NUMERICAL SOLUTION OF A CALCULUS OF VARIATIONS PROBLEM USING THE FEEDFORWARD NEURAL NETWORK ARCHITECTURE

Andrew J. Meade, Jr.* and Hans C. Sonneborn†
Department of Mechanical Engineering
and Materials Science
Mail Stop 321
Rice University
Houston, Texas, 77251-1892, USA
Phone: (713) 527-8101 ext. 3590
FAX: (713) 285-5423
email: meade@rice.edu

To appear in *Advances in Engineering Software*, Vol. 27 No.3.

---

* Associate Professor. Recipient of correspondence. email: meade@rice.edu
† Research Assistant.
This work is supported under NASA Grant NAG 1-1433.

NUMERICAL SOLUTION OF A CALCULUS OF VARIATIONS PROBLEM USING THE FEEDFORWARD NEURAL NETWORK ARCHITECTURE

Andrew J. Meade, Jr. and Hans C. Sonneborn

Department of Mechanical Engineering and Materials Science, Rice University, Houston, TX 77251-1892, U.S.A.

Key Words: Feedforward Artificial Neural Networks, Neural Computation, Calculus of Variations

Abstract

It is demonstrated, through theory and numerical example, how it is possible to construct directly and noniteratively a feedforward neural network to solve a calculus of variations problem. The method, using the piecewise linear and cubic sigmoid transfer functions, is linear in storage and processing time. The L_2 norm of the network approximation error decreases quadratically with the piecewise linear transfer function and quartically with the piecewise cubic sigmoid as the number of hidden layer neurons increases. The construction requires imposing certain constraints on the values of the input, bias, and output weights, and the attribution of certain roles to each of these parameters.

All results presented used the piecewise linear and cubic sigmoid transfer functions. However, the noniterative approach should also be applicable to the use of hyperbolic tangents and radial basis functions.

NUMERICAL SOLUTION OF A CALCULUS OF VARIATIONS PROBLEM USING THE FEEDFORWARD NEURAL NETWORK ARCHITECTURE

Andrew J. Meade, Jr. and Hans C. Sonneborn

Department of Mechanical Engineering and Materials Science, Rice University, Houston, TX 77251-1892, U.S.A.

NOMENCLATURE

Symbols	Explanation
x	independent variable
y	dependent variable
T	total number of hidden layer nodes
Υ_q	transfer function of the qth hidden layer node
c_q	output weight of the qth hidden layer node
ξ, ξ_q	combination of input variable, input weight, and bias weight
B_{2n}	Bernoulli numbers
α_q	input weight of the qth hidden layer node
θ_q	bias weight of the qth hidden layer node
Φ_i	spline centered on the ith knot
x_i	value of independent variable at the ith knot
h	spacing of knots in a uniform grid
a, b	lower and upper bounds of the problem domain
$\Upsilon_i^A, \ldots, \Upsilon_i^F$	transfer functions used to construct the ith spline
$\alpha_i^A, \ldots, \alpha_i^F$	input weights used for construction of the ith spline
$\theta_i^A, \ldots, \theta_i^F$	bias weights used for construction of the ith spline
N	number of basis functions used in solution approximation
$u1_i, \ldots, u6_i$	output weights used in construction of ith spline
$\tilde{\mathbf{w}}$	vector of basis coefficients associated with Φ_i
L	differential operator acting on y
$\tilde{\mathbf{s}}$	vector of independent variables
g	algebraic operator acting on $\tilde{\mathbf{s}}$
R	equation residual
F	weight function
(F, R)	integral inner product defined in Eq.(16)
F_k	kth weight function

p_1, p_2	pump outlet pressure and vessel pressure
Δp	change in pressure
m	mass (kg)
t	time (s)
dm/dt	mass flow rate (kg/s)
P	Power (W)
$d\gamma/dt$	ideal pump flow rate (kg/s)
I	total pumping power
β	constant defined in Eq.(19)
t_{max}	maximum allowed time
τ	normalized time
C_1	constant defined in Eq.(21)
$\mathbf{M1}, \mathbf{M2}, \mathbf{M3}$	matrices
$\tilde{\mathbf{b}}$	vector
$K_1, \ldots, K_4$	scalar coefficients
L_2	square integrable function subspace
E	function error
$\|\|E\|\|$	discrete L_2 norm of function error
C	arbitrary constant

Subscripts

a	network approximation
i, j, k, q	indices

Superscripts

$A, B, \ldots, F$	indices

Note: The symbols defined above are subject to alternation on occasion.

NUMERICAL SOLUTION OF A CALCULUS OF VARIATIONS PROBLEM USING THE FEEDFORWARD NEURAL NETWORK ARCHITECTURE

Andrew J. Meade, Jr. and Hans C. Sonneborn

Department of Mechanical Engineering and Materials Science, Rice University, Houston, TX 77251-1892, U.S.A.

INTRODUCTION

At present an area of increasing interest is the emulation and control of engineering systems by the feedforward artificial neural network (FFANN) architecture. These particular networks use supervised learning algorithms, such as the popular backpropagation method, which allow the network to simulate a system without knowledge of the governing equations. However, since supervised learning algorithms require exposure to numerous training data sets, they can become memory intensive and time consuming without the guarantee of success.

We believe it is possible to embed mathematical models of physical processes directly into the FFANN architecture without resorting to training sets and iterative schemes by starting with two basic assumptions that force a slight paradigm shift: first, that FFANN learning is actually function approximation; and second, that FFANNs can be manipulated as well as conventional numerical techniques. Consequently it should be possible to construct FFANNs that can perform tasks with accuracy and computational efficiency and would therefore be as applicable to hard computing as they are to soft computing (pattern recognition and the like).

The second assumption follows naturally from the first, since all numerical techniques used in computational mechanics can be considered to be methods of function approximation, although they use integral or differential equations rather than data sets. A number of researchers, among them [1]-[3], have published work operating under the first assumption for artificial neural networks (ANNs) in general, but few [4]-[8] have investigated the second. To prove the latter assumption a full investigation must be made of the parameters that govern FFANN performance such as connection weights, activation functions, types of neurons (additive or multiplicative), and connection schemes. The most straightforward and logical approach to this investigation is to develop the mathematics of FFANNs by solving well-posed problems of increasing complexity.

The approach taken by the authors then is to transform the FFANN, via suitable weight constraints, into a directed-graph representation of a functional basis expansion. Under this paradigm the activation functions are effectively the basis functions for the expansion and are defined and distributed in the input space domain by the input and bias weights. The output weights then take on the role of the expansion coefficients. Evaluation of the FFANN parameters can be made as reliable and predictable as standard numerical techniques because constraint of the weights allows us to use standard numerical techniques on the network without violating in any way the "rules" of the FFANN architecture. This makes it possible to apply a large body of both theoretical and practical knowledge, hitherto applied to function approximation, to the problems of training and generalization. More specifically, we can apply a powerful function approximation technique known as the method of weighted residuals [9] and show how it can be made to operate directly on the net architecture.

It would be prudent to test the capabilities of the equation embedding approach on the solution of problems whose behavior is known and which have some practical interest. Our specific objective, and the topic of this paper, is to demonstrate through theory and numerical example that it is possible to construct a FFANN, using the calculus of variations, that provides a pump with the mass flow rate schedule that minimizes the pumping energy required for a specific application. To achieve this, a functional basis expansion will be applied to the Euler-Lagrange equation [10]. The input and bias weights of this network will be determined by explicit formulae, and the output weights by the evaluation of a system of linear algebraic equations. The resulting network, utilizing both the piecewise linear transfer function and the piecewise cubic sigmoid, is topologically identical to those that use conventional training techniques. It will also be demonstrated how the error of the constructed FFANNs can be predicted and controlled. This work is a preliminary step in constructing intelligent databases and controllers for fluid-thermal engineering systems by merging computational mechanics and experimental results, in the neural network structure.

FUNCTION APPROXIMATION

The most common method of function approximation is the functional basis expansion. The general mathematical expression for the approximation of the dependent variable y is

$$y_a(x) = \sum_{q=1}^{T} \Upsilon_q(x)\ c_q \tag{1}$$

where y_a is the approximation of the function, T represents the number of basis functions used, c_q the basis coefficients, and $\Upsilon_q(x)$ represents a set of linearly independent basis functions.

