19940208884

N94-25370

DEVELOPMENT OF PROGRAMMABLE ARTIFICIAL NEURAL NETWORKS

Final Report

NASA/ASEE Summer Faculty Fellowship Program–1993

Johnson Space Center

Prepared By:	Andrew J. Meade, Jr., Ph.D
Academic Rank:	Assistant Professor
University & Department	Rice University Department of Mechanical Engineering and Materials Science Houston, Texas 77251-1892
NASA/JSC	
Directorate:	Information Systems
Division:	Information Technology
Branch:	Software Technology
JSC Colleague:	Robert O. Shelton, Ph.D
Date Submitted:	September 30, 1993
Contract Number:	NGT-44-001-800
Approved By:	Robert O. Shelton
Date Approved:	10/14/93

DEVELOPMENT OF PROGRAMMABLE ARTIFICIAL NEURAL NETWORKS

Final Report

NASA/ASEE Summer Faculty Fellowship Program–1993

Johnson Space Center

Prepared By:	Andrew J. Meade, Jr., Ph.D
Academic Rank:	Assistant Professor
University & Department	Rice University Department of Mechanical Engineering and Materials Science Houston, Texas 77251-1892
NASA/JSC	
Directorate:	Information Systems
Division:	Information Technology
Branch:	Software Technology
JSC Colleague:	Robert O. Shelton, Ph.D
Date Submitted:	September 30, 1993
Contract Number:	NGT-44-001-800

ABSTRACT

Conventionally programmed digital computers can process numbers with great speed and precision, but do not easily recognize patterns or imprecise or contradictory data. Instead of being programmed in the conventional sense, artificial neural networks are capable of self-learning through exposure to repeated examples. However, the training of an ANN can be a time consuming and unpredictable process.

A general method is being developed by the author to mate the adaptability of the ANN with the speed and precision of the digital computer. This method has been successful in building feedforward networks that can approximate functions and their partial derivatives from examples in a single iteration. The general method also allows the formation of feedforward networks that can approximate the solution to nonlinear ordinary and partial differential equations to desired accuracy without the need of examples. It is believed that continued research will produce artificial neural networks that can be used with confidence in practical scientific computing and engineering applications.

INTRODUCTION

Neural networks have proven to be versatile tools for accomplishing what could be termed higher order tasks such as pattern recognition, classification, and visual processing. However, conventional wisdom has held that networks are unsuited for use in more purely computational tasks, such as mathematical modelling and physical analysis of engineering systems. Certainly the biological underpinnings of the neural network concept suggest that networks would perform best at tasks at which biological systems excel, and worse or not at all at other tasks.

Contrary to popular opinion the author believes that continued research into the approximation capabilities of networks will enable the neural network paradigm, with all of its advantages in behavior and adaptability, to be mated to the more purely computational paradigms of mathematically oriented scientific programming and analysis. Additionally, it is felt that the thorough investigation of network approximation capabilities will benefit the network field and connectionism in general.

In a field as conceptually difficult as the study of artificial neural networks, it is best to start investigation with supervised learning, test the established premises, and alter them to circumvent pitfalls in implimentation.

FUNCTION APPROXIMATION

Learning as Function Approximation

Central to the author's research approach is the view that supervised learning in artificial neural networks is equivalent to the problem of approximating a multivariate function and that learning should be able to be explained by approximation theory. Approximation theory deals with the problem of approximating or interpolating a multivariate function. This approach has been considered by other researchers in the field of ANNs [1]-[4]. However, the author extends this assumption of function approximation by assuming that ANNs can model discontinuous multivariate functions and should be at least as accurate and numerically efficient as existing computational techniques used in science and engineering. Also, ANN behavior and adaptation difficulties, from supervised learning to machine vision, should be amenable to the standard error analysis techniques used in numerical analysis [5].

Function Approximation In Engineering

There are three classes of tools used in science and engineering for the analysis of systems:

1. **analytical methods**, which include the formation of equations that model the behavior of systems and the analytic solution of those equations.

2. **computational methods**, which involve the simulation of system behavior by the numerical solution of the governing equations.

3. **experiments**, which involve the investigation of physical phenomena and the gathering of data to validate analytical models and numerical simulations.

