CODEN: LUTFD2/(TFRT-5442)/1–45/(1991)

Neural Networks Applied to Optimal Flight Trajectories

Tomas McKelvey

Department of Automatic Control
Lund Institute of Technology
September 1991

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
September 1991

Document Number
CODEN: LUTFD2/(TFRT-5442)/1–45/(1991)

Author(s)
Tomas McKelvey

Supervisor
K J Åström, B Järmark, S Wickström

Sponsoring organisation

Title and subtitle
Neural Networks Applied to Optimal Flight Trajectories

Abstract

Two flight control problems, a simple aiming problem and an optimal climb problem, are studied using neural networks to develop nonlinear control laws. The networks are trained with the back-propagation learning algorithm using examples of different states in the flight envelope. In the aiming problem an analytical solution exists, which was used to produce the learning material. A differential dynamic programming technique is used to produce optimal climb trajectories used as learning material in the optimal climb problem. The produced neural networks have been simulated with aircraft models and the results are analyzed. They show promise for application to nonlinear control problems.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
45

Recipient's notes

Security classification

Neural Networks Applied to Optimal Flight Trajectories

Tomas McKelvey

Master Thesis

Department of Automatic Control
Lund Institute of Technology
Sweden

Abstract

Two flight control problems, a simple aiming problem and an optimal climb problem, are studied using neural networks to develop nonlinear control laws. The networks are trained with the back-propagation learning algorithm using examples of different states in the flight envelope. In the aiming problem an analytical solution exists which was used to produce the learning material. A differential dynamic programming technique is used to produce optimal climb trajectories used as learning material in the optimal climb problem. The produced neural networks have been simulated with aircraft models and the results are analyzed. They show promise for application to nonlinear control problems.

Acknowledgments

This master thesis work was done for the department of Automatic Control, Lund Institute of Technology, under the supervision of Karl-Johan Åström. The work was completed at SAAB-SCANIA AB, Saab Aircraft Division, Linköping, at the department of Application and System Analysis, under the supervision of Bernt Järmark and Sören Wickström.

Table of contents

1. Introduction

The human brain with its vast resources of remembering and solving problems has inspired many researchers to model the brain's computational capabilities. Some of this research has led to very simplified functional models, called neural networks, which can be described mathematically. In the last few years research on neural networks has expanded significantly and a number of technical applications in different fields have been explored. Current research is directed to enhance the models of the brain and towards the use of current models in different applications. Researchers at engineering faculties tend to work on applications and towards developing an understanding of the model as a computational tool rather than as a model of the brain.

One of the appealing features of a neural network is its capability to learn by example. By providing the network with both the questions and the answers the network builds an internal model of the problem during the learning phase. If correct learning material was used, the network will probably perform well for unknown questions during the recall phase.

This is of course a direct analogy of the use of system identification methods. The only difference is the type of models that are fitted. Most system identification work has been devoted to linear models. Whereas an interesting feature of neural networks is that they are nonlinear.

In this thesis neural networks are used to design nonlinear control laws solving nonlinear control problems. The neural networks are used as feedback controllers. Two nonlinear flight problems have been investigated: A neural network solving a simple aiming problem and a neural network solving an optimal energy climb problem. For the optimal climb problem a numerical method is used to produce a number of open loop solutions used as learning material for the neural network. A wide range of different types of networks exist in the literature. In this thesis only the multi layer perceptron network with back-propagation learning is used.

The methodology used shows promise for application to highly nonlinear control problems where analytical design techniques do not exist today.

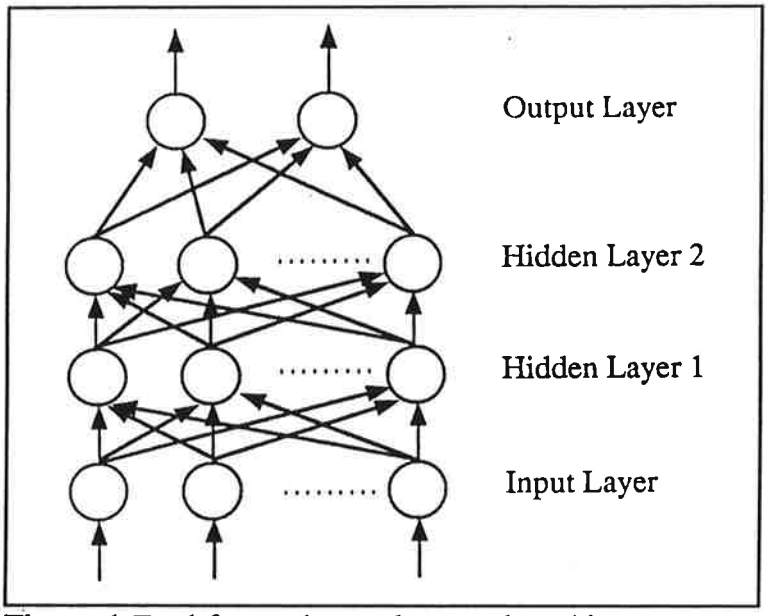

Figure 1 Feed-forward neural network architecture.

2. Neural Networks

This section is compiled from ref. [1] and [2] and describes the neural network type used in this thesis. A commercial neural network program package, NeuralWorks Professional II/PLUS [1] have been used to develop the neural networks.

A neural network is a system with inputs and outputs and is composed of many simple and similar processing elements (PE) also called neurons, see Fig. 1. Each PE is a simple microprocessing unit which receives and combines signals from many other processing elements. The received signals are processed and result in an output signal. The PE's output can then be connected to other PEs' inputs, etc., creating a network of PEs. The PEs are usually organized in layers starting with the input layer, sometimes called the input buffer, one or more hidden layers and an output layer or buffer. Each processing element's output in a layer is fully connected to all processing elements in the succeeding layer thereby creating a full interconnection between the two layers. When signals are applied to the input layer the processed signals propagates through each layer to the output layer where the result is available on the output neurons. The input layer performs no processing in the inputs and works only as a transparent buffer to the succeeding layer.

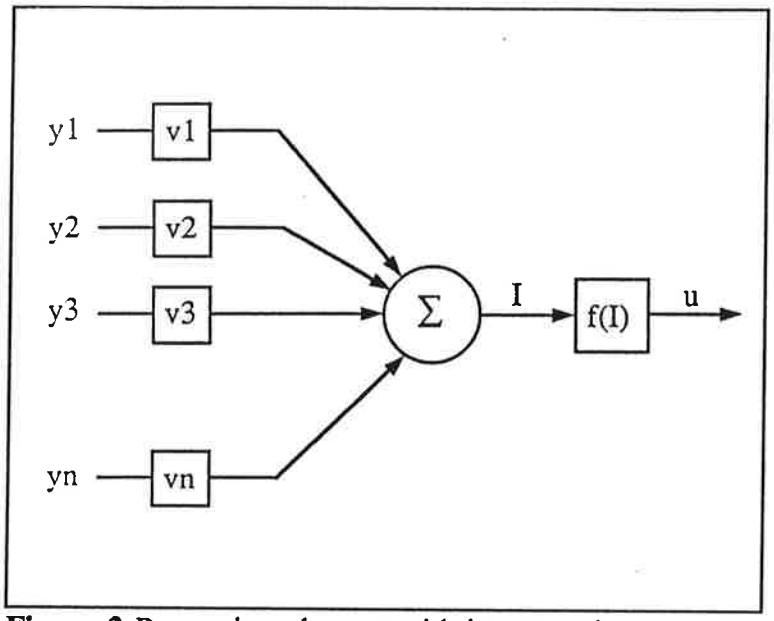

Figure 2 Processing element with inputs and output.

2.1 Processing Element

The processing element used in this thesis is shown in Fig. 2. It has an input vector Y which contains n scalar components, one single scalar output u and a weight vector V which also contains n components. The weights are variable coefficients. The output u equals the sum of inputs multiplied by the weights and then passed through a nonlinear transfer function. In early research on neural networks a step function was often used. In order to use the back-propagation learning algorithm it is necessary to use a differentiable transfer function. The nonlinear function used here is the hyperbolic tangent function Eq. (1) and Fig. 3.

$$f(z)=\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}=\tanh(z) \tag{1}$$

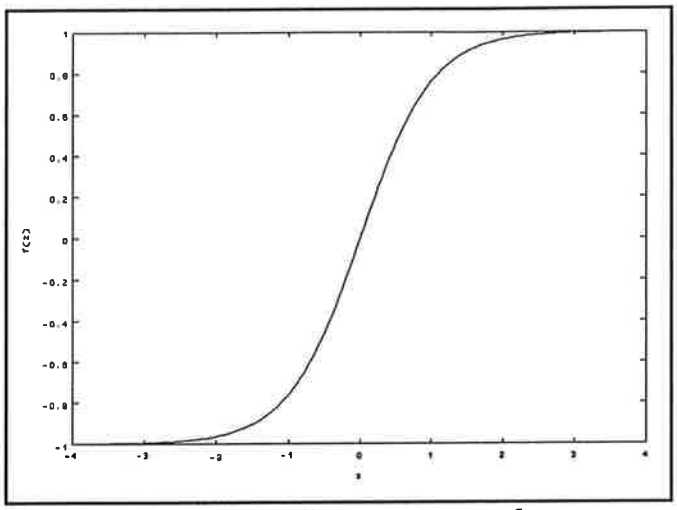

Figure 3 The hyperbolic tangent transfer function.

The output of the neuron can thereby be described by

$$u=f(V^T Y) \tag{2}$$

or equally with a different notation

$$u=f(\sum_{k=1}^{n} v_k * y_k) \tag{3}$$

where V is the n-dimensional weight vector. The weights in V control the function of the neuron. All neurons also have one input connected to a constant value called bias. The bias input fills a similar function as ground connection in an electrical circuit.

2.2 The Layer

A layer is created by connecting many processing elements to the same input vector. Each layer performs a transformation from the preceding layer's outputs $Y^{[s-1]}$ to the current layer's outputs $Y^{[s]}$, the superscript indicating the layer number. If the previous layer has n neurons and the current layer has m neurons then a $m*n$-dimensional matrix W holds all the connection weights between the two layers. Row x in W holds the weights for neuron x in the layer.

If we define

$$f(Y) = \begin{bmatrix} f(y_1) \\ f(y_2) \\ \dots \\ f(y_n) \end{bmatrix} \tag{4}$$

we can describe the transformation between two layers outputs as

$$Y^{[s]} = f(W^{[s]} Y^{[s-1]}) \tag{5}$$

2.3 The Network

The network can be viewed as a number of cascaded layers, also known as a layered feed-forward neural network. Outputs of one layer are connected to the inputs of the next layer. The function from input X to output Y can thus be described as

$$Y = f(W^{[m]} f(W^{[m-1]} \dots f(W^{[1]} X) \dots)) \tag{6}$$

where m is the number of layers, not counting the input buffer.
This architecture makes it possible to perform nonlinear transformations between several continuous inputs and one or more outputs.

It has been proven that a network consisting of only two layers (one hidden and one output layer) of processing elements can implement any continuous function given enough processing elements in the first (hidden) layer. The idea is that each processing element in the hidden layer makes a linear approximation of a small piece of the function. In reality, when size and learning characteristics also are very important, networks with more than one hidden layer are often used. The ability to generalize, to respond correctly to inputs not used in training, can also be different using more hidden layers.