Equation (1) is a scalar product of the basis functions and expansion coefficients. If we similarly restrict the FFANNs to the univariate case with linear output units, it is clear that the mathematical operations performed by the simple FFANN of Fig. 1 can be analogously interpreted as a scalar product of its transfer functions and its output weights.

By appealing to function approximation theory, the transfer functions and weight arrangements can be chosen to optimize the FFANN's performance in representing functional relationships. Further, roles can be assigned to the various parameters in the network. For example, by treating the output weights as expansion coefficients their values should in principle be computable by a host of conventional techniques. Similar insights are found in Cybenko [11].

For this analysis successive approximations to the popular transfer function, the hyperbolic tangent or *tanh*, have been used. These approximations all follow from the following series expansion for the hyperbolic tangent [12]:

$$\tanh(\xi) = \xi - \frac{1}{3}\xi^3 + \frac{2}{15}\xi^5 - ... + \frac{2^{2n}(2^{2n}-1)B_{2n}}{(2n)!}\xi^{2n-1} \pm ..., \quad |\xi| < \frac{\pi}{2} \tag{2}$$

where B_{2n} are the Bernoulli numbers. We have employed the piecewise linear transfer function [13] of Fig. 2 and the cubic polynomial approximation to the hyperbolic tangent, or cubic sigmoid, (Fig. 3) in our research. The piecewise polynomial functions were chosen for the following reasons:

- Contrary to the conventional wisdom in the connectionist field, function approximation theory shows that the smoothest functions (e.g. hyperbolic tangents, sigmoids) are not necessarily the most accurate or computationally efficient. Use of global analytic functions and/or higher order polynomials can cause snaking and Gibbs phenomena unless special steps (e.g. Chebyshev spacing) are taken [14].

- The use of hyperbolic tangents and sigmoids have come from the demands of the backpropagation algorithm which required smooth derivatives of the transfer functions [16], not from considerations of optimality in terms of function approximation.

- The piecewise linear function is particularly attractive due to its simplicity and ease of implementation in software and hardware models [15].

Inspection of Fig. 1 shows that the transfer function of hidden node q operates on the sum of the weighted input and bias values ($\xi_q = \alpha_q x + \theta_q$), i.e., a linear transformation of the independent variable x. This notation is similar to the unscaled shifted rotations of Ito [17]. Using ξ_q, the piecewise linear function can be expressed as

$$\begin{aligned} \Upsilon_q &= -1.0 \quad \text{for} \quad \xi_q < -1.0 \\ \Upsilon_q &= \xi_q(x) \quad \text{for} \quad -1.0 \leq \xi_q \leq 1.0 \\ \Upsilon_q &= +1.0 \quad \text{for} \quad \xi_q > +1.0 \end{aligned} \tag{3}$$

and the cubic sigmoid is expressed as

$$\begin{aligned} \Upsilon_q &= -1.0 \quad \text{for} \quad \xi_q < -1.0 \\ \Upsilon_q &= \frac{3}{2}\left(\xi_q(x) - \frac{1}{3}\xi_q^3(x)\right) \quad \text{for} \quad -1.0 \leq \xi_q \leq 1.0 \\ \Upsilon_q &= +1.0 \quad \text{for} \quad \xi_q > +1.0 \end{aligned} \tag{4}$$

Basis functions in function approximation literature are said to be either local (exhibiting a nonzero value over a limited part of the domain only) or global (nonzero almost everywhere in the domain they span). The piecewise linear and piecewise cubic transfer functions may be viewed as global polynomial basis functions defined in a piecewise manner with the aid of the transformation into ξ_q. All global basis functions have similar weaknesses including ill-conditioning when solving for expansion coefficients and poor approximation between discretization points (knots) [18].

Many of the problems commonly associated with FFANN representation could arise simply from the use of a mathematically ill-suited basis approximation. In function approximation theory, the disadvantages of global basis functions are avoided by using piecewise polynomial splines. We will make use of the Chapeau, or "hat" function, as shown in Fig. 4, and cubic spline (Fig. 5) defined as

$$\begin{aligned}
\Phi_i &= \frac{(x - x_{i-1})}{(x_i - x_{i-1})}, \quad x_{i-1} \le x \le x_i \\
\Phi_i &= \frac{(x_{i+1} - x)}{(x_{i+1} - x_i)}, \quad x_i \le x \le x_{i+1} \\
\Phi_i &= 0, \quad x < x_{i-1} \quad \text{or} \quad x > x_{i+1}
\end{aligned} \tag{5}$$

and

$$\Phi_i = \frac{1}{h^3} \times \begin{cases} (x - x_{i-1})^3 & \text{if} \quad x_{i-2} \le x \le x_{i-1} \\ h^3 + 3h^2(x - x_{i-1}) + 3h(x - x_{i-1})^2 - 3(x - x_{i-1})^3 & \text{if} \quad x_{i-1} \le x \le x_i \\ h^3 + 3h^2(x_{i+1} - x) + 3h(x_{i+1} - x)^2 - 3(x_{i+1} - x)^3 & \text{if} \quad x_i \le x \le x_{i+1} \\ (x_{i+1} - x)^3 & \text{if} \quad x_{i+1} \le x \le x_{i+2} \\ 0 & \text{otherwise,} \end{cases} \tag{6}$$

respectively. The h is the constant spacing between the knots used to discretize the domain. Note that the value of Eq.(5) varies between the values of 0 and 1 and Eq.(6) varies between the values of 0 and 4.

Transforming Piecewise Linear and Cubic Transfer Functions into Splines

Linear Splines

The mathematical advantages of the polynomial splines suggest it would be advantageous to transform the FFANN from a global transfer function representation to a spline based representation of the data.

Consider Fig. 6. If the function centered between x_i and x_{i+1} is multiplied by -1 and added to the one centered between x_{i-1} and x_i the result will be a hat function with a maximum value of 2. Using the following notation

$$\Upsilon_i^A = \Upsilon_i(\alpha_i^A x + \theta_i^A) \tag{7}$$

the transformation of piecewise linear functions to hat functions can be described by the following equation:

$$\Upsilon_i^A - \Upsilon_i^B = 2\Phi_i, \quad (i = 1, 2, ..., N) \tag{8}$$

where the superscripts of the piecewise linear functions, A and B, refer to adjoining intervals $x_{i-1} \leq x \leq x_i$ and $x_i \leq x \leq x_{i+1}$, respectively. A consequence of this formulation is that the number of piecewise linear functions can be linked to the number of hat functions desired. Our approach requires twice as many piecewise linear functions as hat functions ($T = 2N$).

To determine the remaining constraints required to equate the piecewise linear transfer function representation (Eq.(1)) to one using the hat functions, the problem domain $[a, b]$ is discretized using the knots x_i in the following manner:

$$a = x_1 < x_2 < ... < x_{N-1} < x_N = b$$

We will label this discretization as the problem mesh. To generate the appropriate hat functions at the boundaries, two auxiliary knots x_0 and x_{N+1} are required such that $x_0 < a$, and $x_{N+1} > b$. Using the constraint, $T = 2N$, Eq.(1) can be rewritten as

$$\sum_{q=1}^{T} \Upsilon_q(x)\ c_q = \sum_{i=1}^{N} \Upsilon_i^A u_i + \sum_{i=1}^{N} \Upsilon_i^B v_i \tag{9}$$

It can then be shown [19] that the basis expansion in terms of piecewise linear functions can be transformed into a basis expansion in terms of hat functions by selecting the following values for the FFANN weights of Fig. 1. For the input weights and bias weights:

$$\begin{aligned} \alpha_i^A &= \frac{2}{(x_i - x_{i-1})} \quad , \qquad \alpha_i^B = \frac{2}{(x_{i+1} - x_i)} \\ \theta_i^A &= -\frac{(x_i + x_{i-1})}{(x_i - x_{i-1})} \quad , \qquad \theta_i^B = -\frac{(x_{i+1} + x_i)}{(x_{i+1} - x_i)} \end{aligned} \tag{10}$$

where α_i^A, θ_i^A, α_i^B, and θ_i^B correspond to the piecewise linear functions of Eq.(8). For the output weights:

$$\begin{aligned} c_i = u_i &= \frac{w_i}{2}, \quad (i = 1, 2, ..., N) \\ c_{i+N} = v_i &= \frac{-w_i}{2}, \quad (i = 1, 2, ..., N) \end{aligned} \tag{11}$$

The application of Eqs.(8)-(11) permits the following equivalence:

$$y_a(x) = \sum_{q=1}^{T} \Upsilon_q(x)\ c_q = \sum_{i=1}^{N} \Phi_i(x) w_i \tag{12}$$

The hat functions Φ_i are defined as having a value of 1 at their respective knots x_i, as in Eq.(5), and w_i are the basis coefficients associated with the hat functions.