In a general sense analytical and computational methods and experiments can be considered to be forms of function approximation. The governing equations derived from analytical methods are a compact representation of the functions that model some particular phenomena observed in experiments. Computational techniques are used to approximate the function or functions that satisfy the governing equations. The graphs and tables made from experiments are representations of the functions that underlie observed physical phenomena.

Computational Methods

In the wake of the computer revolution in scientific applications, a large number of computational techniques have emerged. Also, particular methods have assumed prominent positions in certain areas of application. For example, finite element methods are used almost exclusively for solving structural problems; spectral methods are becoming the preferred approach to global atmospheric modelling; and the use of finite difference methods is nearly universal in simulating fluid and thermal systems.

Each computational method has its own set of advantages and disadvantages depending on the characteristic of the application. These popular and apparently unrelated techniques are firmly entrenched in computer codes used every day by practicing scientists and engineers. Often the formal numerical training provided the scientist and engineer reinforces the divisions between the various computational methods available. However, Fletcher [6] has demonstrated that each of these numerical methods are in fact particular aspects of a more general approach known as the **method of weighted residuals** [7].

PROGRAMMABLE ARTIFICIAL NEURAL NETWORKS

It is the objective of the author's research program to demonstrate that artificial neural network behavior from supervised learning to machine vision can be derived from the method of weighted residuals. This would link ANNs with the relatively mature and established field of computational mechanics, extend ANN capabilities, and help in transforming ANN applications from an art to a science. This may also advance research in our understanding of biological neural systems.

If we are to assume that ANNs are as valid as established computational techniques, then ANNs should be evaluated in the same manner as are computational techniques.

The first step in evaluating the capabilities of a new numerical method, is to apply it to the solution of algebraic and ordinary and partial differential equations of known behavior. This same approach can be used for ANNs since the solution of algebraic and differential equations can be viewed as the approximation of a function that must satisfy the equation in question subjected to boundary and/or initial conditions.

Applying an ANN to the solution of an algebraic or differential equation effectively uncouples the influences of the quality of data samples, network architecture, and transfer functions from the network approximation performance. The solution of equations also allows us to study the influence of constraining the connection weights. The most immediate benefit in this approach would be the construction of networks that can approximate the solution to desired equations without the need for examples. This would be of value in

engineering applications since considerable effort may be saved if the equations governing a physical process can be directly incorporated into the neural network architecture without the need of examples, thereby shortening or even eliminating the learning phase.

This approach may also lead to the construction of network training routines that are faster and more accurate than those presently in use [8], [9]. In addition, progress made in this network programming approach should provide the research community insight into the working of networks for associative memory, classification, and machine vision applications.

Approach

The MWR approach has been taken by the author using the hard limit [10] as the transfer function. Interesting results have been produced and are presented to demonstrate the validity of the approach.

It can be argued that the supervised training of a feedforward network is a problem in function approximation using unconstrained optimization. In this sense, the task of the optimization scheme is to find the proper combination of connection weights between the processing elements, operating with specific transfer functions, so that the network minimizes the error between the network output and the desired output. Therefore, the training of a network possesses all of the problems one associates with unconstrained optimization such as avoiding local minima in search of the global minimum.

The most obvious remedy to this problem is to constrain the optimization while preserving the approximation capability of the network. Our task then is to form constraints between the weights so that the values of the weights may be determined with computational efficiency.

Constraint of Weights Between the Input and Hidden Layer

A univariate function $u(x)$ can be represented by a feedforward network (Fig. 1) with a single hidden layer, and a single input and output node using a linear transfer function, as follows:

$$u(x) = \sum_{i=1}^{N} \left(\Phi_i\left(\xi_i\right) + s_i\right) w_i, \quad \xi_i = \alpha_i x + \theta_i \quad \text{for} \quad i = 1, ..., N \tag{1}$$

Each hidden processing element is indexed by the subscript i, where N is the total number of hidden processing elements. The variable x is the value of the input and Φ_i represents the nonlinear transfer function for the i^{th} hidden processing element. The coefficients w_i, α_i and θ_i are the values of the connection weights between the hidden and output layers, the input and hidden layers, and the bias node and hidden layer, respectively. The role of the remaining set of coefficients, s_i, will be explained shortly. Equation (1) indicates that we must determine the value of $4N$ coefficients to approximate a function with the feedforward architecture. Our objective in constraining the weights is to decrease the number of unknown coefficients.