2.4 Network Operation

Neural networks have two modes of operation: learning and recall.
Learning is the process of adapting or modifying the connection weights in response to presented input values and desired output values. This is called supervised learning and requires material with correct answers to the problem to be solved by the network. The selection of learning material is fairly critical to the networks final performance and will be further discussed in later sections.

During recall the network weights are fixed and for each input vector X the transformation according to Eq. (6) is performed, yielding the output vector Y.

2.5 Back-Propagation Learning

Learning is the process of adapting the network's weights in order to reduce the output error. If the output is in error, how do you determine which processing element or weight to adjust? Back-propagation solves this by assuming that all processing elements and connections are somewhat to blame for an erroneous response. Responsibility for the error is affixed by propagating the output error backward through the connections to the previous layer. This process is repeated until the input layer is reached. The name 'Back-Propagation' derives from this method of distributing the blame for errors.

The goal of learning is to minimize the network's output error over the whole training set by changing the weights of the network.

Let us use the following notation:

$y_j^{[s]}$ current output state of jth neuron in layer [s]

$w_{ji}^{[s]}$ weight on connection between ith neuron in layer [s-1] and jth neuron in layer [s]

$I_j^{[s]}$ weighted summation of inputs to jth neuron in layer [s]

This notation gives us the following description of a neuron:

$$I_j^{[s]}=\sum_i w_{ji}^{[s]} y_i^{[s-1]} \tag{7}$$

$$y_j^{[s]}=f(I_j^{[s]}) \tag{8}$$

Let X be the input vector presented to the input layer and let O be the output vector produced by the network. If D is the desired output vector given by the teacher, the error function is described by

$$E=\frac{1}{2}\sum_k (d_k-o_k)^2 \tag{9}$$

For each neuron in the network let us define the local error e as:

$$e_j^{[s]}=-\frac{\partial E}{\partial I_j^{[s]}} \tag{10}$$

The local error is passed back through the network in the following way:

$$
\begin{aligned}
e_j^{[s]} &= -\frac{\partial E}{\partial I_j^{[s]}} \\
&= \sum_k \left(\frac{\partial I_k^{[s+1]}}{\partial I_j^{[s]}} * \frac{\partial E}{\partial I_k^{[s+1]}}\right) \\
&= f'(I_j^{[s]}) * \sum_k (w_{kj} * e_k^{[s+1]})
\end{aligned}
\tag{11}
$$

Note that Eq. (11) is not valid for the output layer.

Equation (9) gives us the possibility to describe the local error e for the output layer.

$$
\begin{aligned}
e_j^{[o]} &= -\frac{\partial E}{\partial I_j^{[o]}} \\
&= -\frac{\partial E}{\partial o_j} * \frac{\partial o_j}{\partial I_j^{[s]}} \\
&= (d_j - o_j) * f'(I_j^{[o]})
\end{aligned}
\tag{12}
$$

The weight update is based on the gradient descent rule as follows:

$$
\Delta w_{ji}^{[s]} = -\mu * \frac{\partial E}{\partial w_{ji}^{[s]}} \tag{13}
$$

where μ is the learning coefficient and Δw_{ji} is the weight change. This means that all the weights are changed according to the size and direction of the negative gradient on the error surface.

The weights are thus changed as:

$$
w_{ji}^{[s]}(t+1) = w_{ji}^{[s]}(t) + \Delta w_{ji}^{[s]} \tag{14}
$$

with t as the number of learning iterations.

Developing the partial derivative of Eq. (13) using the local error defined in Eq. (10) gives:

$$
\begin{aligned}
\frac{\partial E}{\partial w_{ji}^{[s]}} &= \frac{\partial E}{\partial I_j^{[s]}} * \frac{\partial I_j^{[s]}}{\partial w_{ji}^{[s]}} \\
&= -e_j^{[s]} * y_i^{[s-1]}
\end{aligned}
\tag{15}
$$

Combining Eq. (13) and Eq. (15) gives

$$\Delta w_{ji}^{[s]} = \mu * e_j^{[s]} * y_i^{[s-1]} \tag{16}$$

The derivative of the hyperbolic tangent transfer function can be expressed in terms of itself:

$$f'(z) = (1 + f(z)) * (1 - f(z)) \tag{17}$$

In the learning phase this makes it computational efficient to calculate the derivative.

2.6 Global Error G_e

The error E describes the current output error for a given pair of input and output vectors (X,Y). The ultimate goal of the learning is to minimize the RMS value of all errors E in the learning set, here called the global error. If n is the number of input/output pairs in the training set, the global error is:

$$G_e = \sqrt{\frac{1}{n}\sum_{k=1}^{n} E_k^2} \tag{18}$$

This is a good variable to use to monitor the network's progress during learning.

2.7 The Choice of Learning Coefficient

The value of the learning coefficient μ in Eq. (16) affects the speed of convergence and the stability of the weights during learning. A small value gives a slow convergence while a too high value might give oscillations in the weights. Different learning coefficients are used for different layers. The first hidden layer has the largest value and the following layers have decreasing values. Values used in this thesis are $\mu=0.3$ for the first hidden layer $\mu=0.25$ for next and $\mu=0.15$ for the output layer. This differentiation helps the network to build a good and accurate model of the problem during the learning phase.

2.8 Learning Schedule

To improve learning speed a high learning coefficient is desired. An excessively large learning coefficient might give instability and prevent the learning to reach the global minimum since too large steps are taken in the weight space. The solution to this problem is to use a time dependent learning coefficient, starting with a large value and gradually decreasing the value as learning progresses.

2.9 The Problem of Existing Local Minima

The global error function over the whole training set usually has several local minima in the weight space. During learning the networks often stabilizes in a local minima and further learning is inhibited. As a neural network designer it is of great importance to be aware of this problem.

Three solutions to the problem have successfully been used in this work:

* Adding random numbers to all the weights at discrete times.
* Reset the learning coefficient to the initial large value.
* Adding noise to all PEs during learning.

Adding random values to the weights is a simple method to apply. When the global error stabilize and it seems like a local minimum is reached, stop the learning and add random values to all the weights in the network. This moves the network to a new position in the weight space and hopefully when learning recommences, the network will find a better path to the real minimum.

Resetting the learning coefficient to the initial large value will make the network take larger steps in the weight space and thus maybe it will step away from the local minimum.

Adding noise to the sum of each processing element is a more sophisticated method. This method is also called simulated annealing. Starting with large values, the values are decreased towards zero during learning. The result is that the global error function continuously changes. If a local minima exists at one point, the changing shape of the curve might make the minima disappear at the next moment.

These methods can be used separately or in a combination. The two first methods require supervision during learning but have been proven effective in this work.

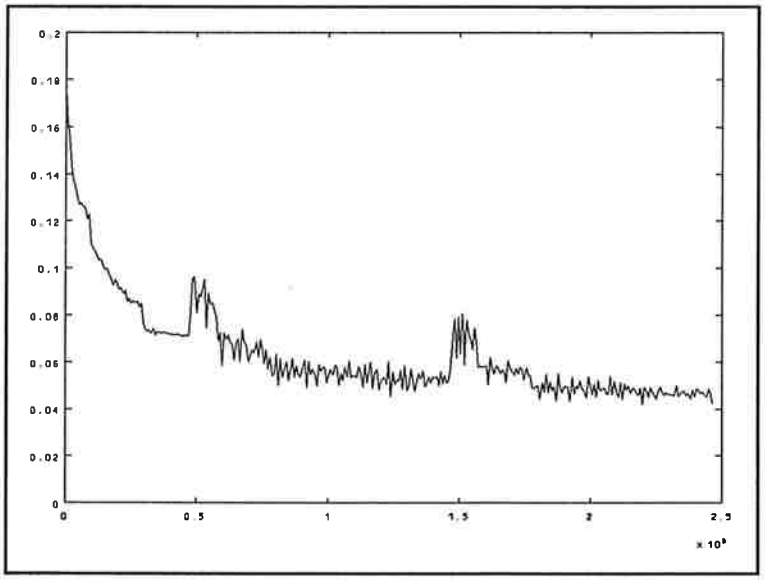

Figure 4 Typical global error curve.

In Fig. 4 the global error vs. number of iterations is shown for a typical network. At 50.000 and 150.000 iterations random numbers are added to the weights and the learning coefficients are reset to their initial values. The actions performed increased the convergence speed.

2.10 Pre-Processing

In order to be successful in the use of neural networks, the literature states that preprocessing of the data is of great importance. Preprocessing can be done in several ways including nonlinear transformations. The goal is to change the input and output data to fit the neural network's input and output characteristics. A simple example is that the output of a neuron is limited by the transfer functions output. With the use of the hyperbolic tangent transfer function the output is limited to the interval [-1,1]. In order to learn effectively the desired outputs should also lie within this range.

This leads us to convert the real world values to network values and back again. The easiest transformation is the linear which is here used.

The first step is to scan the learning material and establish the minimum and maximum value for each input and output. This gives us a maximum value M and a minimum m for each input and output. In order to define the transformation the range of the network is also needed. Let the ranges be denoted with R and r for maximum and minimum values respectively.

The input transformation is then defined as:

$$i=\frac{(R-r)*f+(M*r-m*R)}{M-m} \tag{19}$$

where f is the current real-world value to transform. The same transformation is made for the desired output.

On output, the mapping from network output to real world is

$$g=\frac{(M-m)*o+(R*m-r*M)}{R-r} \tag{20}$$

where g is the real-world output and o is the network output. The choice of network range is closely linked to the number of inputs, type of transfer function in use and the initial weight values in the network. The basic idea is to choose a range that produces summations which will not initially saturate the transfer function in the neurons.
The network ranges used are [-1,1] for inputs and [-0.8,0.8] for outputs. The output range is chosen in order to give the derivative term f' in the learning rule a non-zero value. A neuron with a saturated transfer function stops learning since the derivative is zero.

2.11 Network Initialization

Before learning starts all the weights are initialized in a random fashion with values in a fixed interval. The interval to be used is highly dependent on the number of neurons in each layer. If too large values are used in combination with a high number of neurons the risk increases that the transfer functions saturates and learning is inhibited. A reasonable interval to use is [-0.5,0.5].

2.12 Learning Time

The number of iterations needed for learning in this thesis are in the range from 50.000 to 200.000. Using an IBM-PC 386 computer with a math co-processor this equals 0.5 to 2 hours depending on the network complexity.

3. A Simple Aiming Problem

To get a first indication of a neural networks performance as a feedback nonlinear control law, a simple aiming problem with an analytical control law was studied. The problem of driving the aiming error to zero can be viewed as a regulator problem.

Two aircraft, a target and an aggressor, form an air combat situation. The target is assumed to fly straight along the positive x-axis, and the problem is to find the controls for the aggressor which will close the aiming triangle and keep it closed. The problem is only studied in the horizontal plane, neglecting the bullet's drop due to gravity and only using mass point dynamics.

