1994

NASA/ASEE SUMMER FACULTY FELLOWSHIP PROGRAM

MARSHALL SPACE FLIGHT CENTER
THE UNIVERSITY OF ALABAMA

NEURAL NETWORKS APPLICATIONS TO CONTROL AND COMPUTATIONS

Prepared By: Dr. Leon A. Luxemburg

Academic Rank: Assistant Professor

Institution and Department: Texas A&M University

NASA/MSFC:

Laboratory: Structures and Dynamics
Division: Control Systems
Branch: Flight Dynamics

MSFC Colleagues: Richard Dabney
Henry Waites, Ph.D.

APPLICATIONS of NEURAL NETWORKS to FAST COMPUTATIONS

Our first problem is to use neural networks to develop a general high precision computational algorithm for feedforward neural networks and to show it efficiency by applying it to a solution of the inverse perspective transformation problem in the Automated Rendezvous and Capture Space program.

The inverse perspective problem for the Automated Rendezvous and Capture program is defined as the determination of the 6 degrees of freedom of the chase vehicle relative to the target vehicle. The solution involves the Newton-Raphson method which is rather cumbersome computationally. Therefore, a neural network approach was suggested by Richard Dabney [1]. He suggested that instead of solving a system of equations for each point, parameters of these equations can be considered as inputs to some neural network and the solution of these equations as an output. However, the major problem was the accuracy of the solution. While an RMS value was quite low, the worst case precision for a case of pitch angle remained unacceptably high- around 19.1%

We have used this problem to demonstrate a new approach to a fast, high precision neural computing which reduces the errors to a more acceptable level. This problem was successfully solved by introducing and training a different type of neural network. We have achieved a reduction of the worst case error to about 8.4% instead of 19.1% and the RMS error was also reduced by about 30%. In our approach we used a network with two layers with nodal sigmoid functions being hyperbolic tangent and in the output layer we used identity function $f(x)=x$ as a nodal function. This choice of architecture can realize any function of n variables as soon as we choose sufficiently many neurons in the two inner layers. This follows from the Kolmogorov's theorem on representation of function of several variables by using a superposition of functions of one variables and arithmetic operations.

One of the essential features of our approach is to use an adaptive learning rate to speed up learning and to minimize the worst case error instead of sum-squared error as is usually done. We used a standard backpropagation approach to learning with momentum and adjustable biases. This helped to overcome a number of conversion and precision problems encountered in the initial approach.

APPLICATIONS TO CONTROL PROBLEMS

Consider a linear plant

$$\dot{x} = Ax + Bu$$
$$y = Cx \qquad (1)$$

We assume that this system is controllable and observable and that n, p and q are the dimensions of vectors x,u,y respectively. Our problem is to demonstrate a capability of artificial neural networks to stabilize linear plants.

Here we would like to construct a controller and a feedback

$$\dot{z} = Fz + Gy + Hu$$
$$u = K_1 z + K_2 y + r \qquad (2)$$

such that the overall system 1-2 is stable, where z is an n-q dimensional vector and r is an external input. First, we want to formulate this control problem in terms of an interpolation problem for a neural network with linear nodal functions and with six layers. The overall system (1-2) can be presented in the form:

$$\dot{w} = Tw + Ru$$

where

$$T = \begin{vmatrix} A + BK_2C & BK_1 \\ GC + HK_2C & F + HK_1 \end{vmatrix} \quad \text{and} \quad R = \begin{vmatrix} B \\ H \end{vmatrix}$$

Furthermore, T can be represented as a product of three matrices

$$A^1A^2A^3 \quad \text{where} \quad A^1 = \begin{vmatrix} A & zeros(n,n) & B \\ zeros(n-q,n) & F & G & H \end{vmatrix}$$

is a (2n-q)X(2n+p) matrix, $A^2 = \begin{vmatrix} & I_{2n} & \\ zeros(p,n) & K_1 & K_2 \end{vmatrix}$ is a

(2n+p)X(2n) matrix, and $A^3 = \begin{vmatrix} I_{2n-q} & \\ C & zeros(q,n-q) \end{vmatrix}$ is a (2n)X(2n-q) matrix. Now we recall a Lyapunov's Theorem [2] which says that a matrix T which has a full rank is stable if and only if for any positive definite matrix N there exists a positive definite matrix M such that T' M+MT=-N, where T' is a transpose of T. In this theorem N can be taken to be a unit matrix rather than an arbitrary matrix. Note also, that for any positive definite matrix M there exists a matrix g such that M=gg'. Therefore, in order to stabilize system (1) with a controller of system (2) there should exist such matrices F,G,H,K_1,K_2,g such that