The formulation of Eq. (1) is similar to that given by Cybenko [11]. However, ξ_i will be modified in a manner similar to the radial basis technique. Notice that to provide extended dynamic range, this formulation assumes that the input and output processing elements

use linear transfer functions. However, this formulation also allows the use of nonlinear transfer functions in the output node.

We will constrain the weights θ_i in the following manner. Discretize the domain (Ω) of the input variable into $N-1$ intervals. Each interval is bracketed by the values $\hat{x}_i$ and $\hat{x}_{i+1}$ where $\hat{x}_i < \hat{x}_{i+1}$ and $i = 1, ..., N$. The value of N is equal to the total number of hidden processing elements. We will use the following equation to constrain θ_i:

$$\theta_i = -\alpha_i \hat{x}_i \quad \text{for} \quad i = 1, ..., N$$

so that

$$\xi_i = \alpha_i (x - \hat{x}_i) \quad \text{for} \quad i = 1, ..., N.$$

Notice that by constraining each bias weight (θ_i) in this manner, $\xi_i = 0$ for the i^{th} processing element when $x = \hat{x}_i$. The variables $\hat{x}_i$ are similar to the "centers" used in the radial basis function literature [12],[13].

For our analysis we will use a piecewise continuous polynomial approximation to the hyperbolic tangent (Eq. (2)) known as the hard limit which is illustrated in Fig. 2.

$$\Phi_i(\xi_i) = \xi_i - 1 \quad \text{where} \quad \xi_i = \frac{2(x - \hat{x}_i)}{\hat{x}_{i+1} - \hat{x}_i} \quad \text{for} \quad \hat{x}_i \leq x \leq \hat{x}_{i+1}$$

$$\Phi_i(\xi_i) = -1 \quad \text{for} \quad x \leq \hat{x}_i \quad \text{and} \quad \Phi_i(\xi_i) = +1 \quad \text{for} \quad \hat{x}_{i+1} \leq x. \tag{2}$$

Therefore, for the piecewise polynomial transfer function we can constrain the input weights α_i using Eq. (2).

$$\alpha_i = \frac{2}{\hat{x}_{i+1} - \hat{x}_i} \quad \text{for} \quad \hat{x}_i \leq x \leq \hat{x}_{i+1}$$

Notice then, that transfer functions from each processing element are distributed along the x axis in the domain of interest Ω. Each function is "centered" at the respective values of $\hat{x}_i$ (Fig. 3).

Equation (1) can now be seen as a weighted sum representation of the function $u(x)$. One can think of the transfer functions as being interpolation functions distributed along the transformed x axis. Each interpolation function, and its coefficient s_i, is multiplied by its respective weight, w_i, and summed to approximate a desired curve.

Constraint of Weights Between the Hidden and Output Layer

It can now be shown that the role of the coefficients s_i in Eq. (1) is to add or subtract constant values from the respective transfer function. This effectively moves the transfer function above or below the x axis (Fig. 4). Equation (1) may be rewritten as:

$$u(x) = \sum_{i=1}^{N} \Phi_i(\xi_i) w_i + \sigma \quad \text{where} \quad \sigma = \sum_{i=1}^{N} s_i w_i \tag{3}$$

The coefficient σ acts as the connection weight between the second bias and the output node (Fig. 1).

One final constraint we impose on the output weights in that $w_i = -w_{i+1}$. This requires that we use an even number of processing elements. The final constraint acts to convert the hard limits from global interpolation functions into local interpolation functions.

We have now constrained all of the parameters for the single input and single output feedforward network and have decreased the number of unknowns from $4N$ (α_i, θ_i, s_i, and w_i) to $N/2+1$ (w_i and σ). What remains now is to determine the output weights and the second bias weight so as to approximate not only the desired functional relationship, but also the derivatives of the function. This is done using the method of weighted residuals (MWR). More specifically, the Bubnov-Galerkin and Petrov-Galerkin methods.

ORDINARY DIFFERENTIAL EQUATIONS

Determination of Output Weights: Method of Weighted Residuals

To illustrate the method we will determine the output weights and bias (w_i and σ) needed to approximate the solution to a linear first order differential equation, using only the equation and the initial condition [14].