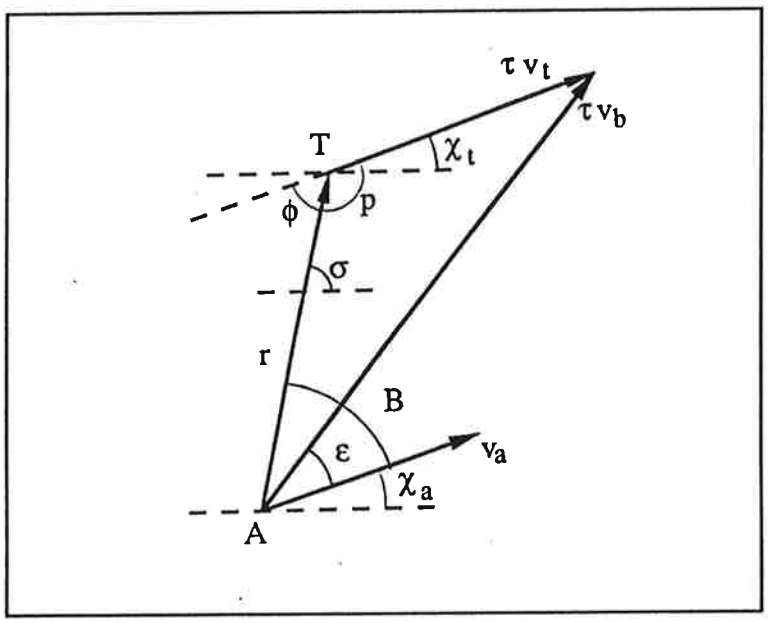

Figure 5 The aiming triangle.

3.1 Aiming Condition

The sides of the aiming triangle consist of the line of sight from the aggressor to the target, the target's flight path and the bullets path. The geometry shown in Fig. 5 gives the following condition:

$$\bar{r}+\bar{v}_t\tau=\bar{v}_b\tau \tag{21}$$

Where r is the distance between the aircraft, v_t is the target velocity and v_b is the mean velocity of the bullet. The time τ is the bullet's time in the air.
This vector equation can be rewritten using the sine theorem as:

$$\frac{v_b\tau}{\sin p}=\frac{v_t\tau}{\sin(\sigma-\chi_a-\varepsilon)} \tag{22}$$

The aiming error ε is then:

$$\varepsilon = \sigma - \chi_a - \arcsin(k \sin p) \tag{23}$$

with

$$k = \frac{v_t}{v_b} \tag{24}$$

3.2 Model of Turning Aircraft

By banking, the wing's lifting force is divided into two forces: the vertical force equating the gravity and a horizontal force turning the aircraft. Let the horizontal force be described with its associated load factor n_c. The geometry of a circle gives:

$$\rho \dot{\chi} = v \tag{25}$$

where ρ is the radius of the turn.
Using the horizontal force gives:

$$n_c m g_0 = m v^2 / \rho = m v \dot{\chi} \tag{26}$$

where g_0 is the acceleration due to gravity and m is the aircraft's mass.
Thus the turn rate χ is then:

$$\dot{\chi} = \frac{g_0}{v} n_c \tag{27}$$

3.3 Dynamics

The dynamics of the aggressor in earth fixed coordinates are:

$$\dot{x}_a = v_a \cos \chi_a \tag{28}$$

$$\dot{y}_a = v_a \sin \chi_a \tag{29}$$

$$\dot{\chi}_a = \frac{g_0}{v_a} n_c; \quad |n_c| < n_{c_{max}} \tag{30}$$

where v_a is the aggressor's constant speed and χ_a the heading.
The target's dynamics are:

$$\dot{x}_t = v_t \tag{31}$$

$$\dot{y}_t = 0; \quad y_t(t_0) = 0 \tag{32}$$

$$\dot{\chi}_t = 0; \quad \chi_t(t_0) = 0 \tag{33}$$

The turning load factor n_c of the aggressor is used as the independent control

variable.

3.4 Control Law

The objective of the control is to make the aiming error go to zero. In ref. [3] a control law that takes the aiming error and its derivatives into account is described. A similar approach is here used.

Assume that

$$\varepsilon(t_0)\neq 0 \tag{34}$$

and it is desirable that

$$\varepsilon(t_0+\Delta t)=0 \tag{35}$$

where Δt is the time to drive ε to zero.
A first order approximation gives

$$\varepsilon(t_0+\Delta t)=\varepsilon(t_0)+\Delta t\dot{\varepsilon}(t_0) \tag{36}$$

Set the left hand side of Eq. (36) to zero and use Eq. (23) to evaluate the derivative of ε. This gives us an explicit control law with the parameter Δt to tune with.

$$\begin{aligned} 0&=\varepsilon(t_0)+\Delta t(\dot{\sigma}-\frac{g_0}{v}n_c-\frac{k\cos p}{\sqrt{1-(k\sin p)^2}}\dot{p}) \\ \Rightarrow\ n_c&=\frac{v}{g_0}(\frac{\varepsilon(t_0)}{\Delta t}+\dot{\sigma}-\frac{k\cos p}{\sqrt{1-(k\sin p)^2}}\dot{p}) \end{aligned} \tag{37}$$

A small value of Δt gives a high gain in the control law.

Simulations with an Euler integration method showed that Δt=0.2 s. and an integration step h=0.1 s. gave reasonable results.

3.5 Test 1

The state input to the network was chosen to be $\Delta x=x_t-x_a$, $\Delta y=y_t-y_a$ and the aggressors bearing to the target, $B=\sigma-\chi_a$. The output was the turning load factor n_c.

By using the control law for different initial positions, a set of states with the correct control were produced and used as the learning material. The initial positions for the aggressor were:

$$\begin{aligned} &r=n*666; && n=1,2,3 \\ &\sigma=n*30^\circ\pm 15^\circ; && n=-5,-4,...,4 \\ &B=n*30^\circ; && n=-1,0,1 \end{aligned} \tag{38}$$

with the target in the origin.

A network with three inputs, five neurons in the first hidden layer, five in the second hidden layer and one output (3-5-5-1) showed a convergent behaviour when learning.

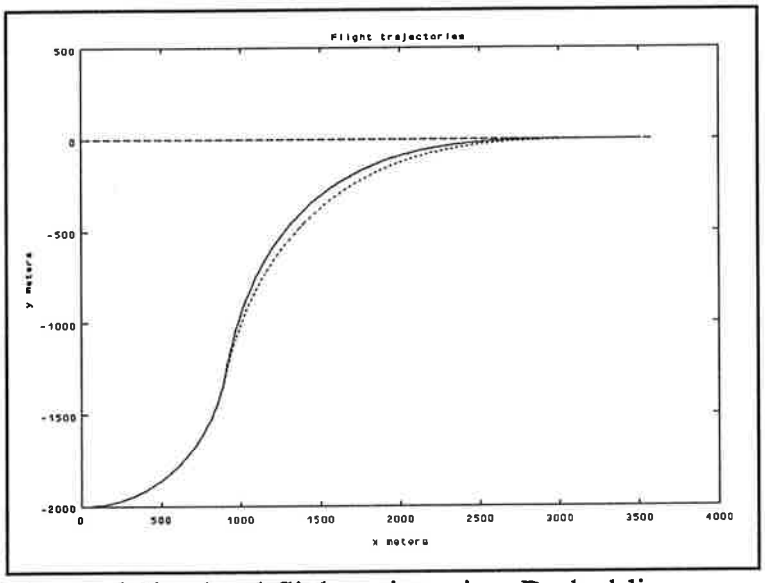

Figure 6 Simulated flight trajectories. Dashed line: target. Dotted line: aggressor with control law. Solid line: aggressor with neural network control.

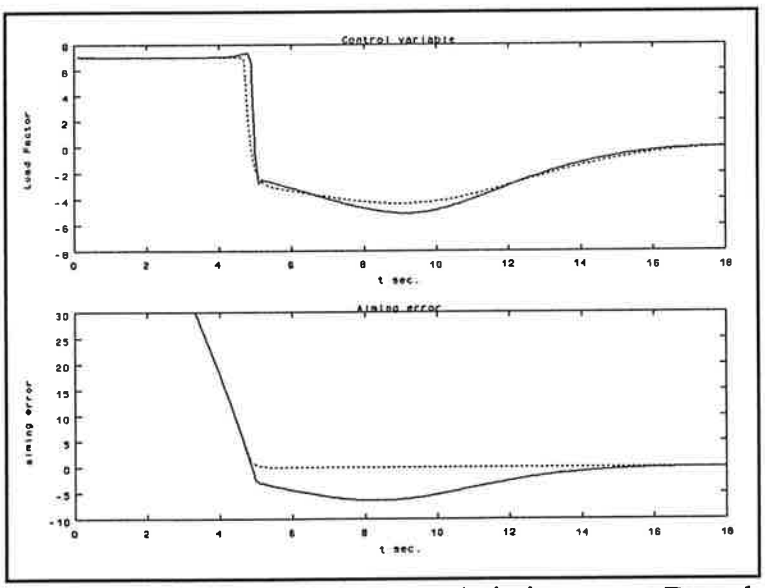

Figure 7 Resulting load factor and aiming error. Dotted line: control law. Solid line: neural network.

Figure 6 and 7 show simulation results with the performance of both the control law and the neural network. The neural network performs well in the initial phase but takes a long time to drive the aiming-error to zero. A quantitative measure of

performance t_i is defined as the time when the aiming error is less than $\pm 0.5°$. This simulation gave $t_i = 5.0$ s. for the control law and $t_i = 14.7$ s. for the network.

3.6 Test 2

The learning material used in Test 1 almost only consisted of the maximum control values, i.e. full left and right turns. This is due to the initial positions used. In order to also include learning examples with other control values, i.e. control values used after the initial maximum turn phase, initial positions with headings giving almost a zero aiming error, were also included in the material.

Using the same network architecture and the extended learning set, the error in the learning phase did not converge as well as in Test 1.

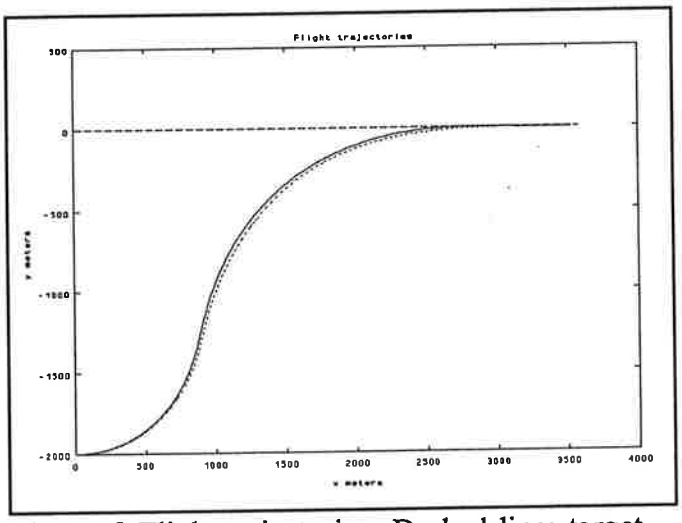

Figure 8 Flight trajectories. Dashed line: target. Dotted line: control law aggressor. Solid line: neural network aggressor.

