

Conf-9206102--2

# NEURAL NETWORK ERROR CORRECTION FOR SOLVING COUPLED ORDINARY DIFFERENTIAL EQUATIONS

CONF-9206102--2

DE92 007241

R. O. Shelton,<sup>1</sup> J. A. Darsey,<sup>2</sup> B. G. Sumpter,<sup>3</sup> and D. W. Noid<sup>3</sup>

\*Research sponsored by the Division of Materials Sciences, Office of Basic Energy Sciences, U.S. Department of Energy, under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems, Inc., and Software Technology Branch of NASA.

<sup>1</sup>Lyndon B. Johnson Space Center, Houston, Texas 77058

<sup>2</sup>University of Arkansas/Little Rock, Little Rock, Arkansas

<sup>3</sup>Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831

## ABSTRACT

A neural network is demonstrated to learn errors generated by a numerical algorithm for solving coupled nonlinear differential equations. Comparisons are made for training the neural network using backpropagation and a new method which is found to converge with fewer iterations.

### I. INTRODUCTION

The molecular dynamics method provides a powerful and versatile tool for studying the microscopic behavior of matter. During the past two decades, the molecular dynamics (MD) method has been used to complete detailed studies of the time-dependent motion of a system of particles, providing information on the mechanisms for processes which occur on both the microscopic and macroscopic level.<sup>1,3</sup> While the fundamental applications of the molecular dynamics method are broad, there still remain some technological problems with computational complexity.

One fundamental problem is the limitation due to numerical integration errors. The numerical solution of the time dependence for a set of coupled ordinary differential equations that describe a given system of particles can only be obtained to within a small but finite error. Various techniques and formulae have been used for solving a set of first-order coupled ODEs such as Runge-Kutta, Adams method, etc. (see for a review Refs. 4-5). In each case, an error will be produced depending on the time step, order of method, and characteristics of the equation and solution. In this paper we propose a new method for correcting errors for this type of numerical solution of ODEs. The method is based on using a neural network to correctly learn the error generated by, for example, Runge-Kutta on a model molecular dynamics problem. The neural network programs used in this study are ones developed by NASA. A fast learning approach is also contrasted with a more traditional backpropagation method. The neural net programs are discussed in the next section. In the third section, we discuss the MD model and calculations, followed by our conclusions.

### II. FAST LEARNING UTILITY FOR BACKPROPAGATION (FLUB)\*

For the purpose of this discussion, we assume familiarity with the basic gradient descent algorithm for determining weights for feed-forward networks.<sup>7</sup> Presented here is a brief description of an accelerated training method for such networks. The next section includes a performance comparison between the accelerated method and a conventional neural network simulation package (NETS).<sup>8</sup> As is well known, finding an acceptable set of weights for a feed-forward network of even moderate size may be extremely difficult, primarily because of the large number of parameters which must be found. The actual nonlinearities in the resulting optimization problem are very mild and result mainly from two sources. These are

- (1) the error function, and
- (2) the nonlinear transfer function for the network.

The standard RMS error function produces a quadratic nonlinearity because the function itself is formed by summing the squares of network errors. The transfer function introduces a more complex global nonlinearity, but in local regions of weight space, the sigmoids are modeled well by linear approximation. The accelerated method is based on a number of engineering compromises which exploit common features of popular three-layer feed-forward architectures. As such, the strategy ignores issues which may be

The submitted manuscript has been authored by a contractor of the U.S. Government under contract No. DE-AC05-84OR21400. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

N92-26407

Unclass  
0094610

G3/64

(NASA-TM-101795) NEURAL NETWORK ERROR CORRECTION FOR SOLVING COUPLED ORDINARY DIFFERENTIAL EQUATIONS (NASA) 6 P

## DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

important in some problems, but in many cases it is extremely effective. In particular, the accelerated method should be considered when most of the connections in the network are between the input and hidden layers.

The complexity of the weight determination problem is managed by reducing the number of optimization variables which are simultaneously active. One obvious means for such a reduction is to perform a sequence of partial optimizations; i.e., temporarily freeze most of the optimization parameters and concentrate on a small number of active variables. After the objective function is brought to the best value possible by means of manipulating the active variables, a new subset of variables is activated, while all others are frozen. The process continues until an acceptable value of the error function is obtained or no further reduction is possible. The effectiveness of this technique is strongly dependent on the way that the active sets of variables are selected. In the present case, the decomposition of the weight space is suggested by the architecture.

