Training of a recurrent neural network using an extended kalman filter for the simulation of dynamic systems

July 5, 1996

Training of a recurrent neural network using an Extended Kalman Filter for the simulation of dynamic systems

K.P.v.Gend

TUE, Department of Mechanical Engineering
WFW stage report 96.103

Eindhoven University of Technology (TUE)
Department of Mechanical Engineering
Division of Fundamental Mechanical Engeneering
July 1996

Contents

Training of a recurrent neural network using an Extended Kalman Filter for the simulation of dynamic systems

K.P. van Gend

July 5, 1996

Abstract

The dynamics of a mass-spring-damper system with friction is teached to a recurrent artificial neural network. The goal is to use the network as a simulation model. The output of the network is fed back to the input using two integrators.

Because a dynamic identification and reconstruction process is involved, an Extended Kalman Filter approach is used to estimate both the state of the process and the weights of the network.

Simulations on two non-linear mechanical systems show that the approach works, but costs huge numbers of calculations and produces a model which is only valid in the range of inputs on which it was trained.

1 Preface

This is the report of my apprenticeship. It is a short summary of the experiments and the theory used. In the mean time, I became familiar with UNIX, LaTeX, and the basic concepts of neural networks. So it is a succes anyway...

Thanks to `asterix.urc.tue.nl` and `sg29.wfw.wtb.tue.nl` for the calculations and Sven Pekelder and Bert van Beek for all their remarks on this article and its predecessors.

Further I must thank R. v.d. Molengraft and G.Angelis for all their remarks on the semi-final version of this report.

2 Introduction

In his M.Sc thesis [6], Edwin Verschueren uses an Artificial Neural Network (ANN) to *predict* the next $(t + \Delta t)$ state of a dynamic system. He trained his ANN using a second-order Levenberg-Marquardt algorithm. He could also have used an Extended Kalman Filter (EKF) approach as was proposed by Singhal and Wu [4]. In both cases the ANN consists of a *static* feed-forward system. Their optimization goal is formulated as follows:

$$J = \int_{t_b}^{t} (m - \hat{y})^2 d\tau; \tag{1}$$

thus the integral over time of the square error, where $m(t)$ is the output generated by the real system and $\hat{y}(t)$ is the output generated by the ANN. They calculate the estimated response by

$$\hat{y}(t + \Delta t) = m(t) + \dot{\hat{y}}(m(t), u(t)) \ \Delta t; \tag{2}$$

Thus, the *previous* state of the real system is used to *predict* the next state, as the derivative $\dot{\hat{y}}$ is calculated by using the previous measured state and the output.

In this article a **recurrent** neural network will be used. The output of this network are the accelerations of the masses in the real system. These accelerations are integrated two times and fed back to the network, as is visualized in figure 1. Note that the ANN does not have bias-nodes.

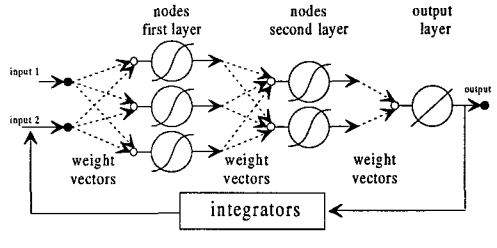

Figure 1: the recurrent network for a 1 mass-spring-damper system.

This results in a different function for the estimated state:

$$\hat{y}(t) = \hat{y}(t_b) + \int_{t_b}^{t} \dot{\hat{y}}(\tau) d\tau; \tag{3}$$

in which it is clear that only previous estimates are used to calculate the next state. These are integrated from the begin time t_b to the current time t. The output of the real system is only used to minimize the error by adapting the weights and the state estimate, using an EKF-algorithm. When training is completed, the network should be able to imitate the real system, thus eliminating the need to continuousely measure the state of the real system. It also enables the implementation of Model-based Predictive Control.

The major disadvantage of using an Extended Kalman Filter is the large number of calculations which must be conducted in order to solve the Matrix Ricatti-equations. This can be avoided by using Puskorius' *Node Decoupled Extended Kalman Filter* (NDEKF) [5]. Puskorius neglected the interdependence of weights of different nodes, resulting in several smaller matrix Ricatti-equations, which are much faster to solve. This has not been tested, because of lack of time, but Murtuza [2] has written an excellent paper providing an easy-to-use MATLAB-algorithm.

Another interesting development in the field of EKF and ANN is Obradovic's paper *Recursive Learning in Recurrent Neural Networks with Varying Architecture* [7] in which a suitably altered EKF-algorithm is used both to estimate the necessary topology of the network and its state.