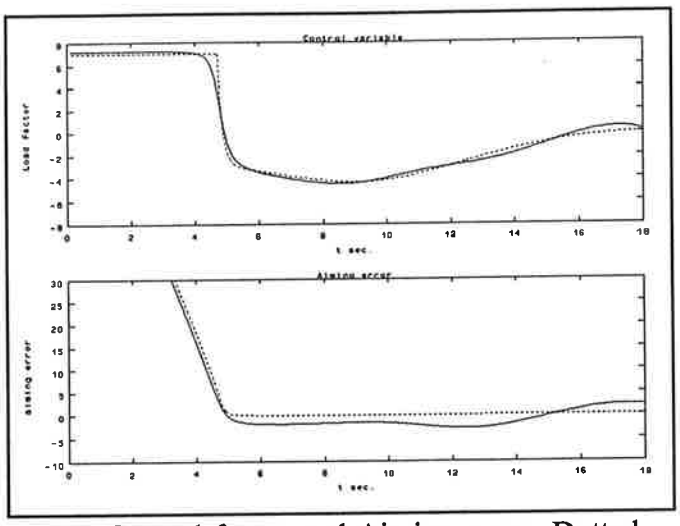

Figure 9 Load factor and Aiming error. Dotted line: control law. Solid line: neural network.

Results from simulation are shown in Fig. 8 and 9. The aiming error is not driven to zero even if both the control and the trajectory of the network are close to the control law's. Clearly here the network's performance was degraded by using an extended learning set with more examples.

3.7 Test 3

One way to reduce the aiming error is to include it among the neural networks inputs. This is one technique to explicitly amplify the importance of keeping the error small. A new network is designed with the same architecture plus one more input (4-5-5-1).

The same initial states as in Test 2 were used in the learning set.

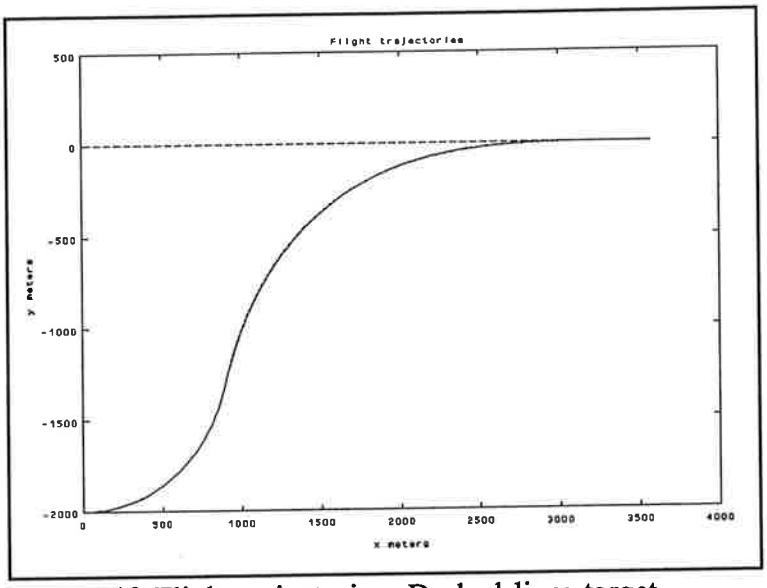

Figure 10 Flight trajectories. Dashed line: target. Dotted line: control law aggressor. Solid line: neural network aggressor.

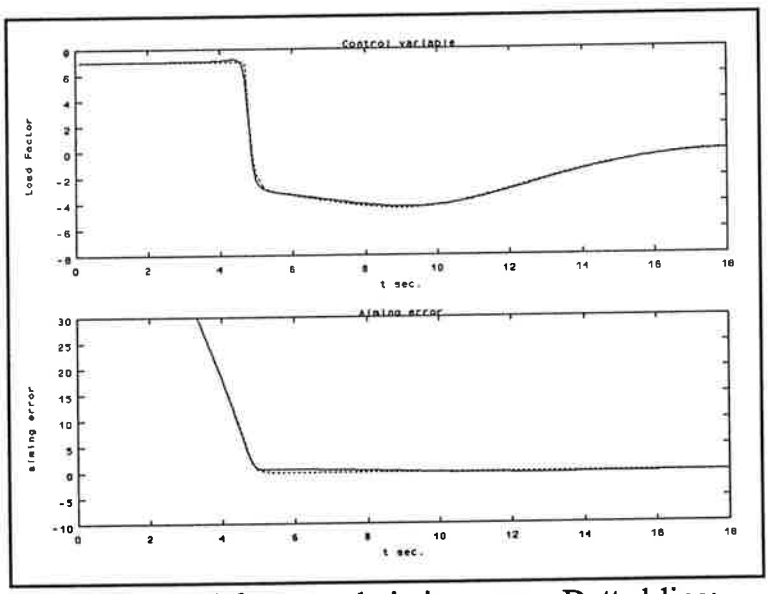

Figure 11 Load factor and aiming error. Dotted line: control law. Solid line: neural network.

The results of a simulation with the network are shown in Fig. 10 and 11. The performance is improved and the performance index is now reduced to t_i=6.9 s. When investigating the network's weights, it is clear that the network, during learning, has made the weights connecting the aiming error and the output large. The controller can thus be seen as a proportional feedback controller with added nonlinear components in the feedback loop.

3.8 Test 4

The earth fixed coordinate system used above is one way of describing the problem. A body-fixed coordinate system fixed to the aggressor is a more natural way of describing the target's position with bearing B, distance r and aspect angle ϕ. Using the notation above gives:

$$B=\sigma-\chi_a$$
$$r=\sqrt{(x_t-x_a)^2+(y_t-y_a)^2} \tag{39}$$
$$\phi=\sigma-\chi_t=\sigma$$

A new set of learning material is constructed using the same initial states as above now with B,r and ϕ as the state variables used as inputs to the network. A network with a (3-5-5-1) architecture was used. During learning the error converged to a low value.

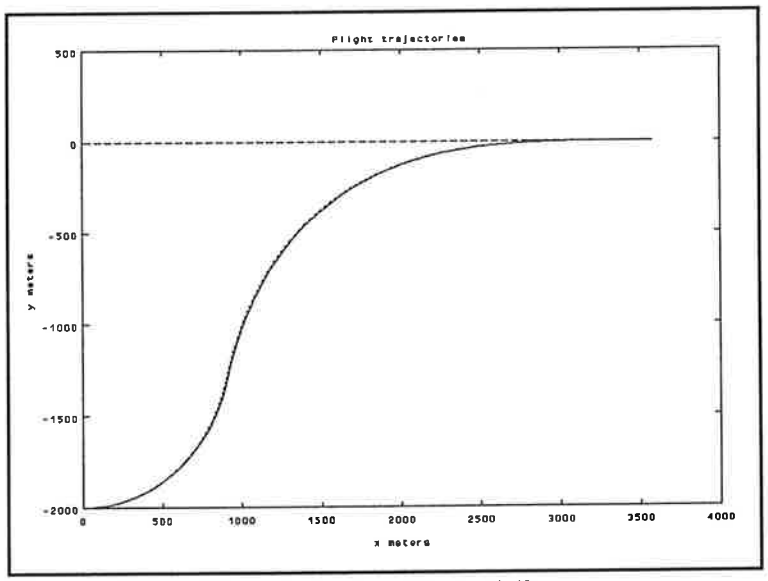

Figure 12 Flight trajectories. Dashed line: target. Dotted line: control law aggressor. Solid line: neural network aggressor.

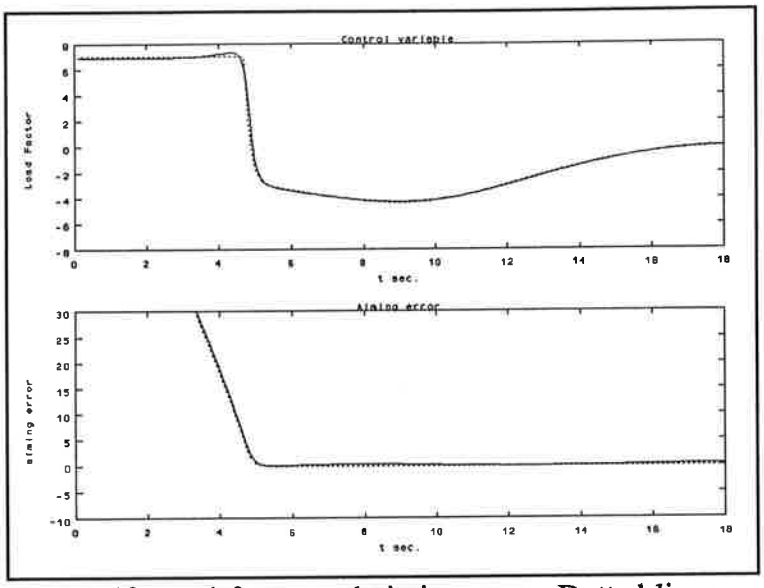

Figure 13 Load factor and aiming error. Dotted line: control law. Solid line: neural network.

Figure 12 and 13 show a simulation. The performance is very close to the desired control law. The performing index is now down to t_i=5.0 s. which is equal to the control law. The change of state description gave an unexpected performance change.

The network performs well also for other initial states. Figure 14-15 shows a simulation with a head on encounter situation.

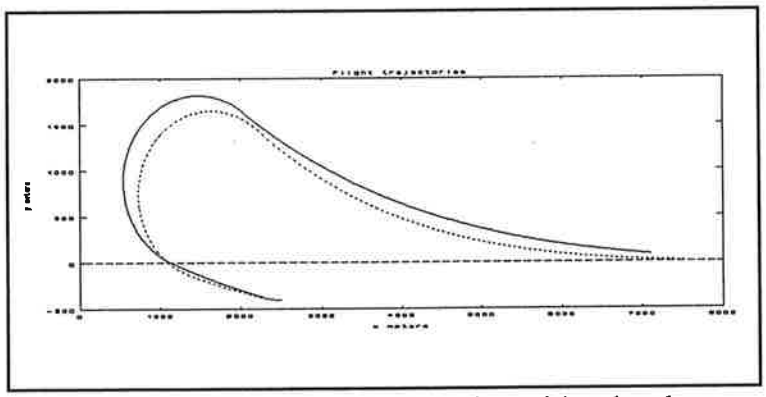

Figure 14 Resulting flight trajectories with a head on encounter initial position. Dashed line: target. Dotted line: control law aggressor. Solid line: neural network aggressor.