$$T\,gg'+gg'\,T+I_{2n-q}=0 \qquad (3)$$

Now, expression $gg'T$ is a product of six matrices g,g',A^1,A^2,A^3 and a unit matrix (in this order). Therefore, this expression can be viewed as a realizing function of a six layer neural network with weights of synaptic interconnections given by matrices g,g',A^1,A^2,A^3, with input vectors being columns of the unit matrix and with linear nodal functions. Analogously, expression $T\,gg'$ is a realizing function of another neural networks with synaptic weight matrices transposed and in reverse order. Taking a discrete sum N of these two neural networks we see that equation (3) is equivalent to an interpolation problem for a neural network. This reduces our stabilization problem for a plant (1) to an interpolation problem (3) for a neural network N. Interpolation problems for a given neural network are typically solved by a backpropagation method. Notice that in our case some of the weights of the neural network are fixed and the only variables weights are the ones that correspond to the unknown matrices F,G,H,K_1,K_2,g. Let matrix E be given by the following formula:

$$E=T\,gg'+gg'\,T+I$$

and let SSE be the sum of squared elements of E. Then the following formulas are true:

$$\frac{\partial SSE}{\partial g} = 4(TE + ET')g, \qquad \frac{\partial SSE}{\partial A^1} = 4gg' E(A^3)'(A^2)', \qquad (4)$$

$$\frac{\partial SSE}{\partial A^2} = 4(A^1)' gg' E(A^3)', \qquad (5)$$

In order to solve equation (3), we need to drive SSE to 0, therefore a training of the neural net can proceed via a gradient descent method using equations (4) and (5) with the obvious modification due to the fact that the only elements in matrices A^1, A^2, A^3 that change are the elements of matrices F, G, H, K_1, K_2. This gradient descent method has to be slightly modified, so that we proceed by training each layer separately rather than all layers at once in order to avoid getting stuck in local minima. We used momentum and adaptive learning rate in our gradient descent method to speed up learning. The method described above has been successfully used to stabilize a specific numerical example with a nonminimal phase plant.

A NEW METHOD OF LEARNING FOR FEEDFORWARD NEURAL NETWORKS

We have successfully implemented a new method of training neural nets which is a combination of gradient descent and a Newton-Raphson method.This method significantly decreases the learning time. The details are presented below. Let us consider a neural network N with k variable weights. We can represent each set of weights as a vector w in a k-dimensional Euclidean space R^k. Let P=$\{p_i\}$ be a finite set of input vectors to a neural network N and let T =$\{t_i\}$ be the set of corresponding target output vectors. For every set of weights w from R^k there is a set of output vectors O=$\{o_i\}$ of N corresponding to the set of inputs. Then the sum-squared error function $f(w)$ is defined as $f(w) = \sum_i \|o_i - t_i\|$ .Our goal is to drive the errror function to 0, i.e. to find a set of weights w such that $f(w) = 0$. We try to find this w by an iteration process starting from some w_0 Let w_i be defined then we assume that there exists a vector Δ from the weight space R^k such that $f(w_i + \Delta) = 0$ and let $f(w_i + \Delta) - f(w_i) = \nabla f(w_i)\Delta$ be a first order approximation. Then in order to find Δ we need to solve the equation

$$f(w_i) + \nabla f(w_i)\Delta = 0 \qquad (7)$$

This is an underdetermined linear equation with respect to Δ. Let us add to (7) another system of k equations:

$$\alpha\Delta = 0 \qquad (8)$$

where α is some nonzero constant. The solution of (7)-(8) does not exist if $\nabla f(w_i)$ is nonzero, however we can find the least squares solution Δ. Then we define $w_{i+1} = w_i + \Delta$. Continuing to update w_i in this fashion we hope to find w_i arbitrary close to a solution of $f(w_i) = 0$. In this algorithm we vary α in order to minimize the error. Notice that α serves as a magnitude control factor so that the magnitude of solution of (7)-(8) decreases as α increases. In this method we start with weight vector w_0 obtained after some initial gradient descent training and then proceed with the method just described. The obtained results are very encouraging. We applied this method to the control problem described above to compare it with the gradient descent method and found that the speed of conversion increases 8-10 times.

CONCLUSION

We have described several interrelated problems in the area of neural network computations. First we considered an interpolation problem, then we have shown how to reduce a control problem to a problem of interpolation by a neural network via Lyapunov function approach, and finally we introduced a new, faster method of learning as compared with the gradient descent method.

REFERENCES

1. Richard W. Dabney, Application of Neural Networks to Autonomous Rendezvous and Docking of Space vehicles.
AIAA Space Programs and Technologies Conference, March 23-27, 1992, huntsville, AL.
2. C.T. Chen, Linear System Theory and Design, Holt,Rhinehart and Winston, New York, 1984.