3 System Equations and Kalman Filters

3.1 Basic System Equations

Using Newton or Lagrange mechanics, it is possible to describe the behaviour of a mechanical system. This results in differential equations, such as:

$$\underline{\ddot{q}} = \underline{f}(\underline{\dot{q}}, \underline{q}, \underline{u}, t); \tag{4}$$

in which $\underline{q}$ represents the positionvector of all masses of the system. For remarks concerning the notation, refer to Appendix A. By combining $\underline{q}$ and $\underline{\dot{q}}$ into one vector, the state $\underline{x}$ is created. The system is written in a *state equation*, containing all important information for the system itself:

$$\underline{\dot{x}} = \begin{bmatrix} \underline{\dot{q}} \\ \underline{\ddot{q}} \end{bmatrix} = \begin{bmatrix} \underline{\ddot{q}} \\ \underline{f}(\underline{x}, \underline{u}, t) \end{bmatrix}; \tag{5}$$

It might not be possible or desirable to measure all elements of the state. The measured state elements are called the *output*, which is generated by an *output equation*:

$$\underline{y} = \underline{g}(\underline{x}, \underline{u}, t); \tag{6}$$

If it is possible to calculate a full rank reconstructability matrix (see Kok [3, pages 3.98 and up]), it is proven that all elements of the state can be calculated when the state equation is known.

In this report, it is assumed that all systems are fully reconstructabe.

3.2 The Augmented State of a Neural Network

An Artificial Neural Network (ANN) consists of *nodes* which contain inputs, a non-linear function and an output, as seen in figure 12. In this article, a recurrent network will be used: the resulting outputs will be integrated and fed back as inputs to the network.

When an artificial neural network, as shown in figure 1, is used, the weight parameters can be combined in a vector $\underline{\phi}$. The resulting equation is (compare to equation 4):

$$\underline{\ddot{q}} = f(\underline{\dot{q}}, \underline{q}, \underline{u}, \underline{\phi}, t); \tag{7}$$

This can also be written in state space form as demonstrated before. When the weight parameters are regarded as the result of the differential equation $\underline{\dot{\phi}} = \underline{0}$, the weightvector $\underline{\phi}$ can be added to the state, thus creating the *augmented state* $\underline{x}^*(t)$ and its equations:

$$\underline{\dot{x}}^*(t) = \begin{bmatrix} \underline{\dot{q}}(t) \\ \underline{\ddot{q}}(t) \\ \underline{\dot{\phi}}(t) \end{bmatrix} = \begin{bmatrix} \underline{\dot{q}} \\ \underline{f}(\underline{\dot{q}}, \underline{q}, \underline{u}, \underline{\phi}, t) \\ \underline{0} \end{bmatrix}; \tag{8}$$

This is just a trick, done for the Extended Kalman Filter (EKF) which will be introduced in Section 3.4. The EKF will adjust all parameters in the augmented state to their correct values. In this case, it will also change the weights, thus adjusting the weights to their correct estimated values.

3.3 Linear Augmented State

In order to use a Kalman filter properly, the state equations need to be linearized. Assumed is that in a infinite small time interval the input and state can be written as a nominal value and a pertubation:

$$\underline{u}(t) = \underline{u}_0(t) + \delta\underline{u}(t) \qquad \forall \qquad t_0 \leq t \leq t_e; \tag{9}$$

In this case, the nominal value is the *previous* state in the trajectory, which means that 9 can be written as $\underline{u}(t) = \underline{u}_0(t) + \delta\underline{u}(t)$. The results are:

$$\begin{array}{rcl} \delta\dot{\underline{x}}^* & = & A(t_0)\delta\underline{x}^* + B(t_0)\delta\underline{u} + \mathcal{F}(t_0); \\ \delta\underline{y} & = & C(t_0)\delta\underline{x}^* + D(t_0)\delta\underline{u} + \mathcal{G}(t_0); \end{array} \tag{10}$$

with $A(t_0)$ to $D(t_0)$ as the Jacobi matrices:

$$\begin{array}{l} A(t_0) = \left.\frac{\partial\underline{f}(\ldots)}{\partial\underline{x}}\right|_{\underline{x}_0,\underline{u}_0}; \quad B(t_0) = \left.\frac{\partial\underline{f}(\ldots)}{\partial\underline{u}}\right|_{\underline{x}_0,\underline{u}_0}; \\ C(t_0) = \left.\frac{\partial\underline{g}(\ldots)}{\partial\underline{x}}\right|_{\underline{x}_0,\underline{u}_0}; \quad D(t_0) = \left.\frac{\partial\underline{g}(\ldots)}{\partial\underline{u}}\right|_{\underline{x}_0,\underline{u}_0}; \end{array} \tag{11}$$