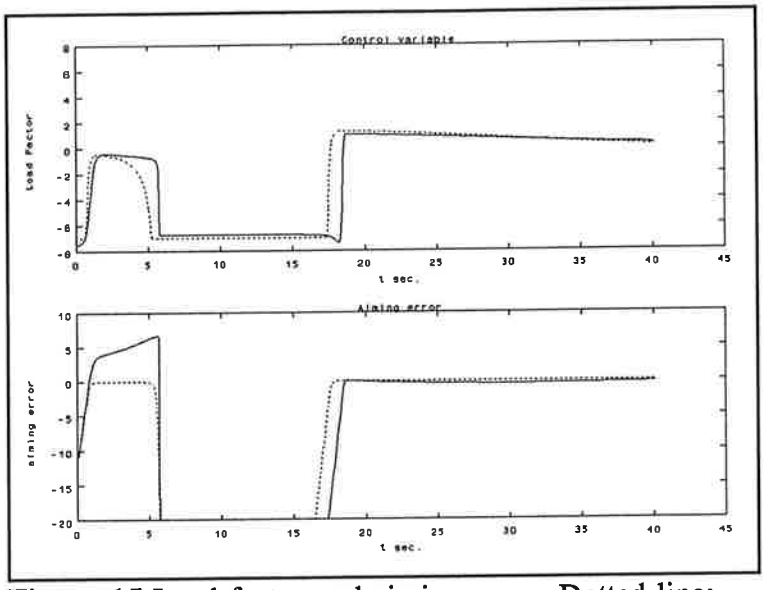

Figure 15 Load factor and aiming error. Dotted line: control law. Solid line: neural network control.

3.9 Summary

In this section neural networks have been applied to a regulation problem. The results indicate that a neural network can be used to develop a nonlinear feedback control law for an aiming problem. The neural network used had 2 hidden layers, each with 5 neurons. The learning set was compiled using different flight states and the corresponding correct control, which was obtained with an analytical control law. Some experimentation was required in order to choose the right state description. The design of learning material is also of great importance.

4. Optimal Flight Trajectories

The problem of finding the control that maximizes a performance index given an initial state and a fixed final time is a different problem class. For a linear system an optimal time varying feedback control can be explicitly calculated by solving a Riccati matrix equation. For a nonlinear system with constraints, current mathematical theory cannot be used to obtain an analytical solution, hence a numerical method must be used. The solution obtained is an open loop control program. Still it is often desirable to find a feedback control law which gives the same optimal control.

Optimal flight trajectory problems has a performance index only dependent on the final state. The problem is then to find the controls along a trajectory that gives a final state which maximizes the performance index. This control problem type is called terminal control.

In optimal trajectory problems it is of interest to find methods and algorithms solving optimization problems on-line that are possible to use in an airborne computer. One possibility is to use iterative numerical methods to find the optimal control. These methods usually require a significant amount of both memory and computer power to be possible to use on-line.

The use of a neural network, trained with open loop optimal flight trajectories, as a nonlinear control law, seems to be one solution. Learning material is produced using a differential dynamic programming method giving the optimal control for a number of trajectories. The initial states of the trajectories are chosen so that a large number of flight states in the flight envelope are covered by the optimal trajectories.

The neural network fills the criteria of low memory requirements and a moderate number of computations which are necessary to by implemented in today's airborne computers.

4.1 Aircraft Model

The aircraft model is based on a modern fighter aircraft. The aircraft is modelled using the following states: velocity v, altitude z, horizontal position x, *mass* and climb angle γ.

$$\dot{x}=v\cos\gamma \tag{40}$$

$$\dot{z}=v\sin\gamma \tag{41}$$

$$\dot{v}=\frac{Thr(z,M)-D(z,M,n)}{mass}-g_0\sin\gamma \tag{42}$$

$$\dot{\gamma}=\frac{g_0}{v}(n-\cos\gamma) \tag{43}$$

$$\dot{mass}=-Sfc(z,M)Thr(z,M) \tag{44}$$

where M is the Mach number, *Thr* is the thrust produced by the motor running on full throttle, D is the drag component composed of induced drag and zero lift drag, g_0 is the acceleration due to gravity, *Sfc* is the specific fuel consumption and n is the load factor on the wing and is used as the independent control variable. *Thr*,*D* and *Sfc* are functions based on a realistic model of a generic aircraft.

4.2 Optimal Energy Climb

As an example of this class of problems we have chosen the classical optimal climb problem. Given an initial state and a fixed final time, the aircraft's sum of kinetic and potential energy E_f at the final time, is to be maximized.

$$E_f=z(t_f)+\frac{v(t_f)^2}{2g_0} \tag{45}$$

E_f is the equivalent energy in meters and t_f is the fixed final time.
Due to the nonlinear properties of the problem the optimal trajectories are dependent on both the state at t_0 and the final time t_f.

4.3 Specific Excess Power (SEP)

The performance of an aircraft over the flight envelope is well represented by the specific excess power. Combining Eq. (41) and (42) gives:

$$\dot{v}=\frac{Thr-D}{mass}-g_0\sin\gamma=\frac{Thr-D}{mass}-\frac{g_0}{v}\dot{z} \tag{46}$$

SEP is defined as the derivative of Eq. (45). Using Eq. (46) gives:

$$SEP=\dot{E}=\frac{\dot{v}v}{g_0}+\dot{z}=\frac{Thr-D}{mass\,g_0}v \tag{47}$$

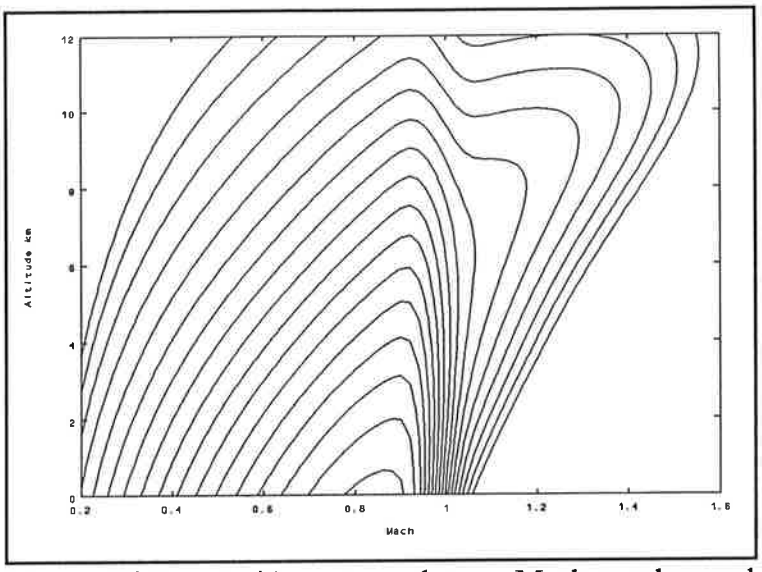

Figure 16 SEP (m/s) contour plot vs. Mach number and altitude (n=1).

Figure 16 shows the SEP contour plots for the aircraft model used.

4.4 DDP-Optimization Method

The numerical method used to produce the optimal flight trajectories is a modified first order differential dynamic programming method. In ref. [4] the method is described for a similar problems using a more advanced aircraft model. The method is iterative and computational intensive. The nonlinear properties of the problem sometimes give convergence problems. Thus the method must be manually monitored when used.

4.5 Experiment 1

The first problem to be examined is with a fixed final time t_f=120 s.

The inputs to the neural network are chosen as the state of the aircraft, z, v, and γ, and time elapsed. The mass state is not included since the mass change during the flight time is only 5%. The output is the load factor n, which is the aircraft's control input. The first and second hidden layers have ten and five neurons respectively. The architecture is thus (4-10-5-1).

z_0 (km)	v_0 (m/s)
1	125
1	175
1	225
1	275
2	150
2	200
2	250
3	125
3	175
3	225
3	275
4	150
4	200
4	250
5	150
5	200
5	250
5	300

Table I Initial states for trajectories used in Example 1.

With the DDP-method a set of trajectories was created using the initial states in Table I. The initial states are distributed in the lower left corner of the flight envelope. Since the DDP-algorithm used works with a 1 second step size, the trajectories are given with 1 second increments. To reduce the number of points only every fourth point were included in the learning material giving a total number of 588 points from all trajectories.

A network was developed using the material above for learning. Using the technique previously described to avoid local minima, the global error G_e was reduced to 0.03.

A network's performance can be viewed in two different ways, as the network's recall precision in discrete points or as the final energy obtained using the neural network to control the aircraft model. The first one is used during learning and is the same as the global error G_e. The second one is the most relevant one for this problem. In order to compare different networks' performance, let us define

$$\Delta E = E_f^{opt} - E_f^{nn} \tag{48}$$

and

$$Q = \sqrt{\frac{1}{n}\sum_{k=1}^{n} \Delta E_k^2} \tag{49}$$

where ΔE is the difference in final energy between the neural network control and the optimal control. Q is the RMS value of ΔE over n different trajectories. If the neural network is optimal the Q value is zero.

z_0 (km)	v_0 (m/s)	E^{opt} (km)	ΔE (km)
1	150	16.11	-0.05
1	200	16.48	-0.01
3	200	16.68	-0.01
3	250	16.46	-0.07
4	225	15.25	-0.01
5	225	15.63	-0.05
			Q = 0.04

Table II Result from simulation of Experiment 1.

The produced network together with the aircraft model was simulated. Table II shows the results obtained using initial states not included in the learning material. The result shows that the neural network is close to the optimal control.

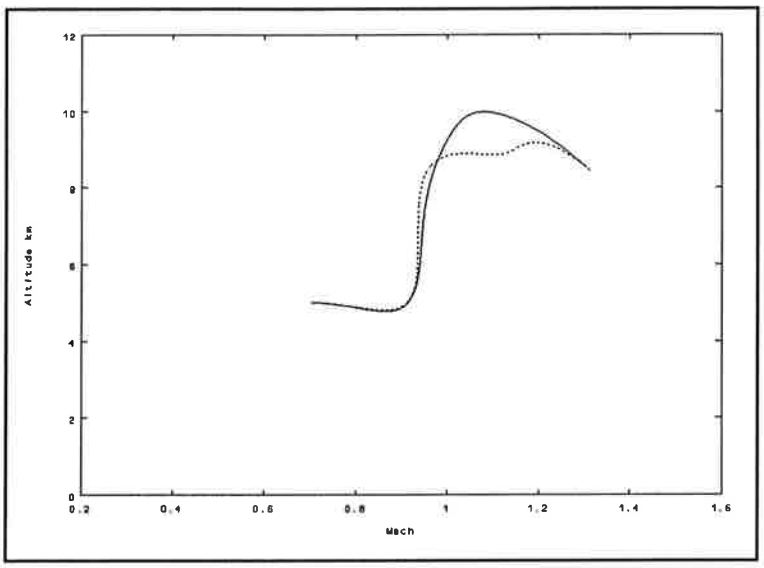

Figure 17 Flight trajectories with initial state z=5 km and v=225 m/s. Dotted line: optimal trajectory. Solid line: neural network trajectory.

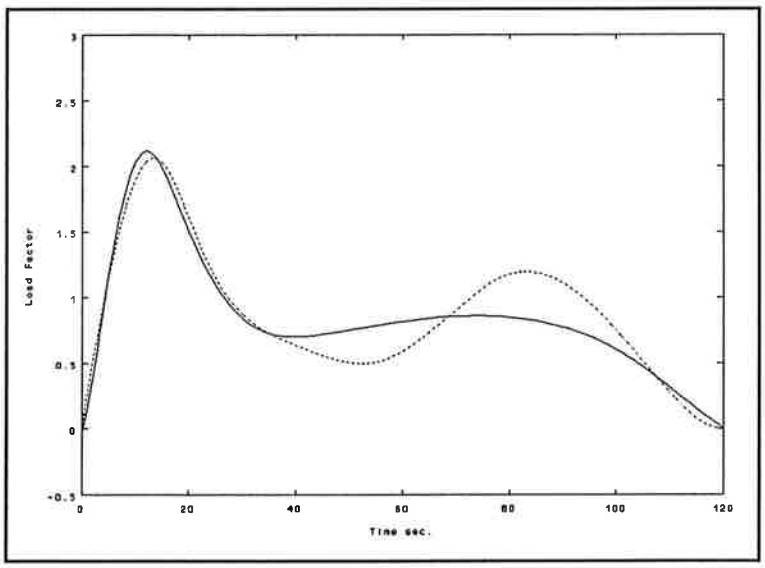

Figure 18 Load factor. Dotted line: optimal control. Solid line: neural network control.