Denote the matrix of weights between the input and hidden layer as  $W_0$  and those between the hidden and output layers as  $W_1$ . We arrange both matrices so that the entry  $w(i, j)$  is the strength of the connection leading from neuron  $j$  to neuron  $i$ . We therefore term the  $i$ th row of a weight matrix to be the weight vector associated with neuron  $i$ , where we associate weights with the neuron to which the corresponding connections lead. This choice of notation is arbitrary, but apparently popular, in part because the actions of the weights on the vector of outputs  $X$  of a layer may be viewed as a multiplication of the vector  $X$  by the weight matrix associated with the target layer. Suppose that the three-layer network has  $n_0$  input nodes,  $n_1$  hidden nodes, and  $n_2$  output nodes. Let  $X$  be the inputs for the network organized as a matrix having  $n_0$  rows and  $p$  columns where  $p$  is the number of examples comprising the training set. The activations of the hidden nodes may be organized in similar fashion as a  $n_1 \times p$  matrix  $Y$  where

$$Y = W_0 X . \quad (1)$$

If  $f$  is the nonlinear transfer function for the network, then denote  $F$  to be the mapping  $f$  applied to a matrix element-wise. Therefore, if the hidden layer outputs are also organized as a  $n_1 \times p$  matrix  $\hat{Y}$ , we have

$$\hat{Y} = F(Y) . \quad (2)$$

In the present implementation, the function  $f$  is given by

$$f(x) = 1/(1 + \exp(-x)) ; \quad (3)$$

however, in practice,  $f$  can be any differentiable function. Continuing in similar fashion, the output activations will be denoted by the  $n_2 \times p$  matrix  $Z$  where

$$Z = W_1 \hat{Y} , \quad (4)$$

and the network outputs by the  $n_2 \times p$  matrix  $\hat{Z}$  where

$$\hat{Z} = F(Z) . \quad (5)$$

If the desired outputs for the network are denoted by the  $n_2 \times p$  matrix  $T$ , then the energy (or error) function  $E$  for the network is given by

$$E = \left( \frac{1}{2} \right) \sum_{i=1}^{n_2} \sum_{j=1}^p (\hat{z}(i, j) - t(i, j))^2 . \quad (6)$$

The RMS error referenced in the following section is defined to be

$$\text{RMS} = 2E / (pn_2) \quad (7)$$

An architecturally natural decomposition of the weight space is to consider  $W_0$  separately from  $W_1$ . Moreover, when  $W_1$  is considered alone, it is relatively easy to produce optimal output weights. This is true for two reasons. First, the variables comprising  $W_1$  may be further subdivided with no penalty. The energy function  $E$  is structured so that

$$\frac{\partial^2 E}{\partial w_{i1,j} \partial w_{i2,k}} = 0 \quad (8)$$

where the subscripted  $w$  denotes an output weight and  $i1$  is distinct from  $i2$ . This means that each row of  $W_1$  may be determined independently. Second, if the output sigmoids are linearized, then the problem of determining each row of  $W_1$  reverts to a well known quadratic optimization problem. These output weight vectors are determined using a pseudo-inverse technique modified to produce new weight vectors which are minima of the quadratic linearized energy function subject to the side condition that they are as close as possible ( $L^2$ -wise) to the previous vector. The side condition provides solutions which obey Widrow's minimum disturbance rule,<sup>9</sup> and, unlike normal least squares which is a one-shot process, the modified technique may be executed iteratively.

The preceding discussion provides a method for finding an optimal  $W_1$  given a fixed  $W_0$ . The remainder of the accelerated training method is concerned with obtaining the matrix  $W_0$ . Unlike  $W_1$ ,  $W_0$  cannot be decomposed by rows. Instead, the decomposition employed is based on the algebraic structure of the input space (column space of the matrix  $X$ ). Denote this vector space as  $C$ . To motivate this view, observe that any gradient update to a row of  $W_0$  is contained in  $C$ . A very useful ortho-normal basis for  $C$  is provided by the singular value decomposition (SVD) of  $X$ .<sup>10</sup> The SVD of  $X$  is described by the equation

$$X = LSR^t \quad (9)$$

where  $S$  is a square diagonal matrix whose diagonal entries are positive numbers generally known as the singular values of  $X$ , and the matrices  $L$  and  $R$  are orthogonal, i.e.,  $LL^t = RR^t = I$ . The column vectors of  $L$  form a basis for  $C$ , while the column vectors of  $R$  provide a basis for the row space of  $X$ . The size of the corresponding singular value is a measure of the importance of a singular vector in the basis representation of  $C$ . If a singular value of  $X$  is 0, then the corresponding singular vector (column of  $L$ ) is perpendicular to  $C$  and thus to any gradient update to  $W_0$ . For the purpose of determining the weights  $W_0$ , each row  $W(i)$  of  $W_0$  is represented as follows:

$$W(i) = \sum_{j=1}^r c(i, j) L(j) \quad (10)$$

where  $c(i, j)$  are variable coefficients,  $L(j)$  is the  $j$ th column of  $L$  ( $j$ th singular vector of  $X$ ), and  $r$  is some number of singular vectors chosen to provide the representation of  $C$ . Note that  $r$  should always be at most equal to the rank of  $X$ . In most cases, lesser values of  $r$  prove to be good choices because the mathematical rank of  $X$  is usually larger than the dimension of the space which is needed to represent important features of the input space. Exclusion of singular vectors corresponding to lesser singular values forces the optimization process to run on a reduced rank representation of the input space. In addition to pruning the size of the problem, using a reduced rank representation often suppresses noise.

