left(\frac{\partial E(\underline{w}(n))}{\partial w_1(n)}, \cdots, \frac{\partial E(\underline{w}(n))}{\partial w_r(n)}\right)$

This method is the same as choosing the "local steepest descent" (The steepest descent direction is opposite to the gradient direction.), and is often called "method of steepest descent". In the neural network application with backpropagation, $\underline{w}$ is the weight vector containing all input weights to the ith neuron of layer l (including the threshold b_i^l), $\underline{w}_i^l$ with components $\left(w_{i1}^l, \cdots, w_{iN_{l-1}}^l, b_i^l\right)$. E is the partial error function $E(k)$ or average error function E_{rev} and η is the learning rate parameter. So the basic idea is to evaluate the partial derivative of the error function with respect to the weights.

The weights are updated according to

$$w_{ji}^l(n) = w_{ji}^l(n-1) + \Delta w_{ji}^l(n)$$

with the increment $\Delta w_{ji}^l(n)$ given by

$$\Delta w_{ji}^l(n) = -\eta \frac{\partial E_{\text{av}}}{\partial w_{ji}^l} = -\eta \sum_{k=1}^{K} \frac{\partial E(k)}{\partial w_{ji}^l}$$

(This is the adjustment in batch mode, which has been used in this study. In pattern learning the summation over k has to be omitted.)

With the use of $\delta_i^l(k)$, the error signal of the ith neuron of the lth layer, which is back-propagated in the network, this equation can be transformed into

$$\Delta w_{ji}^l(n) = \eta \sum_{k=1}^{K} \delta_j^l(k) x_i^{l-1}(k)$$

The error signal at the output neurons (in layer m) is

$$\delta_i^m(k) = y_i(k) - \hat{y}_i(k)$$

and for the neurons in the hidden layer

$$\delta_i^l(k) = \frac{\partial \varphi^l(v_i^l(k))}{\partial v_i^l(k)} \sum_{j=1}^{N^{l+1}} \delta_j^{l+1}(k) w_{ji}^{l+1}(n-1)$$

The smaller the learning rate parameter η, the smaller the changes to the weights in the network will be from one iteration to the next. This smoother trajectory in weight space is attained at the cost of a slower convergence. If, on the other hand, the learning rate is too high so as to speed up the rate of learning, the resulting large changes in weights may cause an unstable network. The backpropagation method can be improved by using an "adaptive learning rate" η_n which is not fixed, but changes with the iteration n. Another method to increase the rate of learning and yet to avoid the danger of instability is to modify the weight increment by including a momentum term α_n ($0 \leq |\alpha_n| \leq 1$), which determines the influence of past weight changes on the current direction of movement in the weight space, as shown by

$$\Delta w_{ji}^l(n) = \alpha_n \Delta w_{ji}^l(n-1) + \eta_n \sum_{k=1}^{K} \delta_j^l(k) x_j^{l-1}(k)$$

The inclusion of momentum in the backpropagation algorithm tends to accelerate descent of the partial derivative $\partial E(k)/\partial w_{ji}(k)$ in steady downhill directions and has a stabilizing effect in directions that oscillate in sign. The momentum term may also reduce the chance that the learning process terminates in a shallow local minimum on the error surface instead of the desired global minimum.

Appendix B: Levenberg-Marquardt optimization

In the gradient method, the notion of local linear approximation of the objective function $f(x)$ is basic. Another approach is to use the quadratic approximation at a point $x(n)$, i.e. the function

$$f_n(x) = f(x(n)) + (\nabla f(x(n)), x - x(n)) + (\nabla^2 f(x(n))(x - x(n)), x - x(n))/2$$

where

$(\cdot,\cdot)$ is the scalar product in $\mathbf{R}^r$; $(x,y) = x_1y_1 + \cdots + x_ry_r$

$\nabla^2 f(x)$ is the Hessian matrix H with elements $\partial^2 f(x) / \partial x_i \partial x_j$

Then the iteration function is

$$x(n+1) = x(n) - \left[\nabla^2 f(x(n))\right]^{-1} \nabla f(x(n))$$

This is Newton's method.

The Levenberg-Marquardt optimization technique is a modification of Newton's method, to make it globally convergent. The iteration function is

$$x(n+1) = x(n) - \left[\nabla^2 f(x(n)) + \alpha_n \mathrm{I}\right]^{-1} \nabla f(x(n))$$

where I is the unity matrix (or unity second order tensor). For $\alpha_n = 0$ the method becomes Newton's. As $\alpha_n \to \infty$ the direction tends to the antigradient (steepest descent). Thus this formula is a compromise between these two methods.