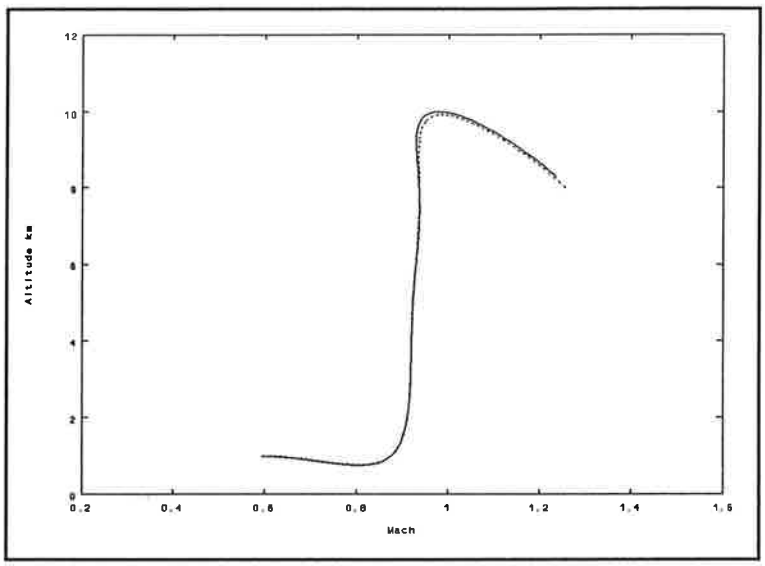

Figure 19 Flight trajectories with initial state z=1 km and v=200 m/s. Dotted line: optimal trajectory. Solid line: neural network trajectory.

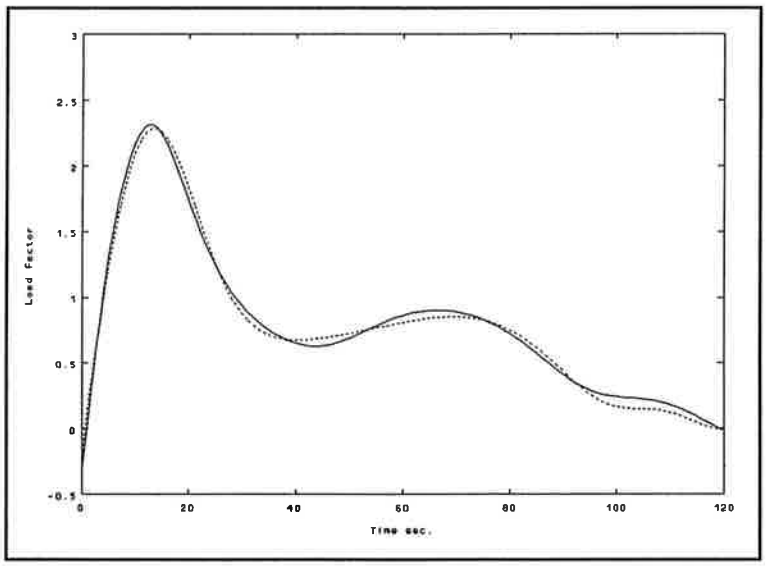

Figure 20 Load factor. Dotted line: optimal control. Solid line: neural network control.

Figure 17-20 shows the control and resulting trajectories for two different initial positions. The climb profile can be divided in three phases, speed increase, climb and a dive phase. The optimal speed to climb at is just below Mach 1. The final dive reduces the induced drag and increases the speed.

The simulation with initial state z=5 km and v=225 m/s results in a quite different trajectory produced by the neural network compared to the optimal trajectory. Investigating the SEP contour plot in Fig. 16 it is noted that the level

curves are more distant in the area were the two trajectories are different. This gives us a hint why the final energy for the neural network's trajectory still is close to optimal.

4.6 Experiment 2

In a real application, the situation continuously changes and a fixed final time problem formulation is not useful. To meet this argument the same problem as in Experiment 1 is to be solved but now with varying final time in the range of 100-140 seconds. This new problem formulation and the knowledge gained above led to a different time concept. Instead of having elapsed time as one input, we used the time to go as an input. This new time information gives, together with the current state, theoretically enough information to the network.

Using the initial states in Table I and using 100, 120 and 140 seconds final times, a set of new trajectories were constructed using the DDP-method. The learning material was composed of every sixth point from all trajectories, a total number of 1116 points.

After learning the global error G_e was reduced to 0.03.

Simulations showed that an almost perfect behaviour were obtained for final times of 100 and 120 seconds. Figure 21 and 22 show a simulation with initial state z=2 km, v=250 m/s and a final time of 120 seconds.

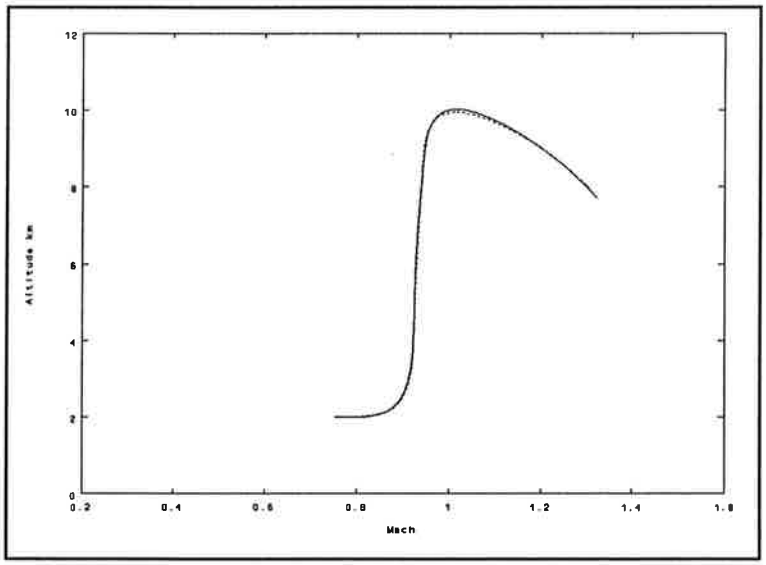

Figure 21 Flight trajectories with initial state z=2 km, v=225 m/s and t_f=120 s. Dotted line: optimal trajectory. Solid line: neural network trajectory.

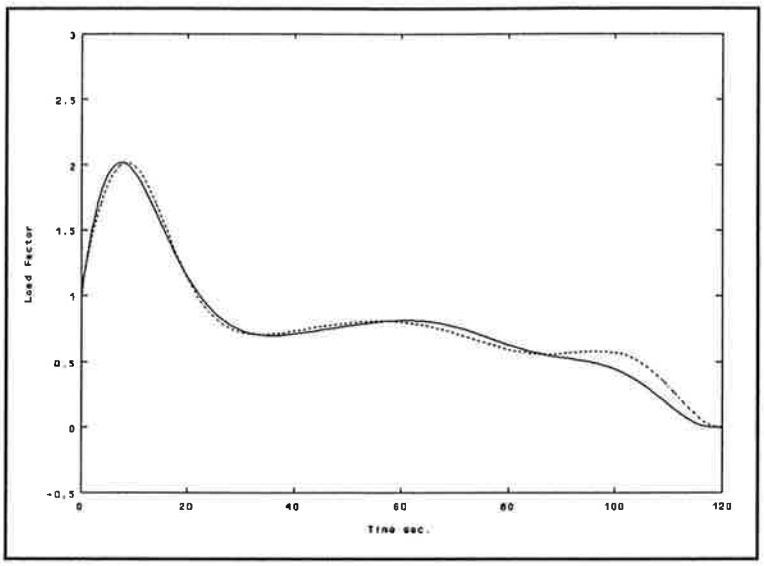

Figure 22 Load factor. Dotted line: optimal control. Solid line: neural network.

The trajectories are almost identical and ΔE=0.00. Other trajectories with a short final time look similar. An example of trajectories starting with a high energy (large altitude and/or speed) and a large final time are shown in fig 23 and 24. The initial state was z=3 km, v=275 m/s and final time 140 seconds.

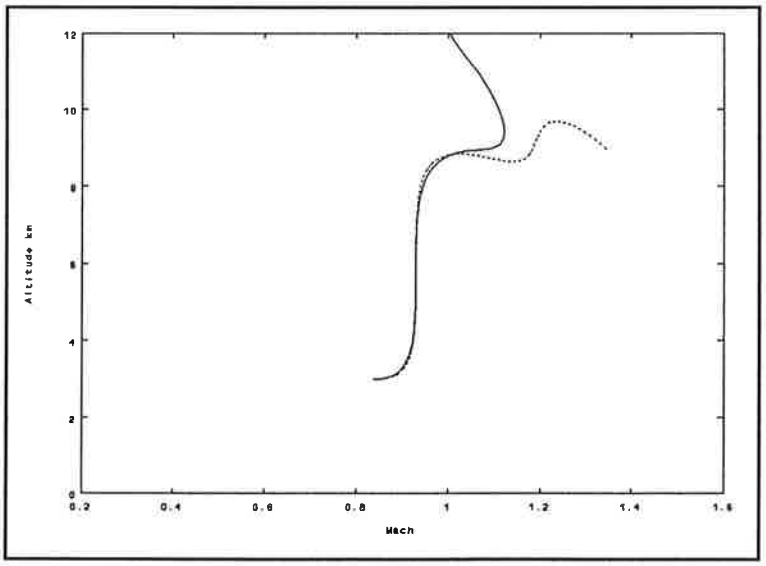

Figure 23 Flight trajectories with initial state z=3 km, v=275 m/s and t_f=140 s. Dotted line: optimal trajectory. Solid line: neural network trajectory.

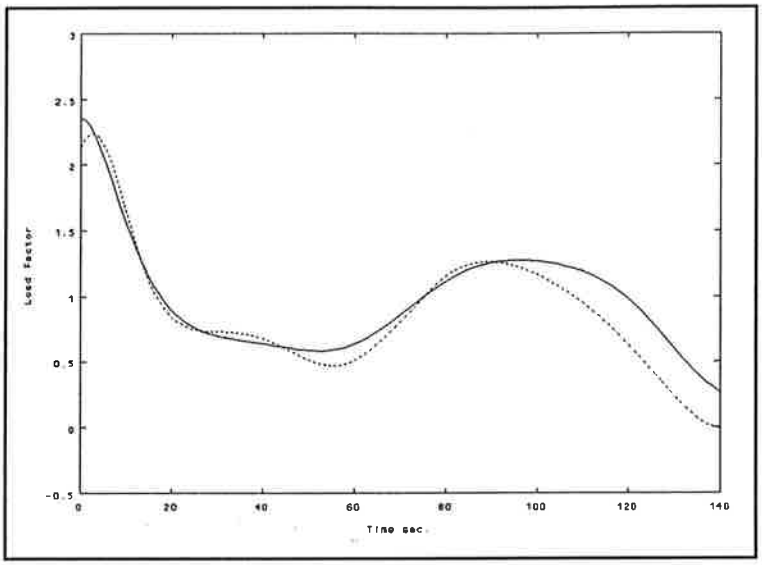

Figure 24 Load factor. Dotted line: optimal control. Solid line: neural network control.