A feedforward network of one input and output node and a single hidden layer is constructed to approximate the solution to the following equation:

$$\frac{du}{dx} - u = 0 \tag{4}$$

with the boundary condition $u(0) = 0$. The exact solution is:

$$u_{\text{exact}} = e^x$$

Substituting Eq. (1) into Eq. (4) we have:

$$\frac{du}{dx} - u = \sum_{i=1}^{N} \frac{d\Phi_i(\xi_i)}{dx} w_i - \sum_{i=1}^{N} \Phi_i(\xi_i) w_i + \sigma = 0. \tag{5}$$

In this example we will set σ to zero. Notice that Eq. (5) is only satisfied if the nontrivial set w_i is exactly correct.

If we use random values for w_i the right hand side would be nonzero and act as a measure of the error. This error is also known as the equation residual (ϵ). We may obtain an acceptable approximation if we force the weighted residual to zero over the domain of interest, Ω.

$$\int_\Omega f_k(x)(\epsilon)\,dx = 0 = \int_\Omega f_k(x) \left(\sum_{i=1}^{N} \frac{d\Phi_i(\xi_i)}{dx} w_i - \Phi_i(\xi_i) w_i \right) dx \tag{6}$$

where $k = 1, \ldots, N$ and $f_k(x)$ is referred to as the weighting function or test function. The approach shown by Eq. (6) is known as the method of weighted residuals.

Since linearly independent relationships are needed to solve for the coefficients w_i, it is clear that f_k must be a set of linearly independent functions. The choice of the weighting functions correspond to different solution techniques such as the subdomain, collocation, least square, and the Galerkin methods [6]. For this investigation we will use a modification of the Galerkin approach or more specifically the Bubnov-Galerkin method [15]. This approach requires that f_k be chosen from the same family of functions as the transfer functions, that is,

$$f_k(x) = \Phi_k(\xi_k) \quad \text{for} \quad k = 1, ..., \ N. \tag{7}$$

Except for the change in the index from i to k, $\Phi_k(\xi_k)$ is described by the hat function.

So Eq. (6) may be rewritten as

$$\sum_{i=1}^{N} \int_{\hat{x}_1}^{\hat{x}_N} \Phi_k(\xi_k) \frac{d\Phi_i(\xi_i)}{dx} \, dx \, w_i - \sum_{i=1}^{N} \int_{\hat{x}_1}^{\hat{x}_N} \Phi_k(\xi_k)\Phi_i(\xi_i) \, dx \, w_i = 0 \quad \text{for} \quad k = 1, ..., N. \tag{8}$$

This forms the linear algebraic system of equations

$$\sum_{i=1}^{N} A_{ki} w_i = g_k \quad \text{for} \quad k = 1, ..., N. \tag{9}$$

In its present form A_{ki} of Eq. (9) is singular and must be modified by the initial condition. The initial condition may be written as

$$\sum_{i=1}^{N} \Phi_i(\hat{x}_1) \, w_i = g_1 = 1. \tag{10}$$

The weights w_i can then be evaluated directly from the solution of Eq. (9). Notice that the initial condition could have been partially satisfied by w_i and the value of σ would have been determined to satisfy the remainder (i.e. $g_1 = 0.5$ and $\sigma = 0.5$)

Comparison of the network approximation with the exact solution of Eq. (4) is shown in Fig. 5 for forty two hidden processing elements (twenty one output weights).

Example: Third Order Nonlinear Ordinary Differential Equation

As a further demonstration of the approximation capability of the network, a feedforward network of one input and output node, and a single hidden layer, was constructed to approximate the solution to the nonlinear third order ordinary differential equation known as the Blasius equation [17]. The Blasius equation is used to describe the steady and laminar two-dimensional flow of a viscous Newtonian fluid about a flat plate:

$$\frac{d^3 f}{d\eta^3} + f \frac{d^2 f}{d\eta^2} = 0 \tag{11}$$

with boundary conditions

$$f(0) = \frac{df}{d\eta}(0) = 0, \ \frac{df}{d\eta}(\eta \to \infty) = 1$$

The network approximation (Fig. 6), using fifty one output weights, is compared against a fifth order Runge-Kutta solver (finite difference) of variable step size that satisfied the boundary conditions with an absolute error value of 10^{-6}. Figure 7 illustrates the rate of convergence of the network using the $L2$ norm of the error and the interval spacing h.