The functions $\mathcal{F}(t_0)$ and $\mathcal{G}(t_0)$ are defined this way:

$$\begin{array}{rcl} \mathcal{F}(t_0) & = & \underline{f}(\underline{x}_0^*, \underline{u}_0, t_0) - \dot{\underline{x}}_0^*; \\ \mathcal{G}(t_0) & = & \underline{g}(\underline{x}_0^*, \underline{u}_0, t_0) - \underline{y}_0; \end{array} \tag{12}$$

Note that if the system would have been linearized around a stable setpoint instead of using pertubations around the previous state, these functions would have equaled zero.

3.4 The Extended Kalman Filter algorithm

Following J.J.Kok [3, Chapter 5], an optimal observer or Kalman-Bucy Filter can be derived, by introducing the reconstruction error $\underline{e}(t) = \underline{x}(t) - \hat{\underline{x}}(t)$ where $\underline{x}$ is the state of the real system and $\hat{\underline{x}}$ is the state of the neural network. See figure 2.

After introducing a measurement noise $\underline{v}(t)$ and a system noise $\underline{w}(t)$, the optimization criterion is defined as:

$$J = \mathrm{tr}\left[\mathrm{E}\{\underline{e}(t)\ \ \underline{e}(t)^T\}\right]; \tag{13}$$

which must be minimized. For the systems of J.J.Kok, this results in the matrix Riccati equations [3, eqn. 5.49 en 5.50]:
(All time dependencies are removed for simplicity)

$$\dot{\hat{\underline{x}}}^* = A\hat{\underline{x}}^* + B\underline{u} + K(\underline{y} - C\hat{\underline{x}}^* - D\underline{u}); \tag{14}$$

$$K = \tilde{Q}C^T V^{-1};$$

$$\dot{\tilde{Q}} = A\tilde{Q} + \tilde{Q}A^T + W - KC\tilde{Q}; \tag{15}$$

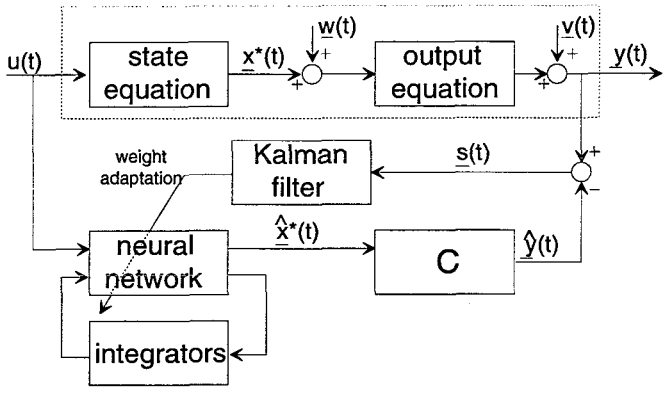

Figure 2: The Kalman-filter.

As derived in paragraph 3.3, the systems in this essay are somewhat more complex, thus requiring the functions F and G.

In the corrector $K(\underline{y} - C\underline{\hat{x}}^* - D\underline{u})$ the linearisation also requires a $\underline{y}_0(t_0) - C\underline{\hat{x}}_0^* - D\underline{u}_0$. Because in the systems discussed here no connection between $\underline{u}$ and $\underline{y}$ exists it is assumed that D equals zero. The function $m_0(t_0) - C\underline{\hat{x}}_0^*$ is also assumed zero, as $C(t_0)\underline{\hat{x}}_0^*$ should be almost a replacement for $\underline{y}_0$. The resulting equation is:

$$\delta\dot{\underline{\hat{x}}}^* = A(t_0)\delta\underline{\hat{x}}^* + B(t_0)\delta\underline{u} + \mathcal{F}(t_0) + K(t_0)\left\{\delta\underline{y} - C(t_0)\delta\underline{\hat{x}}^* - D(t_0)\delta\underline{u}\right\}; \qquad (16)$$

4 One mass-spring-damper systems

4.1 Introduction

The first system which will be considered in this article is a mechanical system with a mass m. This mass is connected to the rigid world using a spring with strength k and a damper of strength b. This is depicted in figure 3.

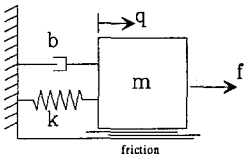

Figure 3: one mass-spring-damper system with friction.