The last part of the neural network's trajectory diverges. ΔE=-0.79. Here the network did not perform as well. One reason for this could be that the neural network did not manage to identify the nonlinear properties at high altitudes and high speeds. The SEP contour plots in Fig. 16 are different in this area compared to the contour plots at lower altitudes. It is obvious that the learning material used did not adequately cover the aircraft's behaviour at high speeds and high altitudes. Also the climbing process is slightly instable i.e. it is sensitive to small variations in the load factor, see Fig 24.

4.7 Experiment 3

The previous experiment indicated a need for more information in the upper part of the flight envelope. Including trajectories with initial states at higher altitudes and with longer final times in the learning material would increase the amount of information in this area to the network.

z_0 (km)	v_0 (m/s)
1	125
1	200
1	275
3	225
5	175
5	300
7	250
10	200
10	250
10	325

Table III Initial states used in Experiment 3.

The range of final times used in Experiment 2 is quite narrow. To extend this, a new set of 40 optimal trajectories were constructed using the initial positions in Table III each with 20, 80, 140 and 200 seconds final times. From this set of trajectories every sixth point were used yielding a total number of 760 points in the learning material.

A network with the same complexity as in Experiment 1 was trained with the new learning material. After some convergence help during learning the global error was reduced to 0.050.

z_0 (km)	v_0 (m/s)	t_f (s)	E^{opt} (km)	ΔE (km)
1.5	240	190	18.97	-0.01
2	110	150	16.61	-0.01
3	300	70	14.32	-0.03
4	180	110	15.70	-0.04
5	250	140	17.82	-0.03
6	200	20	10.08	-0.00
7	200	150	18.26	-0.04
8	250	200	20.57	-0.02
10	225	120	18.41	-0.02
10	300	50	16.94	-0.06
				Q = 0.03

Table IV Initial states and results for network in Example 3.

Simulations with the new network demonstrated an improvement for trajectories with long final time and high altitude/speeds, see Table IV. Compared with Experiment 2, the performance at low altitudes and short final times was also equally good. This implies that distributing the initial states in a larger area in the flight envelope and using a wider range of final times for learning gave an overall better performing network. The new learning material used gave more points in the upper, more difficult, region of the flight envelope and thus made it possible for the network to make a more accurate model of the aircraft.

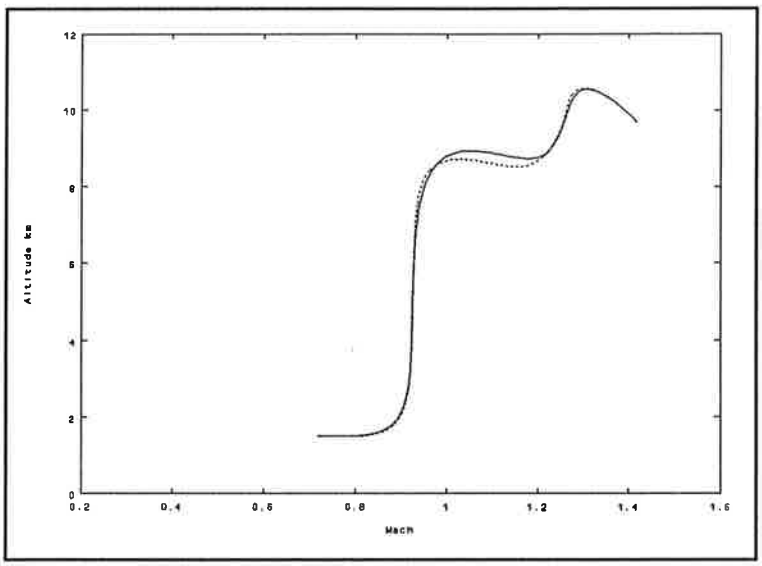

Figure 25 Flight trajectories with initial state z=1.5 km, v=240 m/s and t_f=190 s. Dotted line: optimal trajectory. Solid line: neural network trajectory.

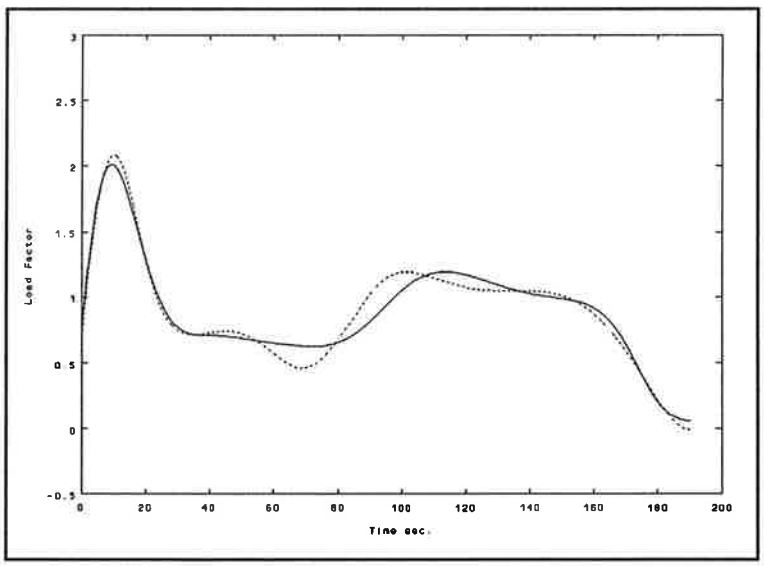

Figure 26 Load factor. Dotted line: optimal control. Solid line: neural network control.

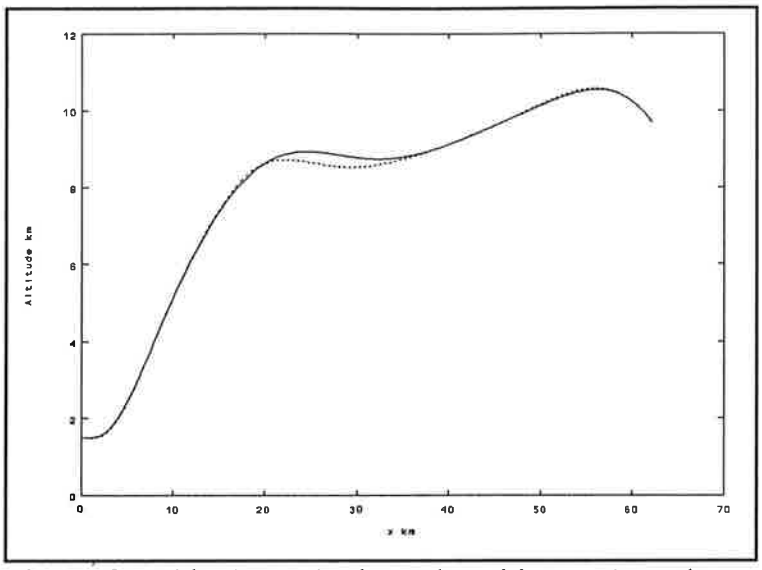

Figure 27 Altitude vs. horizontal position x. Dotted line: optimal trajectory. Solid line: neural network trajectory.

Simulation with initial state $z=1.5$ km, $v=240$ m/s and a 190 seconds final time is shown in Fig. 25-27. In spite of the long final time the final energy of the two trajectories is almost identical. The neural network's control is a little smoother than the optimal control. It seems as if the neural network has understood the basic line in the control but not understood the small variations in the control. This averaging has almost no effect on the final energy.

When investigating the plots of the control it is important to remember that the optimal control is given in open loop and the neural network works in a closed loop. That is, if the two aircraft after some time have different states, it is evident, that the neural network will produce a different control to make the aircraft's trajectory from that point as optimal as possible.

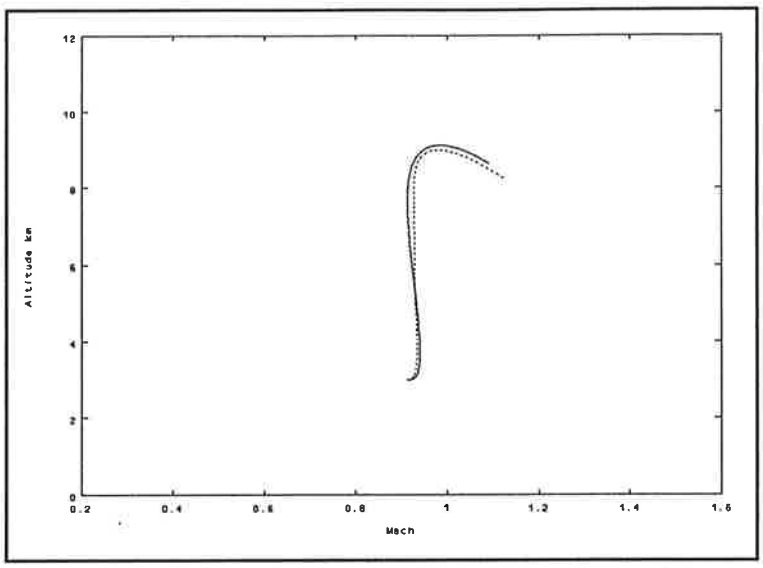

Figure 28 Flight trajectories with initial state z=3 km, v=300 m/s, t_f=70 s. Dotted line: optimal trajectory. Solid line: neural network trajectory.

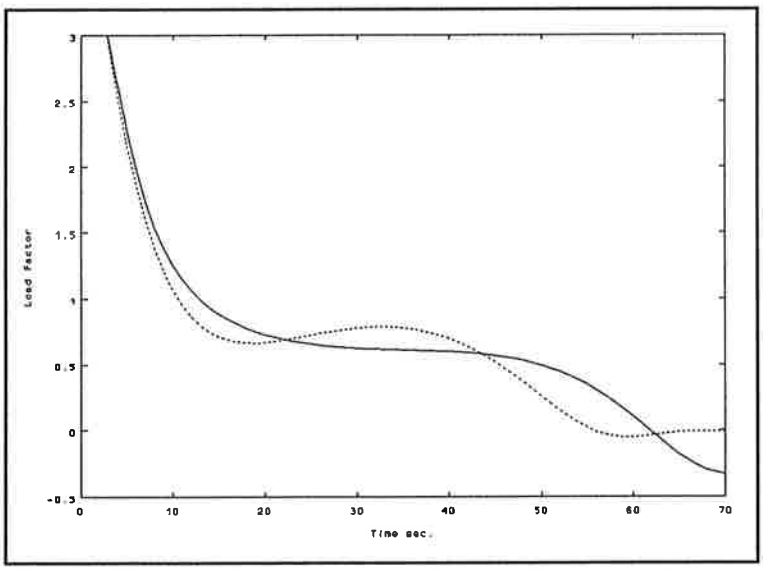

Figure 29 Load factor. Dotted line: optimal control. Solid line: neural network.