PARTIAL DIFFERENTIAL EQUATIONS

The author has successfully programmed Higher Order Networks, also known as Sigma-Pi networks, to approximate the solution of a partial differential equation [16].

Example: Linear Elliptic Partial Differential Equation

A Sigma-Pi network of two inputs, one output node, and a single hidden layer has been constructed to approximate the solution to the linear elliptic equation that models fully developed steady flow of viscous Newtonian fluid through a duct of square cross-section:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + 1.0 = 0.0 \tag{12}$$

where u is the nondimensionalized velocity of the flow along the duct.

The domain (Ω) of the problem is: $-1.0 \leq x \leq 1.0$, $-1.0 \leq y \leq 1.0$

Boundary conditions: $u = 0$ along the perimeter Γ

Figures 8 and 9 show the surfaces made by the exact solution and the network solution of Eq. (12) for two thousand five hundred output weights. Figure 10 illustrates the rate of convergence of the network using the root mean square of the error and the interval spacing h.

REFERENCES

[1] Barron, A.R. and Barron, R.L., "Statistical learning networks: a unifying view," *Symposium on the Interface: Statistics and Computing Science,* Reston, Virginia, April 1988.

[2] Omohundro, S., "Efficient algorithms with neural network behaviour," *Complex Systems*, **1**, 237, 1987.

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

[4] Girosi, F. and Poggio, T., "Networks for learning: a view from the theory of approximation of functions," *Proceedings of the Genoa Summer School on neural networks and their applications*, Prentice Hall, 1989.

[5] Atkinson, K.E., *An Introduction to Numerical Analysis*, John Wiley & Sons, New York, 1989.

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

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

[8] Freeman, J.A. and Skapura, D.A., *Neural Networks: Algorithms, Applications, and Programming Techniques*, Addison-Wesley Publishing, 1991.

[9] Hecht-Nielsen, R., *Neurocomputing*, Addison-Wesley Publishing, 1990.

[10] Maren, A., Harston, C. and Pap, R., *Handbook of Neural Computing Applications*, 48, Addison-Wesley Publishing, 1990.

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

[12] Elanayar, S. and Shin, Y. C., "Approximation Capabilities of Radial Basis Function Neural Networks," *Intelligent Engineering Systems Through Artificial Neural Networks,* Vol. 2, pp. 291-298, ASME Press, New York, 1992.

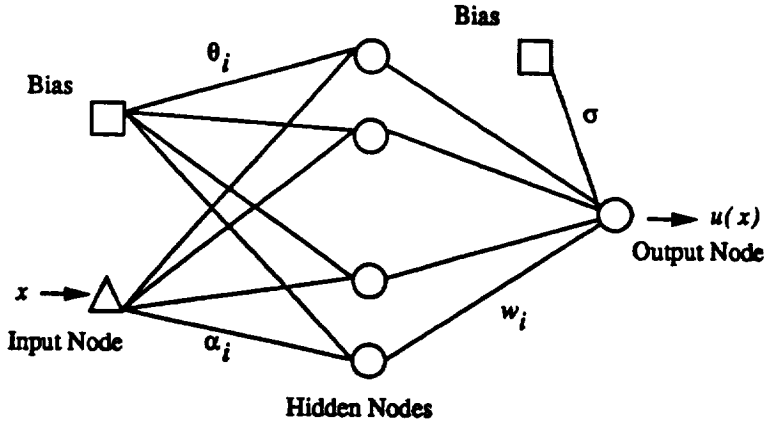
[13] Park, J. and Sandburg, I. W., "Universal Approximation Using Radial Basis Function Networks," *Neural Computation*, **3**, 246-257, 1991.

[14] "Solution of Ordinary Differential Equations by Programmable Feedforward Neural Networks," To be submitted, 1993.

[15] Mikhlin, S. G., *Variational Methods in Mathematical Physics*, Pergamon, Oxford, 1964.

[16] "Solution of Elliptic, Parabolic and Hyperbolic Partial Differential Equations by Programmable Feedforward Neural Networks," To be submitted, 1993.