A further possibility is the introduction of friction. In this example, only Coulomb friction is used. A simple mathematical model for Coulomb friction uses a `sign(x)` function. The disadvantage of using such a function is the bad behaviour in numerical integration-algorithms such as the second order Runge-Kutta with self-seeking stepsize which is used in this essay. The `sign`-function introduces a step in the acceleration equations, around which a self-seeking stepsize algorithm will use very small integration steps.

A workaround for this problem is to replace the `sign(x)` by a mere smooth function, such as the `tanh(n*x)` function in which n is sufficiently large, say 20. This function does not have the infinite derivative around $x = 0$.

The example will have a mass of 1 [kg], a weak spring of 0.8 [N m^{-1}] and a light damper of 0.1 [N s m^{-1}] and some friction. Its state equation is then written as:

$$\ddot{q} = -.8q(t) - .1\dot{q} - .3\tanh(20\dot{q}) + u(t); \tag{17}$$

The eigen-frequency of the system is at $\omega_0 = \sqrt{\frac{k}{m}} = 0.89\text{Hz}$.

4.2 Analytical weight calculation

The network topology to simulate this system is a network with 2×2 hidden layers and a single, linear output node (see figure 4). The weights are chosen as in figure 4.

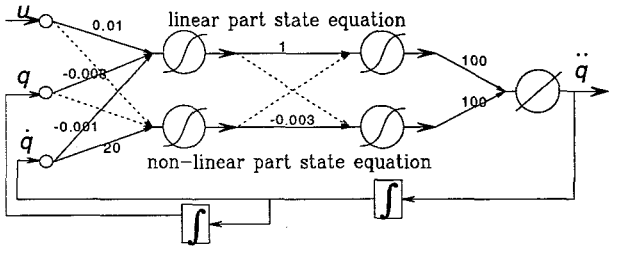

Figure 4: the topology for the analytical 1 mass-spring-damper system.

As most parts of the state equation are still linear, an approximation of the linear relation between input and output is necessary. The use of $100 * \tanh(.01x)$

(two weights and a node) approximates the linear relation of input and output for a sufficient large range of the input.

The second hidden layer is of no use in the analytic solution, but is remained to maintain a layout equal to the trained network.

This results in an analytical approximation of the state equation; written in neural nodes.

4.3 Training using the EKF

The network is trained in the timebase [0 .. 20] seconds, over and over again. Each pass through this timebase is called an *epoch*. The network is trained using either an input consisting of two sines or an input of two other sines and some noise. During the first epoch, the first is used, during the second epoch the second is used, the third time the first again, etc... This is done to prevent 'overlearning'[1]

Note that the input must excite the system enough to determine the separate parameters. This leads to two sines, the first with a frequency higher than the eigen frequency, the second having a lower frequency than the eigen frequency.

The size of the inputs is about 0.4, which is not large. The training took around 40 epochs to reach an average error of 10^{-5} per calculated point in the trajectory (see figure 5).

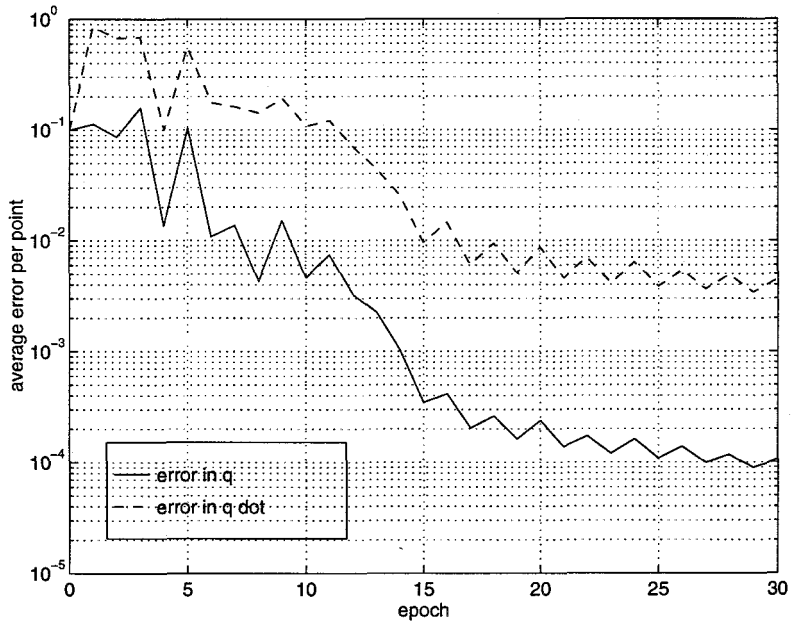

Figure 5: Error per point versus number of epoch for the position (q) and the velocity (qd)

The initial weight estimates are positive random values of about 0.3 except $w1(3,1) = 8$. This is done to help the network to estimate the large value of 20 in the friction-parameter. Refer to Section 5.5 for more details.