In Fig. 28 and 29 a simulation with initial state z=3 km, v=300 m/s and a final time of 70 seconds is shown. The control of the neural network is here again a little smoother compared to the optimal.

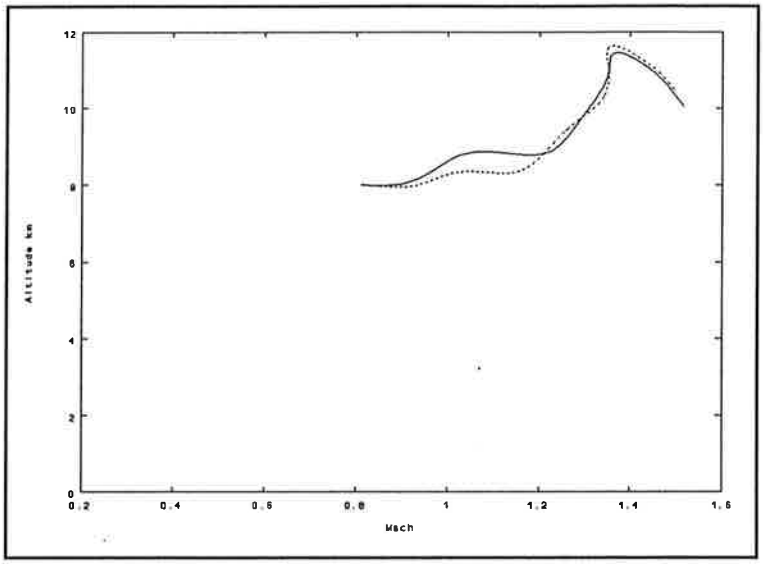

Figure 30 Flight trajectories with initial state z=8 km, v=250 m/s and t_f=200 s. Dotted line: optimal trajectory. Solid line: neural network trajectory.

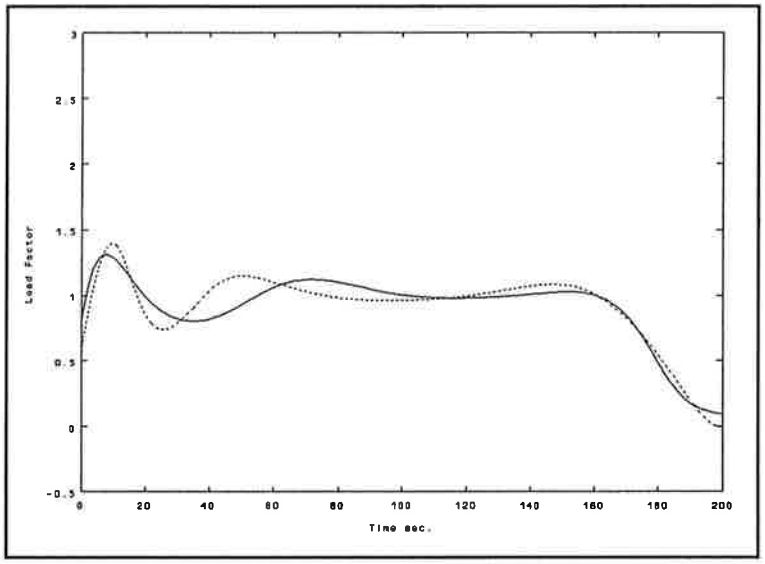

Figure 31 Load factor. Dotted line: optimal control. Solid line: neural network control.

Figure 30-31 show a simulation with initial state z=8 km, v=250 m/s and a final time of 200 seconds. This simulation shows a very different optimal trajectory compared to earlier experiments, due to the high altitude and the high final speed. The neural network performs very well, giving an almost optimal final energy. This simulation illustrates the ability of a fairly simple neural network to cover a wide range of flight states and final times.

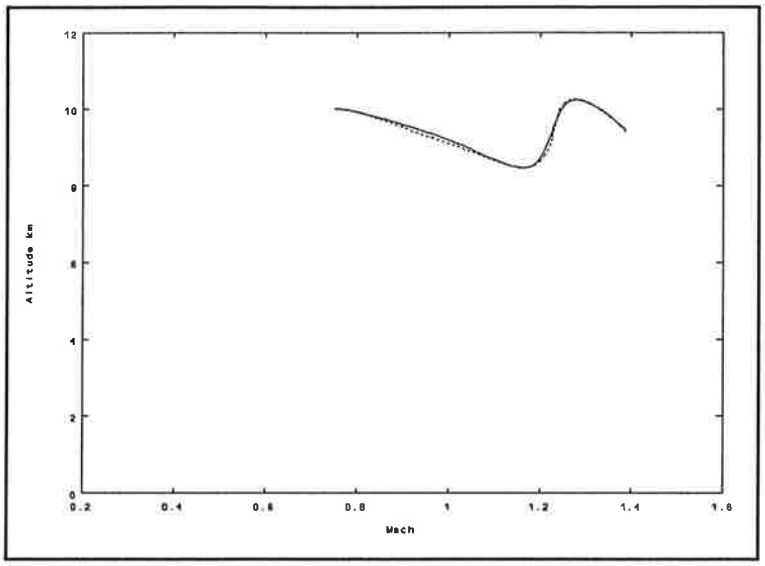

Figure 32 Flight trajectories with initial state z=10 km, v=225 m/s and t_f=120 s. Dotted line: optimal trajectory. Solid line: neural network trajectory.

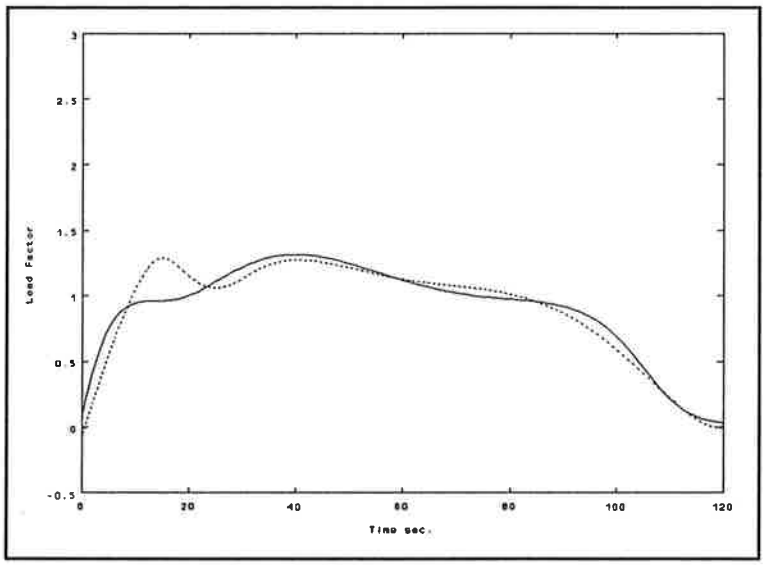

Figure 33 Load factor. Dotted line: optimal control. Solid line: neural network control.

Simulation with initial state z=10 km, v=225 m/s and 120 seconds final time is illustrated in Fig. 32 and 33.

4.8 Network Complexity

In the previous section the same network architecture (4-10-5-1) was used for all experiments. In order to demonstrate different complexities and the effect on the final performance, a number of networks were constructed using the same learning material described in Table III.

Architecture	G_e	Q (km)
4-5-3-1	0.082	0.28
4-5-5-1	0.060	0.061
4-10-5-1	0.050	0.031
4-10-10-1	0.042	0.030

Table V Results using different neural network complexity.

The results in Table V are as expected. A smaller number of neurons gives a less optimal performance and increasing the number of neurons gives a small change in the performance index Q. The change in Q is very small but the complex network shows a greater improvement when comparing the optimal trajectories with the trajectories produced by the network. Using a more complex network involves more weights and hence it is more probable that local minima occur during learning.

4.9 Learning Material

An interesting practical question is: How much learning material is needed to achieve a good network performance? The learning material used above was made up of 40 trajectories sampled every sixth second. Three new learning materials are constructed using the same trajectories now using different sample times. This gives us material with different numbers of input/output pairs.

t_{samp} (s)	total	G_e	Q (km)
10	480	0.051	0.13
6	760	0.050	0.031
4	1140	0.045	0.047
1	4440	0.045	0.031

Table VI Results using different sample times in the learning material.

The results are given in Table VI. Using a too large sample time decreases the amount of information the network get access to which in turn degrades the performance. Using a smaller sample time and thus increasing the information, the network's performance stays at the same level as before. The performance value achieved for the network trained with data sampled with 4 seconds sampling time is probably due to non optimal learning. When working with a neural network it is hard to know during learning if the best possible performance is reached or learning has just stopped due to a local minimum.

4.10 Summary

In this section neural networks have been applied to a nonlinear optimal control problem. A differential dynamic programming technique was used to produce a set of optimal trajectories, including the open loop control, used as learning material. A neural network with two hidden layers with 10 and 5 neurons respectively gave a good performance for the problem studied. The results indicate the importance of including flight states covering the entire state space in the learning material in order to obtain a robust neural network control law.

5. Conclusions

In this thesis neural networks have been used to generate nonlinear control laws for two different flight control problems: A simple aiming problem and an optimal climb problem. Both problems are nonlinear and no analytical solution exists for the second problem. The networks are trained with examples of the correct control for a wide range of states using the back-propagation learning algorithm. Simulations have given evidence that the neural networks work very well for the two applications studied. Only a small network was needed to achieve a good performance, typically 10 to 15 neurons organized in two hidden layers. An increase of the network's complexity gave only a marginal increase of performance for the optimal climb problem. The moderate computational load of a neural network implementation make it possible for implementation in an airborne computer.

During the design of the networks it is of great importance to be aware of the existence of local minima and to take appropriate actions to avoid them. It is also important to evaluate the actual behaviour with the neural network controller and not simply with the global error. This also indicates possible improvements of the learning material to obtain a better performance. The computational load of learning has been quite moderate for the problems studied and does not seem to be a limitation in the use of neural networks. Since current theory does not give straight answers about network complexity and learning material needed for a given problem, a lot of experimentation is needed in order to get good results. Thus, it is sometimes more an art than a science to design well performing neural networks.

In order to gain more knowledge about the possibilities of using neural network control laws in real applications, more control problems with different optimization goals should be explored. The complexity of the problems could also be increased by formulating problems using a state description in three dimensions.

6. References

[1] NeuralWare Inc. "Neural Computing", manual to program package NeuralWorks Professional II/PLUS, 1991.

[2] Nguyen Derrick H. and Widrow Bernard, "Neural Networks for Self-Learning Control Systems", *IEEE Contr. Syst. Mag.* vol. 10 April 1990.

[3] Järmark Bernt and Speyer Jason, "Optimal Fuselage Aiming With Decoupled Short Period Dynamics", AIAA-88-4156-CP, GNC Conference, Minneapolis, Minnesota, 15-17 August, 1988.

[4] Järmark Bernt, "Various Optimal Climb Profiles" AIAA-91-2859-CP pp. 124-130, AFM Conference, New Orleans, Lousiana, 12-14 August, 1991.