At each step of the process, a special  $j(i)$ ,  $1 \leq j(i) \leq r$  is selected, and the vector of coefficients  $c(i, j(i))$ ,  $i = 1, \dots, n_1$  is manipulated to determine optimal values of the coefficients  $c(i, j(i))$  which minimize  $E$  subject to the condition that all other parameters remain constant. Following this optimization step, an optimal  $W_1$  is found using the process described at the beginning of the section. This process is repeated until an acceptable error is obtained or no further reductions are possible. At each step the choices of  $j(i)$  are re-evaluated. The evaluation process simply consists of a sensitivity analysis which, for each  $i$ ,  $1 \leq i \leq n_1$ , determines which coefficient  $c(i, j)$  changes the error function the most for a given change in  $c(i, j)$ .

The weakness of this method is that the coefficient selection process only crudely approximates even simple gradient descent. Despite this shortcoming, it often works because there are usually many choices for the weight matrix  $W_0$  for which there is an acceptable output matrix  $W_1$ .  $W_0$  is a feature detector which, in the accelerated method, is strongly based on the algebraic features of the input space. The big advantage of this method is that the entire matrix  $W_0$  can be manipulated in a nearly optimal manner by only handling a relatively small number of coefficients  $c(i, j(i))$ . Observe that there are only  $n_i$  such coefficients active at any given time, whereas conventional optimization methods must consider  $n_o \times n_i$  coefficients to find the matrix  $W_0$ .

The computational complexity of this algorithm depends on several factors: (1) the one-time cost of executing an SVD on the matrix  $X$ ; (2) the once-per-cycle cost of executing the pseudo-inverse solution for  $W_1$ ; (3) the once-per-cycle cost of performing the sensitivity analysis needed to isolate the set of coefficients  $c(i, j(i))$  for optimization of  $W_0$ ; and (4) manipulation of the coefficients  $c(i, j(i))$  to obtain their optimal values.

For networks with large input spaces, (1) is the dominant term. It is important to observe that (1) only need to be redone if the input space changes. Changes in network size, desired outputs, or learning parameters do not require recomputation of (1). Our experience with networks of widely differing sizes has shown that the per-cycle cost of FLUB is two to three times that of NETS; however, the cycle costs of FLUB tend to be inflated due to the relatively small number of cycles necessary for convergence and the one-time cost of the initial SVD.

### III. CALCULATIONS AND RESULTS

Hamiltonian systems are classified as integrable or nonintegrable, depending on whether a separation of variables can be found. Nonintegrable Hamiltonian systems, such as ones derived for molecular dynamics, are usually integrated by various numerical methods. In the 1960's it was demonstrated that some simple Hamiltonian systems exhibited, for some regions of phase space, chaotic motion.<sup>11</sup> These models were found to be relevant to atomic and molecular dynamics.<sup>12</sup> However, an important question arose about the accuracy of such numerical integrations. The most commonly used methods are characterized by the order of series in time that the techniques implement. In this study we have used a well-known model from the chemical physics literature to determine if the numerical errors that are generated can be corrected using neural networks.

Our Hamiltonian is a simple two-degree freedom system with the kinetic energy part represented by

$$T = \frac{1}{2} P_x^2 + \frac{1}{2} P_y^2 \quad (11)$$

where  $P_x$  and  $P_y$  are Cartesian momenta for the  $x$  and  $y$  motion. The potential function is nonseparable and of the form of two harmonic oscillators coupled through a quadratic term  $xy^2$ . The total Hamiltonian is

$$H = \frac{1}{2}(P_x^2 + P_y^2) + \frac{1}{2}\omega_x^2 x^2 + \frac{1}{2}\omega_y^2 y^2 + \lambda(xy^2 + \eta x^3) \quad (12)$$

Hamilton's equations ( $\dot{q} = \partial H / \partial p$ ,  $\dot{p} = -\partial H / \partial q$ ) were numerically integrated using the methods described in the next paragraph. The parameters  $\omega_x = 1$ ,  $\omega_y = 1$ ,  $\lambda = \sqrt{.0125}$ , and  $\eta = -1/3$  for Case I. In Case II, we allowed the  $\omega_x^2$  and  $\omega_y^2$  to be randomly chosen in the range 0.9 to 1.1. Although the model seems simple, it actually was one of the first to demonstrate both quasiperiodic and chaotic dynamics and has been extensively studied.