While the weights have been constrained in this approach, the architecture of the network has been preserved. As an example, Fig. 7 shows the resulting architecture for six processing elements used to form three hat basis functions. This type of univariate FFANN architecture has been used to model the solutions to linear and nonlinear ordinary differential equations [19], [20].

Cubic Splines

To form an arbitrary cubic spline centered at x_i, a cubic "bell" is constructed by adding two large opposed cubic sigmoids covering the interval $[x_{i-2}, x_{i+2}]$ and centered at the middle knot of each side (Fig. 8).

To form the cubic spline, two smaller negative bell distributions are added. These are formed by two additional pairs of opposed cubic splines, centered appropriately and scaled to fit the respective intervals, as in Fig. 9.

It is thus possible to form the cubic spline by superposition of six cubic sigmoids, three per side of the cubic spline. The equation for transforming cubic sigmoids to cubic splines can be written as

$$\Phi_i = 2\left(\Upsilon_i^A - \Upsilon_i^B\right) - 0.5\left(\Upsilon_i^C - \Upsilon_i^D\right) - 0.5\left(\Upsilon_i^E - \Upsilon_i^F\right)\ ,\ (i = 1, 2, \ldots, N) \tag{13}$$

The superscripts of the cubic sigmoid refer to the various intervals. Consequently A and B refer to the sigmoids located on the intervals $[x_{i-2}, x_i]$ and $[x_i, x_{i+2}]$, and superscripts C, D, E, and F refer to intervals $[x_{i-2}, x_{i-1}]$, $[x_{i-1}, x_i]$, $[x_i, x_{i+1}]$, and $[x_{i+1}, x_{i+2}]$, respectively. Note the sigmoids that span the entire spline are scaled by factors of two.

In a manner similar to that shown for the piecewise linear case the problem domain $[a, b]$ is discretized into the knots x_i.

$$x_1 < x_2 = a < \ldots < x_{N-1} = b < x_N$$

To properly define the transfer functions, two auxiliary knots are required on either end of the domain (x_1 and x_N).

In this scheme by restricting the discretization to uniform spacing (h = constant) there are six times as many cubic sigmoids as resultant cubic splines ($T = 6N$). The individual summations of weighted functions may be rewritten as

$$\sum_{q=1}^{N} \Upsilon_q c_q = \sum_{i=1}^{N} \Upsilon_i^A u1_i, \quad \sum_{q=N+1}^{2N} \Upsilon_q c_q = \sum_{i=1}^{N} \Upsilon_i^B u2_i, \quad \sum_{q=2N+1}^{3N} \Upsilon_q c_q = \sum_{i=1}^{N} \Upsilon_i^C u3_i$$

$$\sum_{q=3N+1}^{4N} \Upsilon_q c_q = \sum_{i=1}^{N} \Upsilon_i^D u4_i, \quad \sum_{4N+1}^{5N} \Upsilon_q c_q = \sum_{i=1}^{N} \Upsilon_i^E u5_i, \quad \sum_{q=5N+1}^{6N} \Upsilon_q c_q = \sum_{i=1}^{N} \Upsilon_i^F u6_i$$

Furthermore, to derive the input and bias weights when using the cubic sigmoid, the first priority is to distribute the cubic sigmoids in the problem space so that their cubic behavior occurs across the appropriate intervals. Therefore, by inspection

$$\Upsilon_i^A = \Upsilon\left(\frac{(x - x_{i-2})}{h} - 1\right) \quad , \quad x : [x_{i-2}, x_i] \tag{14}$$

$$\Upsilon_i^B = \Upsilon\left(\frac{(x - x_i)}{h} - 1\right) \quad , \quad x : [x_i, x_{i+2}]$$

$$\Upsilon_i^C = \Upsilon\left(\frac{2(x - x_{i-2})}{h} - 1\right) \quad , \quad x : [x_{i-2}, x_{i-1}]$$

$$\Upsilon_i^D = \Upsilon\left(\frac{2(x - x_{i-1})}{h} - 1\right) \quad , \quad x : [x_{i-1}, x_i]$$

$$\Upsilon_i^E = \Upsilon\left(\frac{2(x - x_i)}{h} - 1\right) \quad , \quad x : [x_i, x_{i+1}]$$

$$\Upsilon_i^F = \Upsilon\left(\frac{2(x - x_{i+1})}{h} - 1\right) \quad , \quad x : [x_{i+1}, x_{i+2}]$$

Referring to Eq.(13), we can determine the corresponding output weights $u1_i$, $u2_i$, $u3_i$, $u4_i$, $u5_i$, and $u6_i$, resulting in

$$\begin{aligned} u1_i &= 2w_i \ , \ u2_i = -2w_i \ , \ u3_i = -0.5w_i \ , \ u4_i = 0.5w_i, \\ u5_i &= -0.5w_i \ , \ u6_i = 0.5w_i \end{aligned}$$

The value of the input and bias weights can be calculated using Eq.(14)

$$\alpha_i^A = \frac{1}{h} \quad , \quad \theta_i^A = -\frac{(x_{i-2} + h)}{h} \ , \ \alpha_i^B = \frac{1}{h} \ , \ \theta_i^B = -\frac{(x_i + h)}{h}$$

$$\alpha_i^C = \frac{2}{h} \quad , \quad \theta_i^C = -\frac{(2x_{i-2} + h)}{h} \ , \ \alpha_i^D = \frac{2}{h} \ , \ \theta_i^D = -\frac{(2x_{i-1} + h)}{h}$$

$$\alpha_i^E = \frac{2}{h} \quad , \quad \theta_i^E = -\frac{(2x_i + h)}{h} \ , \ \alpha_i^F = \frac{2}{h} \ , \ \theta_i^F = -\frac{(2x_{i+1} + h)}{h} \tag{15}$$

By using N cubic polynomial splines the weights are mapped directly to the output weights of the FFANN using cubic sigmoids for transfer functions. The resultant architecture for the cubic spline basis is shown in Fig. 10.

For both the piecewise linear and piecewise cubic transfer functions the values of the input and bias weights are fixed for a given number of knots in the problem domain and are effectively uncoupled from the determination of the output weights. The construction of the network has been reduced to a question of finding the appropriate values for coefficients w_i. The computation of the expansion coefficients can be made from a variety of numerical methods. We now apply a procedure stemming from the method of weighted residuals mentioned earlier.

DETERMINATION OF OUTPUT WEIGHTS - THE METHOD OF WEIGHTED RESIDUALS

If independent spatial variables are described by the vector $\tilde{\mathbf{s}}$ and $L(y)$ is some differential operator, then the substitution of an approximation of y into the governing differential equation $L(y) = g(\tilde{\mathbf{s}})$, results in the equation $L(y_a) - g(\tilde{\mathbf{s}}) = R(w_1, ..., w_N, \tilde{\mathbf{s}})$, where R is the equation residual.

If we specify that y_a satisfy the boundary conditions exactly, then the basis coefficients w_i that satisfy the differential equation can be determined by requiring that the weighted equation residual R be integrated over the problem domain Ω and set to zero,

$$\int_\Omega FR \, d\Omega = (F, R) = 0 \tag{16}$$

where $F(\tilde{\mathbf{s}})$ is the weight or test function and (F, R) is defined as the integral inner product of functions F and R. It is from Eq.(16) that the method of weighted residuals (MWR) derives its name. The function F must be made up of N linearly independent functions F_k $(k = 1, ..., N)$ to generate the N algebraic equations needed to solve for the basis coefficients of Eq.(16). The system of algebraic equations is linear if the differential operator is linear. Otherwise the equations may be nonlinear though usually still quite tractable. In either case, these equations can be evaluated for a unique solution through standard techniques [21], allowing the evaluation of both linear and nonlinear differential equations assuming the problem is well-posed.