[17] White, Frank M., *Viscous Fluid Flow*, McGraw-Hill, New York, 1974.



θ_i : Bias weight for the i^{th} hidden node.

α_i : Input weight for the i^{th} hidden node.

w_i : Output weight for the i^{th} hidden node.

σ : Bias weight for the output node.

Figure 1: Feedforward Network Architecture

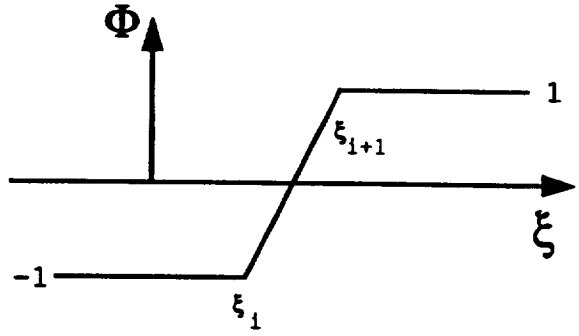


Figure 2: Piecewise Polynomial Transfer Function

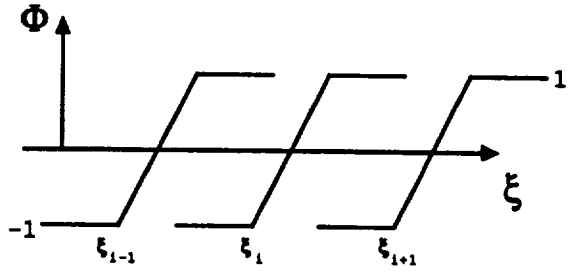


Figure 3: Distribution of Transfer Functions

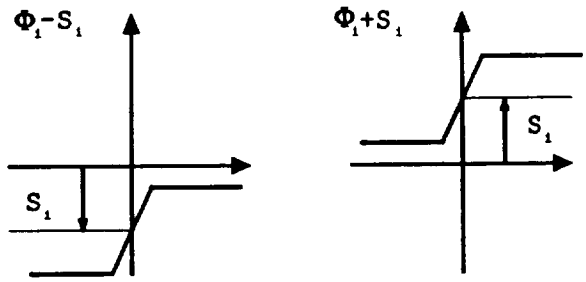


Figure 4: Transfer Function Offset

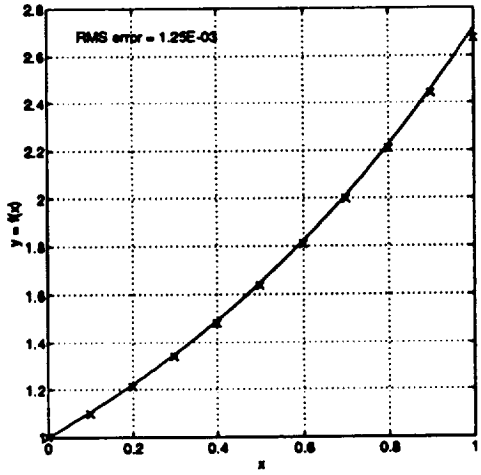


Figure 5: Comparison of Exact Solution and Network Approximation of Eq. (4) for 21 output weights

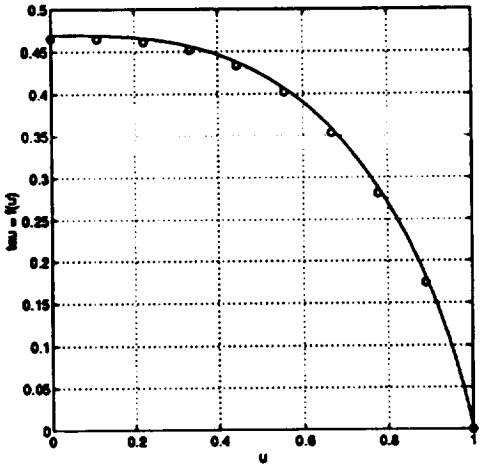


Figure 6: Comparison of Runge-Kutta Solution and Network Approximation of Eq. (11), for 51 output weights