[1] Overlearning means that an ANN adapts itself to all phenomena in a specific dataset, including effects created by the numerical solver. This is an undesirable effect, which can be eliminated by stopping training after a reasonable time interval.

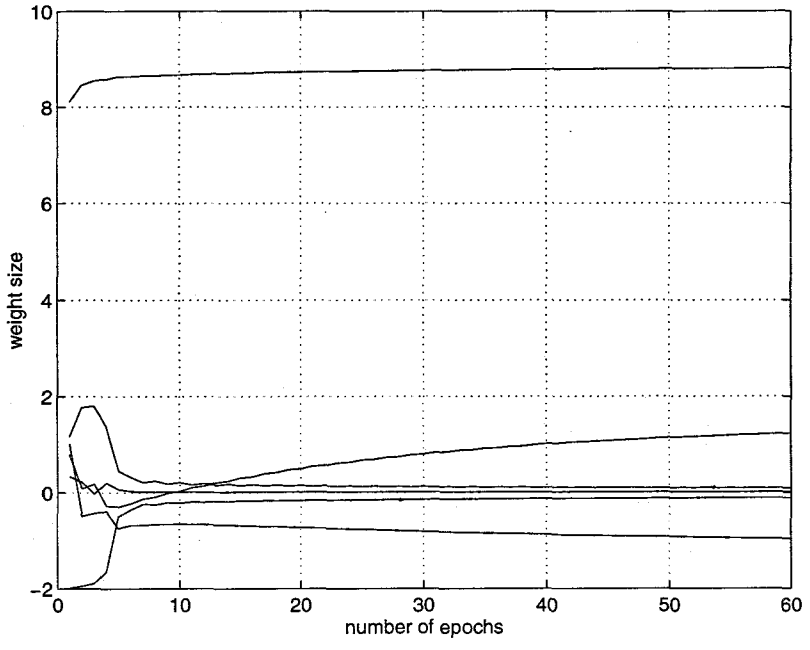

Figure 6: The weights between the inputlayer and the first hidden layer

The variation of the weights of the first layer during the training process are drawn in figure 6. Note that one of the weights is not at its stable point yet.

4.4 Comparing the results

The responses of the real system (q_s), the analytical network (q_a) and the trained network (q_n) are compared using a stepfunction as input (see figure 7).

For relative small steps, the trained ANN performs reasonably well, but larger inputs reveal a major problem concerning neural nets: they don't extrapolate well. They cannot be used outside their trained region.

Training on larger inputs resolves the problem, multiplying the input function by 3 (thus bringing it around 1.2) gives reasonable responses up to about a stepsize of 3.0 (which is more than three times the boundary of the small training.)