Different choices to the weight function F give rise to different computational methods that are subclasses of MWR. By approaching the evaluation of the output weights through the method of weighted residuals, we have access to both theoretical and computational results from the most commonly used techniques in the fields of scientific and engineering computing. The various subclasses of the method of weighted residuals are discussed and compared at great length by Finlayson [9] and Fletcher [22]. For this paper, we will use the conventional Galerkin method (p.30 [22])

$$F_k(\tilde{\mathbf{s}}) = \Phi_k(\tilde{\mathbf{s}}), \quad \text{for} \quad k = 1, 2, ..., N \tag{17}$$

where $\Phi_k(\tilde{\mathbf{s}})$ is the same type of interpolation function used by y_a. In our case, $\Phi_k(\tilde{\mathbf{s}})$ is either the piecewise linear or piecewise cubic spline. Our use of the Galerkin method is motivated following the observation made by Fletcher that " . . . the Galerkin method produces results of consistently high accuracy and has a breadth of application as wide as any method of weighted residuals" (p.38 [22]).

NUMERICAL EXAMPLE

Reference [23] presents a problem in which it is necessary to increase the pressure of air in a closed vessel in a specific period of time. As illustrated in Fig. 11, the problem involves a pump delivering hydraulic fluid through a long pipe to a vessel containing air. The governing equations for the system are:

- piping pressure drop

$$\Delta p = p_1 - p_2 = 15 \, \frac{dm}{dt}$$

where p_1 is the pump outlet pressure (kPa), p_2 is the pressure inside the vessel (kPa), and the mass flow rate dm/dt is measured in kg/s.

- vessel pressure

$$p_2 = 100 + 0.5m$$

- pumping power

$$P = \frac{d\gamma}{dt}(p_1 - 100)$$

where $d\gamma/dt$ = ideal pump flow rate, (kg/s)

- back leakage in the pump

The resultant flow through the pipe becomes

$$\frac{dm}{dt} = \frac{d\gamma}{dt} - 0.2(p_1 - 100)$$

The initial pressure of the air in the vessel is 100 kPa and it is required to increase it to 190 kPa within 120 seconds. In order to increase the pressure it is necessary to increase the mass of hydraulic fluid in the vessel, m, from zero to 180 kg within the time previously specified.

As a numerical example, we wish to construct a single input - multiple output FFANN that approximates the temporal variation in mass and mass flow rate for the pump such that the power consumed during the process is minimized. The constructed FFANN will utilize both piecewise linear and cubic transfer functions.

The integral to be minimized in this problem is the total pumping power

$$I = \int_o^t \frac{d\gamma}{dt}(p_1 - 100)dt$$

Acknowledging the time constraint, and after substituting $d\gamma/dt$ and p_1 as a function of m and dm/dt, the integral becomes

$$I = \int_0^{120} \left[60 \left(\frac{dm}{dt} \right)^2 + 3.5m\frac{dm}{dt} + 0.05m^2 \right] dt \tag{18}$$

Subjecting Eq.(18) to the Euler-Lagrange equation yields:

$$\frac{d^2m}{dt^2} - \beta m = 0 \quad \text{for} \quad 0 \leq t \leq 120 \quad \text{where} \quad \beta = \frac{1}{1200} \tag{19}$$

with associated boundary conditions

$$m = 0 \quad \text{at} \quad t = 0 \qquad \text{and} \qquad m = 180 \quad \text{at} \quad t = 120$$

This is the differential equation whose solution must be approximated by the FFANN in order to obtain the functions m and dm/dt that minimize the total power consumed by the pump. Since the temporal range is limited, Eq.(19) is a two-point boundary value problem and the FFANN will have constant output weights.

To measure the accuracy of the constructed FFANN we will compare its output to the exact solution of the differential equation for this problem:

$$m = 11.28 \sinh\left(t\sqrt{\beta}\right) \quad \text{and} \quad \frac{dm}{dt} = 0.3257 \cosh\left(t\sqrt{\beta}\right) \tag{20}$$

FFANN Construction Using Piecewise Linear Transfer Functions

The problem domain is reduced by a change in the independent variable

$$\tau = \frac{t}{t_{max}} \quad \text{where} \quad t_{max} = 120$$

so that Eq.(19) becomes

$$\frac{d^2m}{d\tau^2} = C_1 m \quad \text{for} \quad 0 \le \tau \le 1 \quad \text{where} \quad C_1 = \beta t_{max}^2 \tag{21}$$

To use the piecewise linear transfer functions, the second-order ordinary differential equation must be reduced to first-order. This is accomplished by a change in the dependent variable so that Eq.(21) reduces to a set of two first-order differential equations

$$\frac{dp}{d\tau} = C_1 m \quad \text{and} \quad \frac{dm}{d\tau} = p \tag{22}$$

Applying the method of weighted residuals to Eq.(22), the weak form is obtained,

$$\left(F_k, \frac{dp}{d\tau}\right) = C_1\left(F_k, m\right) \quad \text{and} \quad \left(F_k, \frac{dm}{d\tau}\right) = \left(F_k, p\right) \quad \text{for} \quad k = 1, 2, \ldots, N \tag{23}$$

The problem domain for the acting independent variable τ is discretized into N knots

$$\tau_0 < \tau_1 = 0 < \tau_2 < \ldots < \tau_N = 1 < \tau_{N+1}$$

where τ_0 and τ_{N+1} are the two required auxiliary knots outside the problem domain. Using the input and bias weights calculated from Eq.(24), the piecewise linear transfer functions of the FFANN are constrained to form hat functions centered at each knot in the problem domain.

$$\begin{aligned}
\alpha_i^A &= \frac{2}{(\tau_i - \tau_{i-1})} \quad , \quad \theta_i^A = -\frac{2\tau_{i-1}}{(\tau_i - \tau_{i-1})} - 1 \\
\alpha_i^B &= \frac{2}{(\tau_{i+1} - \tau_i)} \quad , \quad \theta_i^B = -\frac{2\tau_i}{(\tau_{i+1} - \tau_i)} - 1
\end{aligned} \tag{24}$$

where $i = 1, 2, \ldots, N$.

The dependent variables are approximated by the basis expansions

$$m_a = \sum_{i=1}^{N} \Phi_i w1_i \quad \text{and} \quad p_a = \sum_{i=1}^{N} \Phi_i w2_i \tag{25}$$

Substitution of the approximations and their derivatives and application of the conventional Galerkin technique to Eq.(23) results in

$$\begin{aligned}
C_1 \sum_{i=1}^{N} (F_k, \Phi_i)\, w1_i - \sum_{i=1}^{N} \left(F_k, \frac{d\Phi_i}{d\tau}\right) w2_i = 0 \\
\sum_{i=1}^{N} \left(F_k, \frac{d\Phi_i}{d\tau}\right) w1_i - \sum_{i=1}^{N} (F_k, \Phi_i)\, w2_i = 0
\end{aligned} \tag{26}$$

We define the following matrices,

$$M1_{ki} = \left(\Phi_k, \frac{d\Phi_i}{d\tau}\right) = \int_0^1 \Phi_k \frac{d\Phi_i}{d\tau} d\tau \quad \text{and} \quad M2_{ki} = (\Phi_k, \Phi_i) = \int_0^1 \Phi_k \Phi_i \, d\tau \tag{27}$$

The nature of the hat functions as bases reveal that matrices **M1** and **M2** are positive definite and tridiagonal and therefore of order N in storage, since they can be stored as three vectors. After imposing boundary conditions Eq.(26) can be rewritten into a single system of algebraic equations

$$\sum_{j=1}^{2N} M3_{qj} w3_j = b_q \quad \text{for} \quad q = 1, \ldots, 2N \tag{28}$$

where

$$w3_j = \begin{bmatrix} w1_i \\ w2_i \end{bmatrix} \quad \text{and} \quad M3_{qj} = \begin{bmatrix} C_1 M2_{ki} & -M1_{ki} \\ M1_{ki} & -M2_{ki} \end{bmatrix}$$

but

$$\begin{aligned}
M3_{11} &= 1 \quad \text{and} \quad M3_{1j} = 0 \quad \text{for} \quad j = 2, \ldots, 2N \\
M3_{NN} &= 1 \quad \text{and} \quad M3_{Nj} = 0 \quad \text{for} \quad j = 1, \ldots, N-1 \quad \text{and} \quad j = N+1, \ldots, 2N
\end{aligned}$$

and

$$\begin{aligned}
b_1 &= m(0) = m_a(0) = \sum_{i=1}^{N} \Phi_i(0) w1_i = w1_1 = 0 \\
b_N &= m(1) = m_a(1) = \sum_{i=1}^{N} \Phi_i(1) w1_i = w1_N = 180 \\
\text{with} \quad b_q &= 0, \quad q = 2, \ldots, N-1 \quad \text{and} \quad b_q = 0, \quad q = N+1, \ldots, 2N
\end{aligned}$$