We have chosen to study the error generated by the 4th-order fixed-step Runge-Kutta method. A high-level integration was obtained using the 12th order ODE method and requiring an accuracy of 12 digits. Every 20 time steps of 0.1, a training example was obtained by computing the error in  $\Delta x$ ,  $\Delta y$ ,  $\Delta P_x$ , and  $\Delta P_y$  from the trajectory. Eighty points from the trajectory were chosen to represent a segment of the Runge-Kutta trajectory and should be related to the errors  $\Delta$ 's. Twenty segments were chosen for 20

randomly chosen trajectories in an energy range 0 to 10 (in our units  $h = 1$ ; see Ref. 12). Four hundred examples were thus developed which should contain both chaotic and quasiperiodic trajectory segments. Three-fourths of the examples was used for training, and one-fourth was used to test the ability of the network to generalize the data. Our fully connected network contained 80 inputs (20 sequential points of our trajectory), 4 outputs which were corrections to our trajectory, and a single hidden layer containing 40 nodes.

Using NETS 2.01 we trained the network for Case I to a maximum error of 0.07 and rms error of  $\sim 0.02$ . The test Case I data found a maximum error of 0.16 and an rms error of 0.03. For the second example wherein the zeroth order Hamiltonian was altered, the network was trained to a maximum error of 0.1 and rms error of  $\sim 0.03$ . The test data error were found to generate a maximum error of  $\sim 0.18$  with an average error of only  $\sim 0.05$ . Using this correction, the trajectory error would be substantially reduced. We have not investigated any other network topologies or parameters. However, it seems reasonable to assume that more information about the trajectories would lead to a more accurate neural network calculation. Also, because more degrees of freedom are required for most practical models, a much larger network would be required. For this reason we have generated preliminary results on a "fast" learning method called FLUB. For a training criteria of 0.1 for both NETS and FLUB, we found on the CRAY-XMP that in Case I NETS required 433 cycles, which is contrasted to 4 cycles for FLUB. In our computer calculation of Case II, we found NETS required 768 cycles compared to 28 cycles for FLUB.

#### IV. CONCLUSIONS

In our preliminary study of neural network error correction to numerical integrations, we have found that:

- (1) In a general Hamiltonian system, the numerical errors are related to the dynamical trajectory.
- (2) A neural network can learn this error relationship and make useful generalizations of the error criteria, thereby giving significantly more accurate solutions in approximately the same computation time.
- (3) The fast learning method FLUB can reduce the time needed for training over the standard backpropagation technique by several orders of magnitude.

Finally, we suggest that this technique may provide a general method to correct numerical integration errors in some algorithms.

#### REFERENCES

- (1) Fincham, D.; Heyes, D. M. *Adv. Chem. Phys.* 1985, 63, 493.
- (2) Klein, M. L. *Ann. Rev. Phys. Chem.* 1985, 36, 5525.
- (3) Hoover, W. G. *Ann. Rev. Phys. Chem.* 1983, 34, 103.
- (4) Shampine, L. F.; Gordon, M. K. *Solutions of Ordinary Differential Equations--The Initial Value Problem*; W. H. Freeman Press: San Francisco, 1975.
- (5) Hull, T. E.; Enright, W. H.; Felin, B. M.; Sedgwick, A. E. "Comparing Numerical Methods for Ordinary Differential Equations," *SIAM J. Numer. Anal.* 1972, 9 603.
- (6) Shelton, R. "Fast Learning in Feed Forward Neural Networks by Constrained Weight Modification and Partial Optimization" (to be submitted).
- (7) Rummelhart, D. and McClelland, J. In *Parallel Distributed Processing*, Vol. 1; MIT Press: Cambridge, MA, 1986.
- (8) Baffes, P. T. NNETS Program Version 2.01, Johnson Space Center No. 23366, September 1989.
- (9) Widrow, B. "Generalization and Information Storage in Networks of Adaline 'Neurons'," *Self Organizing Systems* (Yovitz, M. C.; Jacobi, G. T.; and Goldstein, G. D., eds.), pp. 435-461; Spartan Books: Washington, DC, 1962.
- (10) Golub, G. and Van Loan, C. *Matrix Computations*; Johns Hopkins University Press, 1983.
- (11) Henon, M. and Heiles, C. *Astron. J.* 1964, 69, 73.
- (12) Noid, D. W.; Koszykowski, M. L.; and Marcus, R. A. *Ann. Rev. Phys. Chem.* 1981, 32, 267.