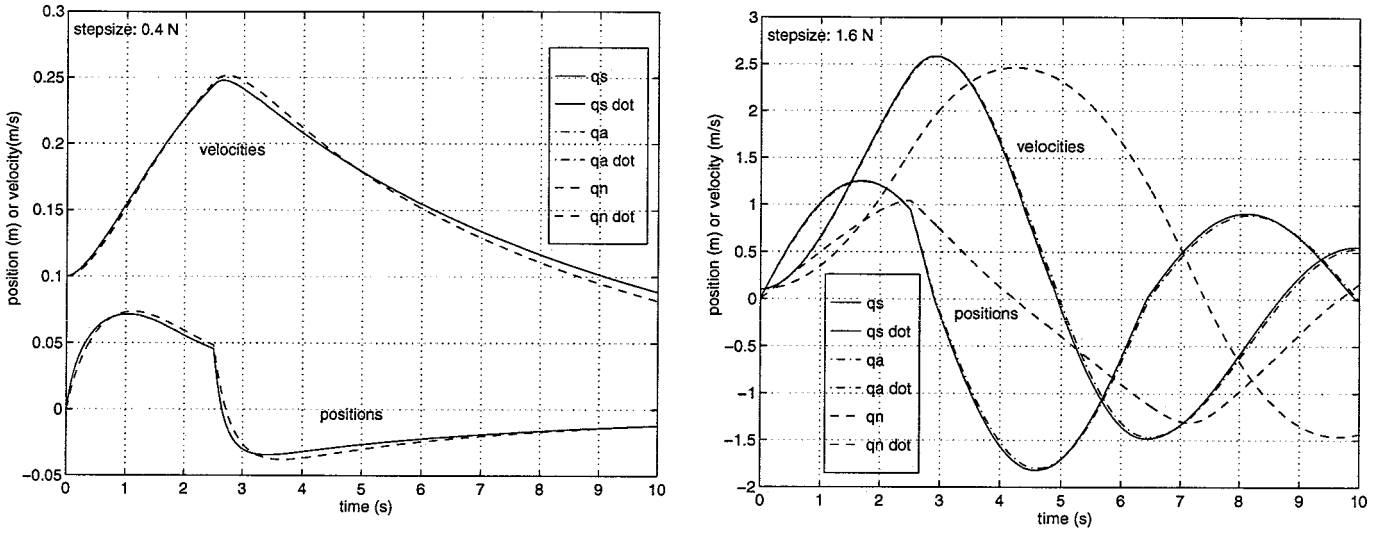

Figure 7: Responses of analytical (qa) and trained ANN (qn) vs real system (qs) on an input step size 0.4 [N] (left) and 1.6 [N] (right).

5 Two mass-spring-damper systems

5.1 Introduction

The second system used is depicted below in figure 8. It contains two masses, two springs and two dampers. The first mass is connected to the fixed world through a spring and a damper, the second is connected only to the first through another spring and damper. All springs and dampers are linear.

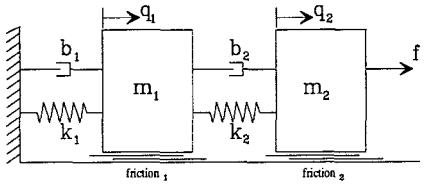

Figure 8: two mass-spring-damper system with friction

Both masses are different. The same goes for the springs and dampers. In this configuration, Coulomb friction is assumed, generated by the function $\frac{2}{\pi}\arctan(180x)$. This is a somewhat more `sign`-like function as in the previous example (see equation 17), but still without the infinite derivate at $x = 0$.

As this system has two masses, both can have a eigen frequency. The eigenvalues of this system are $-1.4930 \pm 1.0338i$ and $-0.1570 \pm 0.3915i$, which means eigen frequencies of 0.267 Hz and 0.025 Hz.

5.2 Identification with EKF on a mathematical model

For comparison, this system is also learned into the mathematical model of the system with $m_1, m_2, k_1, k_2, b_1, b_2$ and f_1 and f_2 as unknown parameters. The f_1 and f_2 parameters are weights for the friction.

This system trains fast: in 10 epoch the average error is below 10^{-8}, which is much faster than the ANN. The resulting estimates for the parameters are exact to four digits.

The reason is that only 8 parameters are unknown, the matrix Riccati equations consist of 94 equations. Compared to the 706 equations for the EKF (see below) this is not much, which means fewer calculations are required, thus calculations take less time.

5.3 Training ANN using EKF

The neural network for this system is chosen to be 5-4-2-2 (input - layer 1 - layer 2 - output), this is the minimum for the analytical version.

This results in a 36×36 matrix Riccati-equation, which means that in total (including the real system and the network) 706 differential equations have to be solved. (In the previous section, only 94 equations were necessary)

Training is done by an input function of three sines, with frequencies spread around both eigenfrequencies. Two inputfunctions, varying in frequencies, are used alternately, of which one also contains some noise.

After the 15th epoch, the weights $w_1(4,3)$ and $w_1(5,4)$ (refer to Appendix A for notation) are set to 40 . These weights have the largest values (both around 1.6), used for imitating the friction.

This is done by hand to overcome the saturation-problems of the Kalman algorithm, see Section 5.5 for details.

After this change, a significant drop in the error of the estimates occurs, as is shown in figure 9.

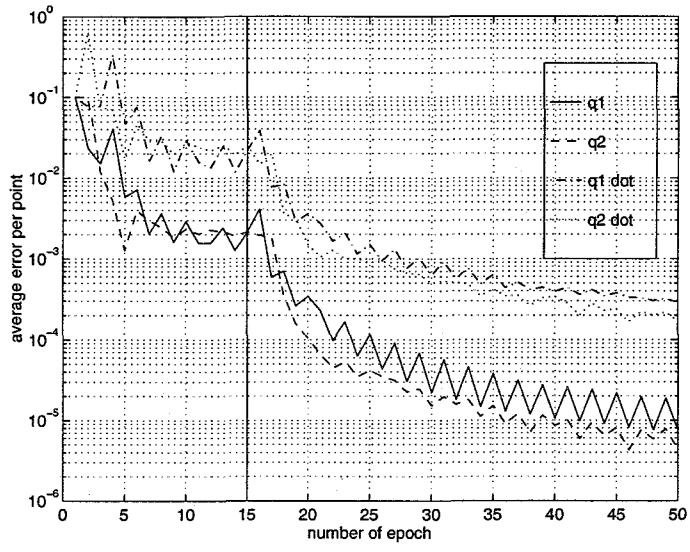

Figure 9: average error per point per epoch for q_1, q_2 and their derivatives

In this figure, it is shown that the estimates for the position are better than those of the velocity. The error in the estimates does not vary much between both masses.