Post-Processing of Solution

The set of linear algebraic equations in Eq.(28) can be solved by any of the standard linear equation solvers. Since the **M3** is tridiagonal, we resort to the computationally efficient generalized Thomas algorithm [24] which requires only order N operations. With the values for $w3_i$ known, the values for $w1_i$ and $w2_i$ can be extracted and the expansions for m_a and dm_a/dt given in Eq.(25) are complete with the following modification

$$\frac{dm_a}{dt} = \frac{d\tau}{dt}\frac{dm_a}{d\tau} = \frac{1}{t_{max}}\frac{dm_a}{d\tau} = \frac{p_a}{t_{max}} = \sum_{i=1}^{N} \Phi_i \frac{w2_i}{t_{max}} \tag{29}$$

By applying Eq.(8) and Eq.(24) to $w1_i$ and $w2_i/t_{max}$, we can construct a single input - multiple output feedforward network similar to that illustrated in Fig. 12. If the input weights α_i^A and α_i^B are divided by t_{max} then for a given value of $0 \le t \le 120$, the network will output approximations for m and dm/dt.

FFANN Construction Using Piecewise Cubic Transfer Functions

Since the piecewise cubic transfer function is nonzero up to the third derivative we can apply MWR to Eq.(21) directly without resorting to a second dependent variable.

$$\left(F_k, \frac{d^2m}{d\tau^2}\right) = C_1\left(F_k, m\right)$$

We again introduce the approximate solution m_a, with the problem domain for the acting independent variable τ again discretized into N knots,

$$\tau_1 < \tau_2 = 0 < \tau_3 < \ldots < \tau_{N-1} = 1 < \tau_N$$

where τ_1 and τ_N are located outside the problem domain. Again, by using the input and bias weights, the cubic splines are constrained to form functions centered at each knot. The dependent variable m and it derivatives can be approximated with the basis expansions:

$$m_a = \sum_{i=1}^{N} \Phi_i \; w_i \;, \quad \frac{dm_a}{d\tau} = \sum_{i=1}^{N} \frac{d\Phi_i}{d\tau} w_i \;, \quad \text{and} \quad \frac{d^2 m_a}{d\tau^2} = \sum_{i=1}^{N} \frac{d^2\Phi_i}{d\tau^2} w_i$$

Substitution of the bases expansions and their derivatives into the weak formulation, with application of the conventional Galerkin technique, results in

$$\sum_{i=1}^{N} \left(\Phi_k, \frac{d^2\Phi_i}{d\tau^2} \right) w_i = C_1 \sum_{i=1}^{N} (\Phi_k, \Phi_i)\, w_i \quad \text{for} \quad k = 2, \ldots, N-1 \tag{30}$$

If we define the following matrices,

$$M1_{ki} = \left(\Phi_k, \frac{d^2\Phi_i}{d\tau^2} \right) = \int_0^1 \Phi_k \frac{d^2\Phi_i}{d\tau^2} d\tau \quad \text{and} \quad M2_{ki} = (\Phi_k, \Phi_i) = \int_0^1 \Phi_k \Phi_i \, d\tau$$

with

$$M3_{ki} = M1_{ki} - C_1 \; M2_{ki} \quad \text{and} \quad b_k \;\; = \;\; 0, \quad \text{for} \;\; k = 2, \ldots, N-1$$

then the algebraic system formed by Eq.(30) can be rewritten as

$$\sum_{i=1}^{N} M3_{ki} w_i = b_k \quad \text{for} \quad k = 2, \ldots, N-1 \tag{31}$$

Similar to the piecewise linear case, the nature of the piecewise cubic transfer function used as a basis function reveals that matrices **M1**, **M2**, and **M3** are positive definite and pentadiagonal and therefore of O(N) in storage.

Like the previous example, it is necessary to modify the coefficient matrix **M3** to properly impose the boundary conditions of the problem. Because the cubic spline extends over a total of five knots, it is necessary that the contribution of the bases centered one knot beyond the boundaries of the problem domain be included in the solution. This is implemented using what are typically termed phantom nodes or fictitious points [25]. Working with only the Dirichlet boundary conditions, it is necessary to apply the "not-a-knot" [26] boundary conditions to the assembled matrix **M3** and vector $\tilde{\mathbf{b}}$. The modification are as follows

$$
\begin{aligned}
M3_{11} &= \frac{3\Delta\tau_3}{\Delta\tau_2} \ , \quad M3_{12} = \left(K_1 - K_2 + \frac{3(\Delta\tau_2 + \Delta\tau_3)}{\Delta\tau_3}\right) \ , \\
M3_{13} &= \left(4K_1 - 3K_2 - \frac{3\Delta\tau_3}{\Delta\tau_2}\right) \ , \quad M3_{14} = \left(K_1 + 3K_2 - \frac{3(\Delta\tau_2 + \Delta\tau_3)}{\Delta\tau_3}\right) \ , \\
M3_{15} &= K_2 \quad \text{with} \quad M3_{1i} = 0 \quad \text{for} \quad i = 6, \ldots, N \quad \text{and} \quad b_1 = 6K_1 \ m(0)
\end{aligned}
$$

$$
\begin{aligned}
M3_{NN-4} &= K_4 \ , \quad M3_{NN-3} = \left(K_3 + 3K_4 - \frac{3(\Delta\tau_{N-3} + \Delta\tau_{N-2})}{\Delta\tau_{N-2}}\right) \ , \\
M3_{NN-2} &= \left(4K_3 - 3K_4 - \frac{3\Delta\tau_{N-3}}{\Delta\tau_{N-2}}\right) \ , \\
M3_{NN-1} &= \left(K_3 - K_4 + \frac{3(\Delta\tau_{N-3} + \Delta\tau_{N-2})}{\Delta\tau_{N-2}}\right) \ , \quad M3_{NN} = \frac{3\Delta\tau_{N-3}}{\Delta\tau_{N-2}} \\
\text{with} \ \ M3_{Ni} &= 0 \quad \text{for} \quad i = 1, \ldots, N-5 \quad \text{and} \quad b_N = 6K_3 \ m(1)
\end{aligned}
$$

where

$$
\begin{aligned}
\Delta\tau_i &= \tau_{i+1} - \tau_i \ , \quad K_1 = \frac{(3\Delta\tau_2 + 2\Delta\tau_3)\,\Delta\tau_3}{(\Delta\tau_2 + \Delta\tau_3)\Delta\tau_2} \ , \quad K_2 = \frac{\Delta\tau_2^2}{(\Delta\tau_2 + \Delta\tau_3)\Delta\tau_3} \\
K_3 &= \frac{(2\Delta\tau_{N-3} + 3\Delta\tau_{N-2})\,\Delta\tau_{N-3}}{(\Delta\tau_{N-2} + \Delta\tau_{N-3})\Delta\tau_{N-2}} \ , \quad K_4 = \frac{\Delta\tau_{N-2}^2}{(\Delta\tau_{N-2} + \Delta\tau_{N-3})\Delta\tau_{N-3}}
\end{aligned}
$$

The final matrix **M3** with imposed boundary conditions is pentadiagonal and the unknown vector $\tilde{\mathbf{w}}$ can be solved for using the same generalized Thomas algorithm introduced in the piecewise linear case. For the pentadiagonal matrix the algorithm requires only order N operations.

For a uniform grid $\Delta\tau_i = h$ and with the values of w_i and t_{max} known, we can construct a single input - single output FFANN for m_a by application of Eq.(15) and division of the input weights by t_{max}. The values of dm_a/dt may be found by differentiating the cubic sigmoids of the network with respect to τ and application of the chain rule.

