Application of neural network computing to the solution for the ground-state eigenenergy of two-dimensional harmonic oscillators

J.A. Darsey
Department of Chemistry, The University of Arkansas/Little Rock, Little Rock, AR 72204, USA

D.W. Noid [1]
Chemistry Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831-6182, USA
and Department of Chemistry, The University of Tennessee/Knoxville, Knoxville, TN 37996-1600, USA

and

B.R. Upadhyaya
Department of Nuclear Engineering, The University of Tennessee/Knoxville, Knoxville, TN 37996-2300, USA

Received 13 November 1990

In this Letter, we teach a neural network the solution of the Schrödinger equation for some model potential energy functions. The network can then predict eigenvalues of other test cases with an error of a few percent.

1. Introduction

A neural network could be defined as "a computing system made up of a number of simple, highly interconnected processing elements, which processes information by its dynamic state response to external inputs" [1]. The inspiration for neural network research comes primarily from studies involving the brains of higher life forms, more particularly the cerebral cortex of mammalian brains.

Neural network models are specified fundamentally by network architectures, transfer functions and learning laws [2,3]. Neural networks do not store information in separate memory arrays as in more conventional computers. They typically "learn" by taking weighted sums of inputs and comparing their results to known outputs, after having applied an appropriate transfer function. The result of this learning process is information stored in a weighted distribution of connections between the same set of inputs and outputs [1–3].

In this work, we were interested in whether we could use neural network computing to learn a series of transformations from a potential energy surface to the corresponding ground state eigenvalue energy. In this way, the neural network is learning the transformation generated by the Schrödinger equation. In all previous work, very complex and tedious calculations were needed to solve the Schrödinger equation, but in our calculations a neural computer associates a potential function with an eigenvalue or set of eigenvalues. We chose for this initial test case a simple two-dimensional harmonic oscillator. We trained our network on 41 different potential energy surfaces on which the corresponding eigenenergy was provided and then asked our "trained" neural network to predict the eigenenergies for an additional

[1] Research sponsored by the Division of Materials Sciences, Office of Basic Energy Sciences, US Department of Energy, under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems, Inc., and the Materials Research Division of the NSF, Grant #DMR-8818412.

0009-2614/91/$ 03.50

22 surfaces. We selected values of our parameters both within and external to the values used in training. In all cases, as will be seen in section 4, predicted values of eigenenergies were within a range of error between 0.14 and 5.4%, with an average error of about 1.6%. We feel that this small average error demonstrates the feasibility of using neural network computing for solving these types of problems, at least for this simple system. However, these results strongly suggest that much more complicated systems are approachable by this technique, and future work will deal with these more complex systems.

2. Theory of neural network computing

An artificial neural network is a parallel distributed processing system. In a multi-layer network, the information is propagated from the input layer (measured data) to the output layer (parameters to be estimated) in a manner that the relationship between input and output is generally nonlinear. This property of a multi-layer network lends itself to efficient interpolation and estimation of a set of parameters from measurements. Neural network paradigms have been successfully applied to pattern classification, sensor validation [4], EKG analysis, adaptive control, and others.

In a feed forward network, the processing elements (PEs) are interconnected through unidirectional information channels or connections. A typical processing element is shown in fig. 1. The output y is given by the nonlinear transformation

$$y=f\left(\sum_{i=1}^{N} w_i x_i-\theta\right), \tag{1}$$

where θ is a node bias and $\{w_i\}$ are the connection weights. Various types of threshold functions are used in networks [3], the most common being the sigmoidal function

$$f(z)=\frac{1}{1+e^{-\beta z}}, \quad \beta>0. \tag{2}$$

This satisfies the nonlinear differential equation [5]

$$df/dz=\beta f(1-f). \tag{3}$$

Thus, the relationship between two sets of information can be approximated, in general, by any degree of nonlinearity. The key to the success of developing a neural network to model this relationship is to train the network with as many examples as possible. An artificial neural network should not simply memorize patterns, but generate an implicit functional relationship between "input–output" data. This unique property of neural networks has been exploited in this paper.

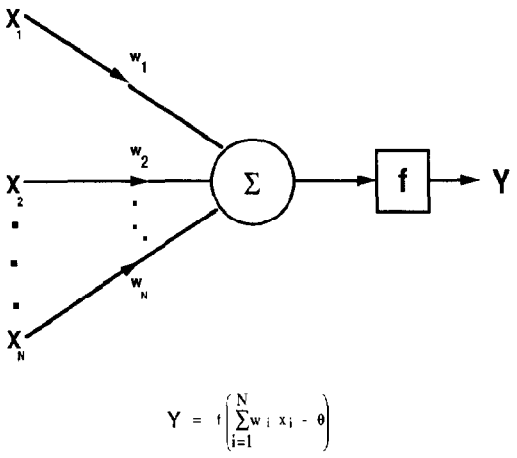

Fig. 1. A typical processing element (PE).