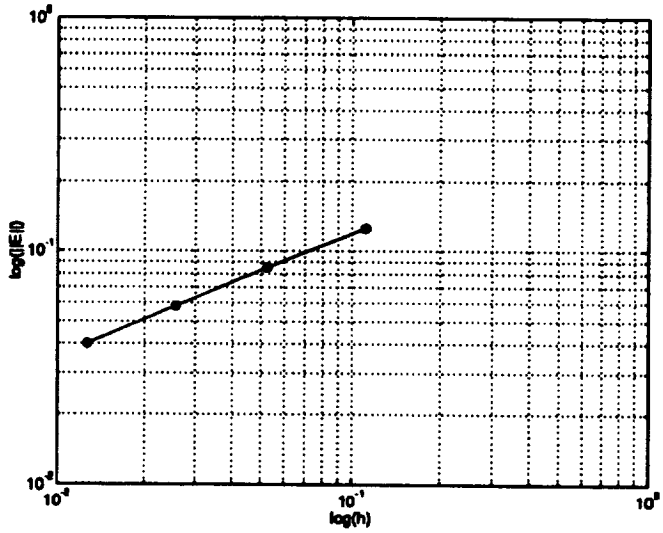


Figure 7: Convergence Rate of Network for Eq. (11)

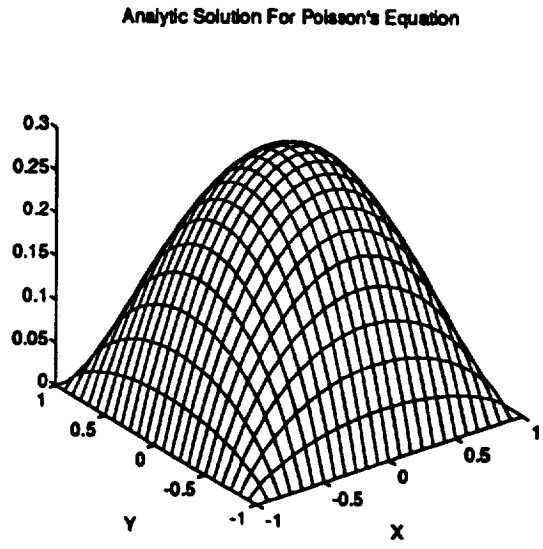


Figure 8: Analytic Solution of Eq. (12)

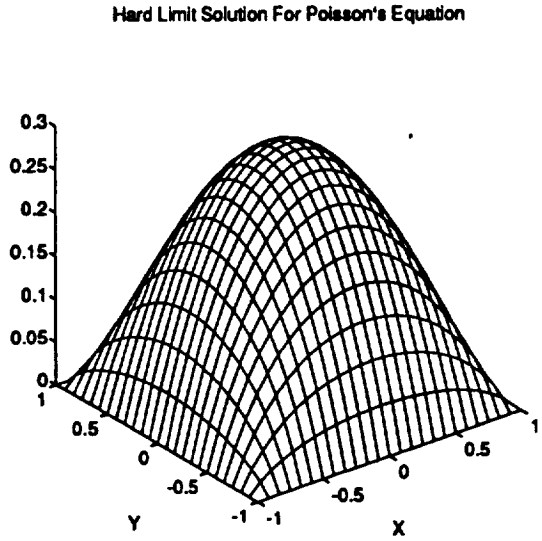


Figure 9: Network Solution of Eq. (12) for 2500 output weights

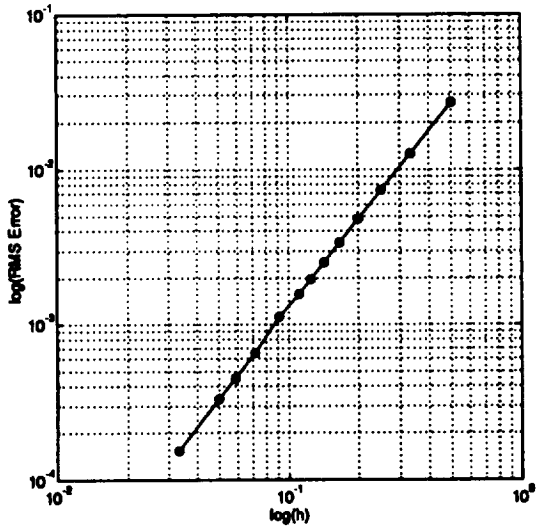


Figure 10: Convergence Rate of Network for Eq. (12)