Since sections of the splines centered at knots τ_1, τ_2, τ_{N-1}, and τ_N lie outside the problem domain, these splines require fewer transfer functions to describe them. So, for this example problem

$$
\begin{aligned}
\Phi_1 &= 2\left(1-\Upsilon_1^B\right) - 0.5\left(\Upsilon_1^E - \Upsilon_1^F\right) \ , \\
\Phi_2 &= 2\left(\Upsilon_2^A-\Upsilon_2^B\right) - 0.5\left(1 - \Upsilon_2^D\right) - 0.5\left(\Upsilon_2^E - \Upsilon_2^F\right) \ , \\
\Phi_{N-1} &= 2\left(\Upsilon_{N-1}^A-\Upsilon_{N-1}^B\right) - 0.5\left(\Upsilon_{N-1}^C - \Upsilon_{N-1}^D\right) - 0.5\left(\Upsilon_{N-1}^E - 1\right) \ , \\
\Phi_N &= 2\left(\Upsilon_N^A-1\right) - 0.5\left(\Upsilon_N^C - \Upsilon_N^D\right)
\end{aligned}
$$

As a result the total number of required transfer functions are $T = 6N - 8$.

RESULTS

Piecewise Linear Transfer Functions

A single input - double output FFANN (Fig. 12) was constructed using a single hidden layer of 512 processing elements. This corresponds to an even spacing, within the problem domain, of 256 knots. The input, bias, and output weights for the network were computed via a conventional FORTRAN77 code. All computations were done in double precision on a Sun SPARCstation 10 with a run-time on the order of 1 second. Figures 13 and 14 compare the numerical output generated by the network (triangles) plotted against the exact solution (solid line) of Eq.(20) with root mean square errors of 1.79×10^{-3} and 7.63×10^{-5}, respectively.

From the bounding term on the error in the conventional Galerkin method, we are led to expect that the discrete L_2 norm of the error in m and dm/dt should be bounded by a quadratic term based on the spacing of the knots [27]. Using a uniform grid spacing

$$(\tau_{i+1} - \tau_i) = (\tau_i - \tau_{i-1}) = h = \left(\frac{1}{N-1}\right)$$

So, for an error of $E = m - m_a$, and with the discrete L_2 norm defined as

$$||E|| = \sqrt{h\left[\sum_{i=1}^{N}(m(\tau_i) - m_a(\tau_i))^2\right]} \quad \text{we have} \quad ||E|| \leq Ch^2 \text{ where } C \text{ is a constant.}$$

The convergence plot is shown in Fig. 15 for the calculation of m (circles), while the plot for dm/dt (triangles) appears nearly identical. The size of the mesh was halved starting at $h = 0.25$ to a value of $h = 3.91 \times 10^{-3}$. For both m and dm/dt the line is found to have a slope of about 2, showing the expected convergence.

Piecewise Cubic Sigmoids

Figures 13 and 14 compare the output (circles) of a single input - single output FFANN, similar to Fig. 1, with the exact solution for m and dm/dt at 10 equispaced knots (RMS $= 3.37\times10^{-4}$ and 4.01×10^{-4} respectively).

From the bounding term on the error in the conventional Galerkin method, we are led to expect that the discrete L_2 norm of the error should be bounded by a quartic term based on the spacing of the knots [27]. Figure 16 illustrates the convergence for m and dm/dt. Like the piecewise linear case the mesh spacing was halved, starting at $h = 0.25$. The convergence slope is approximately equal to 4, showing the expected convergence.

CONCLUSION

The engineer, as an FFANN user, is usually capable of casting basic knowledge of a physical system in the form of algebraic and/or differential model equations. Training FFANNs with experimental observations using arbitrary initial connection weight configurations when theoretical models are available can be time consuming, without the guarantee of convergence. It would be more convenient and efficient to incorporate mathematical models directly into the FFANN architecture. The technique presented in this paper is such a method.

By theory and numerical example, we have shown that it is possible to construct FFANNs with piecewise linear and piecewise cubic functions to model the solution of a calculus of variations problem without the need for training in the conventional connectionist sense. The construction requires imposing certain constraints on the values of the input, bias, and output weights. The net effect of placing these constraints on the weights, in the scheme presented here, is to transform the FFANN into a representation of a basis expansion and thus allows conventional numerical techniques to be employed in the analysis and synthesis of the neural network. The transformation renders the evaluation of the FFANN weights indistinguishable from the solution of problem variables in a computational mechanics algorithm. Hence, all theoretical and practical findings regarding convergence, optimality, and accuracy apply to the transformed network as well. Also, once the available information about a system of interest is incorporated, supervised learning can be used to correct and refine the possibly inadequate initial theoretical model. An additional and important benefit of an equation embedding approach is that an analog simulation of the physical system of interest, or arbitrary ordinary differential equations, can be constructed in hardware for rapid real-time computation.

Though the algorithms used in this paper were not conventionally connectionist in nature, it was the authors' intent to show that connectionist systems are amenable to analysis and that they can be reduced to well-known numerical analysis techniques by applying assumptions which do not change the systems' fundamental architecture. The numerical methods used are all linear ($O(N)$) in storage and processing time. The L_2 norm of the network approximation error decreases quadratically with the increasing number of hidden layer neurons when using the piecewise linear transfer function, and quartically when using the piecewise cubic sigmoid.

It may seem that the programming approach is severely limited by the use of the piecewise linear and piecewise cubic transfer functions. However, for applications in which speed of real-time computation is important, the piecewise polynomial functions are among the more computationally efficient functions to use in hardware. The authors can determine no reason why other transfer functions reported in the literature, such as hyperbolic tangents, exponentially based sigmoids, and radial basis functions could not be used.

The accuracy of this implementation was demonstrated for a linear one-dimensional example in this paper, however the validity of this approach with respect to first and second order nonlinear ordinary differential equations has previously been demonstrated by the first author when using piecewise linear transfer functions [20]. We would like to extend the equation embedding method for FFANNs to two and more dimensions using the piecewise linear and cubic splines.

As the quality of the basis function affects the interpolation, storage, speed, and convergence properties of the equation embedding method, work is progressing on generating higher order splines using such functions as the hyperbolic tangent, sigmoid and radial basis function. The higher order splines to be investigated include extensions of the Lagrange, Hermite, and B-splines.

ACKNOWLEDGEMENTS

The authors would like to thank Dr. Robert Shelton of NASA Johnson Space Center for helpful discussions. This work is supported under NASA grant NAG 1-1433.

REFERENCES

[1] S. Omohundro, "Efficient Algorithms with Neural Network Behaviour," *Complex Systems* **1**, 237 (1987).

[2] T. Poggio and F. Girosi, "A Theory for Networks for Approximation and Learning," A.I. Memo No. 1140, *Artificial Intelligence Laboratory, Massachusetts Institute of Technology*, 1989.

[3] F. Girosi and T. Poggio, "Networks and the Best Approximation Property," Biological Cybernetics, **69**, 169-176 (1990).

[4] E. Barnard and D. Casasent, "New Optical Neural System Architectures and Applications," *Optical Computing 88*, SPIE Proceedings Series, **963**, 1988, pp. 537-545.

[5] H. Lee and I. Kang, "Neural Algorithms for Solving Differential Equations," *Journal of Computational Physics* **91**, 110-131 (1990).

[6] L. Wang and J. M. Mendel, "Structured Trainable Networks for Matrix Algebra," *IEEE International Joint Conference on Neural Networks*, Part 2 (of 3), San Diego, 1990, pp. 125-132.

[7] J. Wang, "Electronic Realisation of Recurrent Neural Network For Solving Simultaneous Linear Equations," *Electronics Letters* **28**(5), 493-495 (1992).

[8] M. W. M. G. Dissanayake and N. Phan-Thien, "Neural-Network-Based Approximations for Solving Partial Differential Equations," *Communications In Numerical Methods In Engineering* **10**, 195-201 (1994).

[9] B. A. Finlayson, *The Method of Weighted Residuals and Variational Principles*, Academic Press, New York, 1972.

[10] R. Weinstock, *Calculus of Variations*, Dover Publications, New York, pp. 20-24, 1974.