5.4 Comparing the results

After training the ANN using the adaptation as mentioned in Section 5.3 , the resulting network is compared to the real system and the mathematical EKF from Section 5.2.

A block wave of size 1.4 [N] and period 5 [s] is applied to the system. The responses are drawn in figure 10.

Around the training point, the network performs reasonably well. Note that the mathematical model is not plotted, because it doesn't differ significant from the real system. When using larger (> 2.4 [N]) or smaller stepsizes (< 1.0 [N]), the results have no accuracy .

5.5 The saturation of nodes

The Kalman algorithm as used in this report reveals a major problem:

to imitate the friction, at least one of the neural nodes has to become saturated. Thus the input must be weighted very heavy (at least about 15) to create the

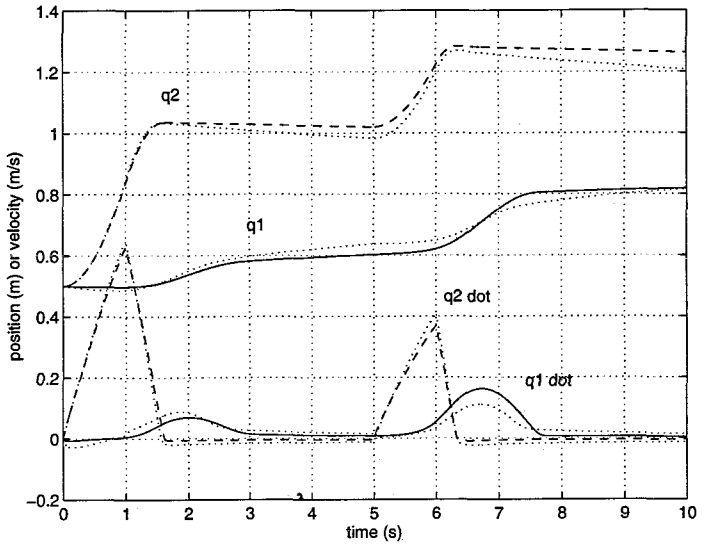

Figure 10: Comparison between the real system and the neural network (dotted) on a block-input of size 1.4 [N].

`sign`-like function.

The Kalman-algorithm does not saturate to this amount. The highest viewed value is about 3.5 in more than 80 epoch... This is by far too low.

There may be several ways to work around this problem:

- Adjust the weights in advance. A —correct— large weight will be maintained and the error on the output decreases significant, as can be seen in figure 9.
- Use much more nodes, thus enabling the filter to adapt more weights. As this results in very time consuming calculations, this has not been tested.
- Try other non-linear functions for the nodes, such as *sigmoid* functions. See Singhal and Wu [4].
- Split the first network layer horizontally and insert sign-like network-equations. This is drawn in Figure 11.

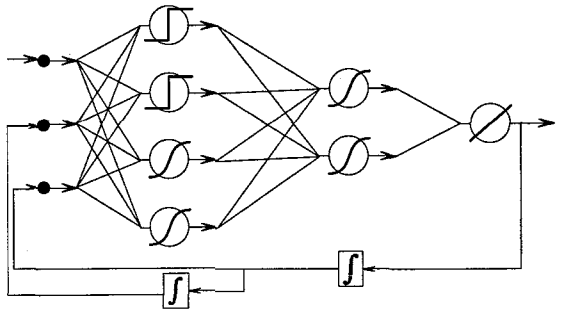

Figure 11: A proposal for a better black-box identification network

6 Conclusions and Recommendations

6.1 Conclusions

It is possible to use an Artificial Neural Network (ANN) as a model for the system equations of a mechanical system. By adding the weights to the state equations, the augmented state was created. Using an Extended Kalman Filter approach, it was possible to estimate the weight parameters of dynamic systems. This was demonstrated on two non-linear mechanical systems: a single and a double mass-spring-damper system, both with Coulomb friction. The results show that the approach does work, but three major disadvantages were found:

- The EKF algoritm does not push nodes severely into saturation. This was necessary, as the number of nodes was kept small.