[11] G. Cybenko, "Approximation by Superposition of a Sigmoidal Function," *Mathematics of Control Signals and Systems* **2**, 303-314 (1989).

[12] W. H. Beyer, *CRC Standard Mathematical Tables, 26th Edition*, CRC Press, p. 219, 1981.

[13] A. Maren, C. Harston and R. Pap, *Handbook of Neural Computing Applications*, Academic Press, New York, p. 236, 1990.

[14] K. Atkinson, *An Introduction to Numerical Analysis*, Wiley, New York, pp. 228-230, 1989.

[15] L. O. Chua, C. A. Desoer and E. S. Kuh, *Linear and Nonlinear Circuits*, McGraw Hill, New York, pp. 76-83, pp. 171-212, 1987.

[16] S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach* , Prentice Hall, New Jersey, p. 584, 1994.

[17] Y. Ito, "Approximation of Functions on a Compact Set by Finite Sums of a Sigmoid Function Without Scaling," *Neural Networks* **4**, 817-826 (1991).

[18] P. M. Prenter, *Splines and Variational Methods*, Wiley, New York, pp. 28-35, pp. 68-76, 1989.

[19] A. J. Meade, Jr. and A. A. Fernandez, "The Numerical Solution of Linear Ordinary Differential Equations by Feedforward Neural Networks," *Mathematical and Computer Modelling* **19**(12), 1-25 (1994).

[20] A. J. Meade, Jr. and A. A. Fernandez, "Solution of Nonlinear Ordinary Differential Equations by Feedforward Neural Networks," *Mathematical and Computer Modelling* **20**(9), 19-44 (1994).

[21] G. Strang, *Linear Algebra and Its Applications*, Second Edition, Academic Press, New York, pp. 48-99, 1980.

[22] C. A. J. Fletcher, *Computational Galerkin Methods*, Springer-Verlag, New York, 1984.

[23] W. F. Stoecker, *Design of Thermal Systems*, McGraw-Hill Book Company, New York, pp. 472-478, 1989.

[24] C. A. J. Fletcher, *Computational Techniques for Fluid Dynamics*, **1**, Springer-Verlag, New York, pp. 183-188, 1988.

[25] D. A. Anderson, J. C. Tannehill and R. H. Pletcher, *Computational Fluid Mechanics and Heat Transfer*, Hemisphere Publishing Corporation, New York, p. 461, 1984.

[26] C. de Boor, *A Practical Guide to Splines*, Springer-Verlag, New York, pp. 55-56, 1978.

[27] C. Johnson, *Numerical Solution of Partial Differential Equations by The Finite Element Method*, Cambridge University Press, Cambridge, p. 90, 1990.

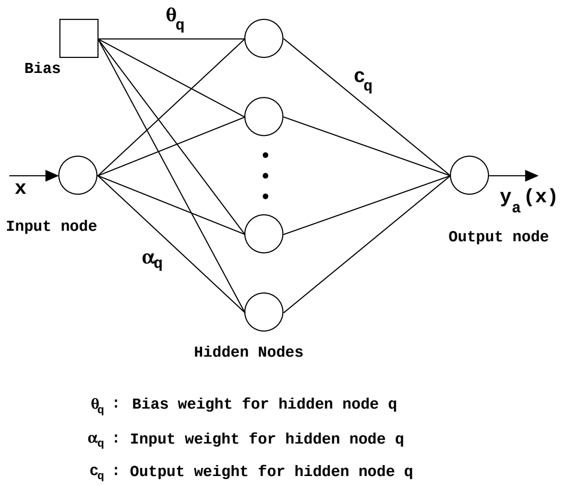

Figure 1: Simple feedforward neural network with one input and one output node.

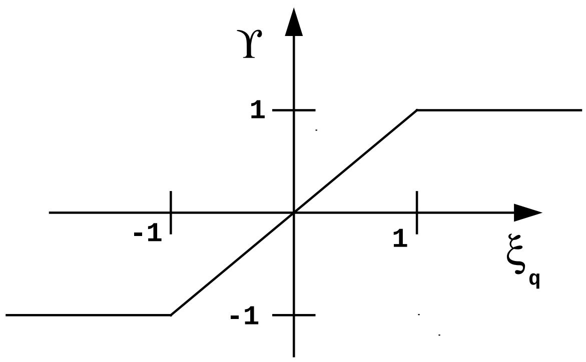

Figure 2: Hard limit transfer function

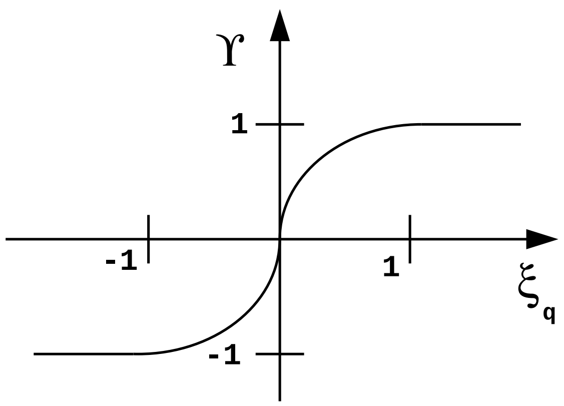

Figure 3: Cubic sigmoid transfer function

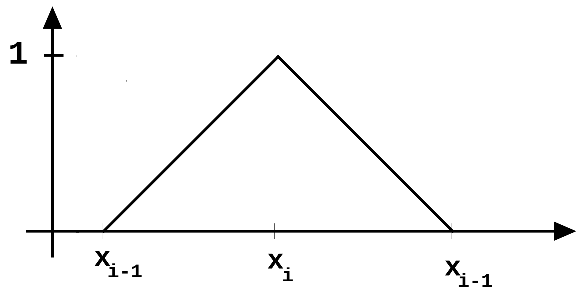

Figure 4: The Chapeau or "hat" function.

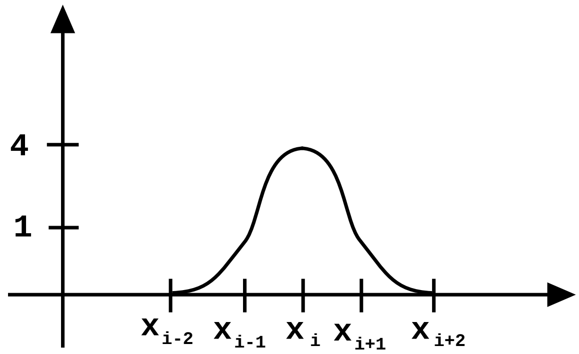

Figure 5: The cubic polynomial spline.

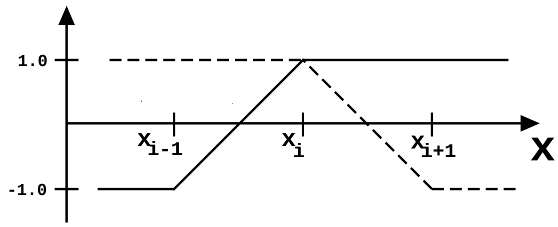

Figure 6: Right side transfer function (dashed) rotated by change of output weight sign. Addition of the two piecewise linear transfer functions creates a hat function. Refer to Figure 3.

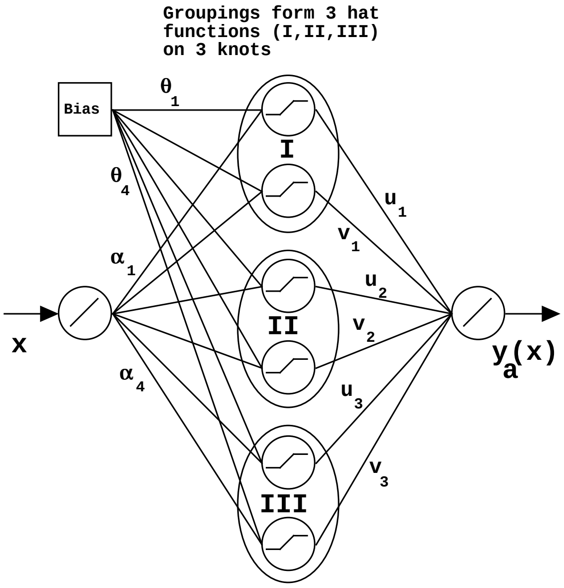

Figure 7: Single input, single output FFANN using hard-limit transfer functions to create hat function bases for solution approximation.

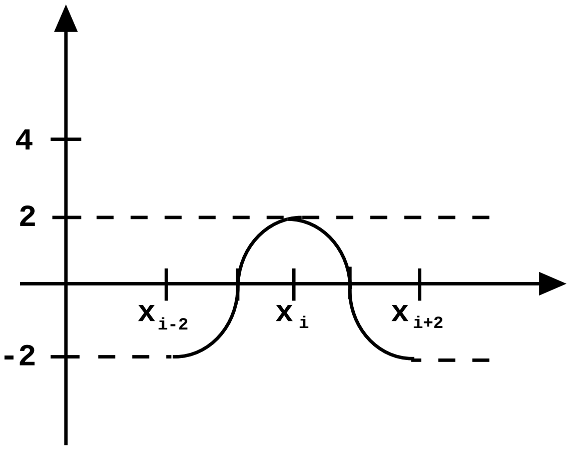

Figure 8: Adding two large opposing cubic sigmoids to form "bell."

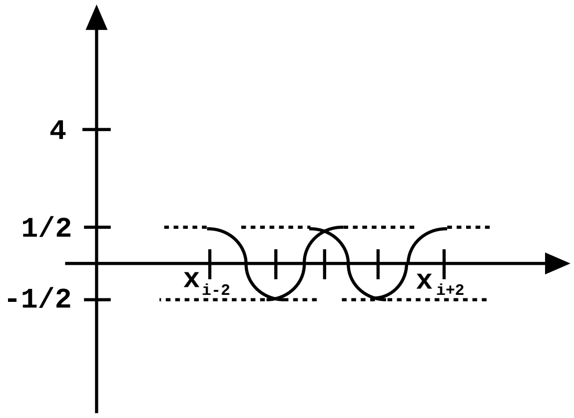

Figure 9: Construction of smaller "bell" distributions by two pairs of opposed cubic sigmoids.

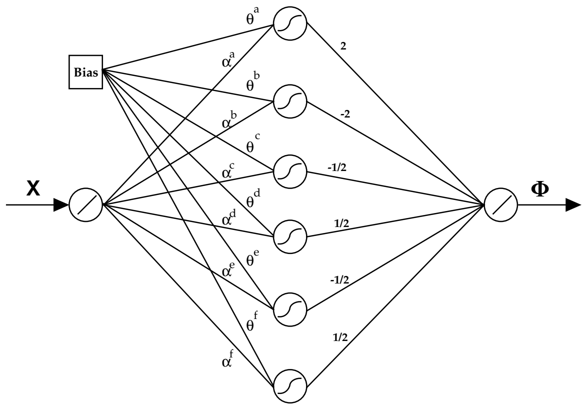

Figure 10: Single input, single output FFANN using cubic sigmoid transfer functions to form a cubic spline.

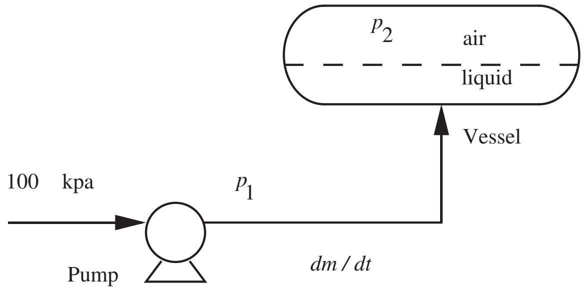

Figure 11: Example problem.

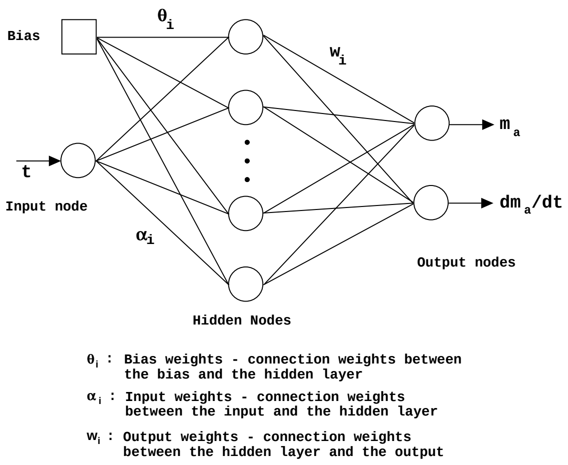

Figure 12: FFANN for example problem using piecewise linear transfer functions.

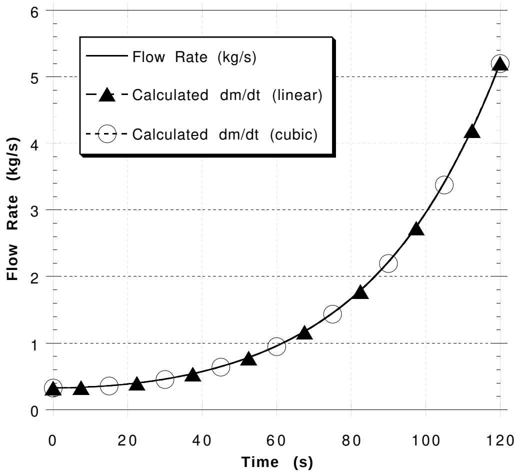

Figure 14: Comparison of exact solution and network output for mass flow rate. Linear: T = 512 with RMS Error = 7.63×10^{-5}. Cubic: T = 52 with RMS Error = 4.01×10^{-4}.

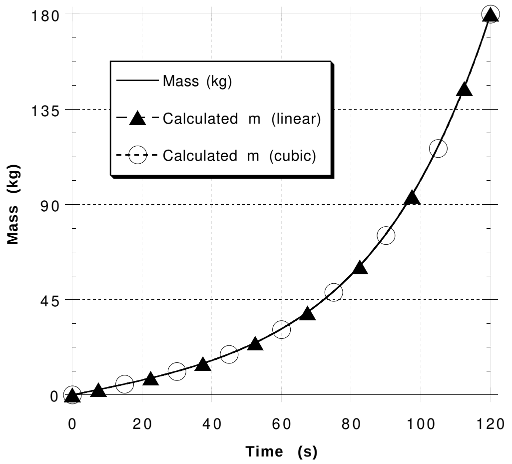

Figure 13: Comparison of exact solution and network output for mass. Linear: T = 512 with RMS Error = 1.79 $\times10^{-3}$. Cubic: T = 52 with RMS Error = 3.37×10^{-4}.

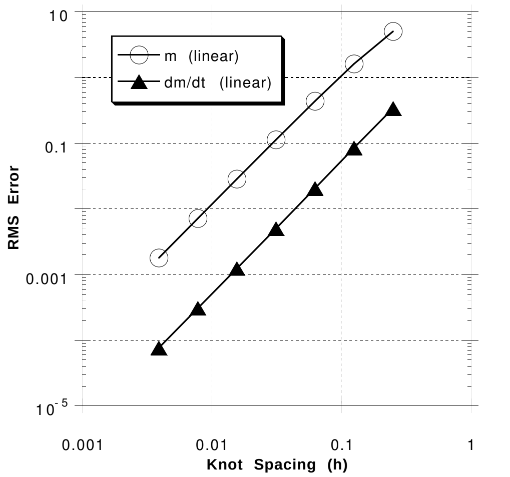

Figure 15: Logarithmic convergence plot for m and dm/dt using linear basis functions. avg. slope ≈ 2.

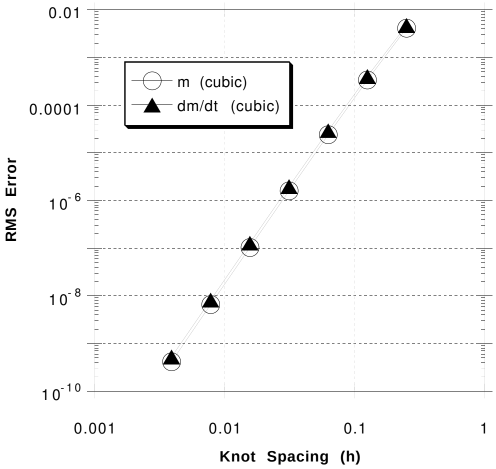

Figure 16: Logarithmic convergence plot for m and dm/dt using cubic spline basis functions. avg. slope ≈ 4.