  Correction by hand resolved the problem.

- The number of calculatons is huge, even for small systems. This is a disadvantage of the (Global) EKF-algoritm. (See Recommendations).

- An ANN is not capable of extrapolation. The resulting model is only usable within the range of inputs with which it was trained.

6.2 Recommendations

A few recommendations for further investigations can be made:

- To overcome the first disadvantage as mentioned in the Conclusions, the number of nodes in the ANN must be enlarged. This will increase the number of calculations severely. To work around this problem, a few solutions are proposed:
  - implement the *Node Decoupled Extended Kalman Filter* algoritm as proposed by Puskorius [5] and Murtuza [2] (see Introduction) instead of the EKF. This wil reduce the calculation time to about 15%. The NDEKF neglects the interdepence of the nodes which amounts to about 80% of the Q-matrix in the Ricatti-equations.
  - try Obradovic's varying topology-algorithm. (see Introduction and literature [7]). This can help estimating the right size of the network which must be used, thus eliminating the calculations neccesary for the adaption of unused nodes.

- It is also possible to use different kinds of node-functions simultaneously, including of real `sign`-like functions in the network might result in better estimation of the Coulomb friction.

- This dynamic EKF-algorithm is only tested on simulated dynamic systems. The results on real systems, e.g. the rotating masses-experiment in the lab of WFW, may reveal other disadvantages of this approach.

References

[1] Syed Murtuza: *A Concise Presentation of Supervised Learning Algorithms for feedforward neural networks* in: *Proceedings of the 1994 IFAC Advances in Control Education*, Tokyo, Japan 1994
pages 91-94.

[2] Syed Murtuza en Steven F. Chorian: *Node Decoupled Extended Kalman Filter-based learning algorithm for neural networks* in: *Proceedings of the 1994 IEEE International Symposium on Intelligent Control (ISIC)* Columbus Ohio, USA 1994.
pages 364-360, Including MATLAB source.

[3] J.J. Kok: *Werktuigkundige Regeltechniek II*
reader Eindhoven University of Technology, Eindhoven, The Netherlands, 1990.

[4] Sharad Singhal en L. Wu: *Training feed-forward networks with the Extended Kalman algorithm* in: *IEEE ICASSP (International Conference on Artificial Speech and Signal Processing 1989* Glasgow Scotland 1989.
Volume 2 pages 1187 - 1190.

[5] G.V. Puskorius en L.A. Feldkamp: *Decoupled Extended Kalman Filter Training of Feedforward Layered Networks* in: *Proceedings of IJCNN'91 Seattle*, Seattle, USA, 1991,
pages I771-I777.

[6] Edwin Verschuren: *Identification and Control of an Inverted Pendulum using Neural Networks*
M.Sc.thesis Eindhoven University of Technology, section Fundamental Engineering, Eindhoven, The Netherlands, december 1995.

[7] D. Obradovic: *Recursive Learning in Recurrent Neural Networks with Varying Architecture* in: *ICANN'94 Proceedings of the International Conference on Artificial Neural Networks* Springer Verlag Berlin, Germany, 1994,
volume 1, pages 447-450.

A Notation

In this report, notation will conform to Syed Murtuza's [1] terrific introductory article on training static networks.

- The numbering of layers starts with the input layer as layer #0.
- The definition of a node and its various signals is shown in figure 12.
- A neuron has a `tanh`-function as neural function.

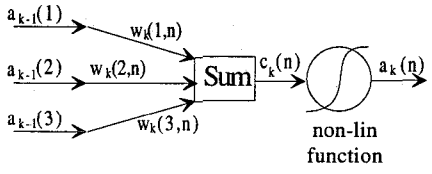

Figure 12: Definition of a neural node.

An exception to this is the notation of derivates: A derivate to x is denoted by $f_{,x}(\ldots)$ and a dervate of time is denoted by a dot on top: $\dot{q}$.

The notation will be enhanced by the following conventions:

- All matrices are capitalized. (W)
- All vectors are underlined. $(\underline{w}_1)$
- A hat will denote an estimated variable. $(\hat{x})$.
- A weight matrix will have size $s_0 \times s_1$, where s_0 is the number of nodes in layer zero and s_1 of the first layer.
- All equations end in a semi-colon. (;)

B About the source code

A floppy disk should accompany this report. This floppy disk, formatted using MS-DOS contains all relevant source code.

The floppy contains two subdirectories, each containing a full set of so-called M-files to reproduce the results obtained in this essay.

The programmes are written for use with MATLAB 4.2 and work both under UNIX and Windows 3.x .