The training of a three-layer, feed forward and fully connected network was performed using the backpropagation algorithm, first developed by Rumelhart and McClelland [6]. Experiments with three- and four-layer perceptrons indicated that a three-layer perceptron is more than adequate in application to interpolation problems. This is also an indication that the mapping interface is fairly regular. The backpropagation network (BPN) algorithm is an iterative gradient technique that minimizes the global error between the actual output and the network output. The BPN algorithm requires a continuously differentiable nonlinear function, such as the sigmoidal function (see eq. (2)). The connection weight update is given recursively by

$$w_{ij}^{\rho}(k+1)=w_{ij}^{\rho}(k)+\alpha\delta_j^{\rho}x_i^{\rho-1}+\mu[(w_{ij}^{\rho}(k)-w_{ij}^{\rho}(k-1)], \quad 0<\mu<1. \tag{4}$$

w_{ij}^{ρ} is the connection weight between PE i in layer $\rho-1$ and PE j in layer ρ. δ_j^{ρ} is referred to as the gen-

eralized delta function, and the last term is called the momentum term. This latter eliminates stagnancy at local minima. Typical values of μ range in the interval (0.5, 1) and the weight update coefficient α has the range (0.4, 0.8). The "error function" δ_j^P in the computation of correction weights between the last two layers is given by

$$\delta_j^P = x_j^P(1-x_j^P)(t_j - x_j^P)\,, \tag{5a}$$

and for the correction weights between hidden layers (lower level corrections)

$$\delta_j^P = x_j^P(1-x_j^P)\sum_i \delta_i^{P+1} w_{ji}^{P+1}\,. \tag{5b}$$

Note that σ_j^P at a lower level P is a function of σ_i^{P+1} corresponding to the error at the layer $P+1$. Because of this property, the algorithm is referred to as the back propagation network. t_j is the target value of the jth output node. The output of node j at layer P is given by

$$x_j^P = \left[1+\exp\left(-\sum_i w_{ij}^P x_i^{P-1} + \theta_j^P\right)\right]^{-1}\,. \tag{5c}$$

The iterative procedure continues until the global error is less than a preset tolerance. The evolution of the correction weights w_{ij} during training, or their pattern for a trained network, is not studied in this paper. This is a very important aspect of quantifying network performance.

3. Calculations

The potential energy surfaces generated were those of a two-dimensional harmonic oscillator given by the equation

$$V(x,y) = \tfrac{1}{2}\omega_x^2 x^2 + \tfrac{1}{2}\omega_y^2 y^2\,, \tag{6}$$

where the ground state eigenvalues were given by ($H = \frac{1}{2}(P_x^2+P_y^2)+V$, $\hbar=1$, and $M=1$)

$$E_{0,0} = \tfrac{1}{2}\omega_x + \tfrac{1}{2}\omega_y\,. \tag{7}$$

These examples are similar to previously studied coupled oscillator systems [7]. For the present calculations, the separable Hamiltonian was chosen but would add no difficulty in our calculations. We produced 41 potential energy surfaces with corresponding eigenenergies for training our network. We have chosen the equilibrium of the potential function to be at $E=0$. For more general application, we plan not to do this, but then we expect to have to further train the network accordingly. The initial values chosen were for x and y ranging from -3 to $+3$ in increments of plus one. The grid values for ω_x ranged from 0.5 to 0.9, and the values of ω_y ranged from 1.1 to 1.8 in steps of 0.1 and the example $\omega_x = \omega_y = 1$. Each potential energy surface had a total of 49 points, and there were a total of 41 potential energy surfaces.

In the training part of this project, we chose a standard three-layer network architecture. Our input layer of nodes consisted of the 49 grid points of each potential energy surface. Our second layer consisted of 40 nodes, and our output layer was a single node representing the appropriate eigenenergy. Our network was a completely interconnected one; that is, every node was connected to every other node as explained in section 2. The weights assigned to each interconnection were originally assigned via a random number generator and were changed via the generalized delta rule at each cycle. We stored these weights after 2738 cycles, 4441 cycles, and 4501 cycles. It required approximately 12 h on a VAX 2000 workstation to reach 4501 cycles. The results of predicting the eigenenergies for various values of ω_x and ω_y sampled after different numbers of cycles are shown in table 1.

A second series of potential energy surfaces were produced in which the x and y axes were varied between -2.5 and $+2.5$ in increments of plus one. This had the effect of sampling a different region of the potential energy surfaces. It also reduced the total number of input data points and therefore input nodes from 49 to 36. Training was again performed with the 41 values of ω_x and ω_y of table 1. The connection weights were stored after 2739 cycles and 3740 cycles. Table 2 summarizes the results of calculations using this latest set of training surfaces.

One final point should be mentioned concerning the preparation of the input data. In order to use the particular software program available, it was necessary to normalize all potential energy and eigenenergy values to numbers between 0.1 and 0.9. The exact reasons for this are briefly discussed in section 2, and for a more detailed explanation one may refer to refs. [1–4] and references therein.

Table 1
Results of predicted eigenenergy values with corresponding actual eigenenergies and associated error for $-3 \leqslant x,y \leqslant 3$

ω_x	ω_y	2738 cycles		4441 cycles		4501 cycles		Actual energy [a]
		eigenenergy predicted	% error	eigenenergy predicted	% error	eigenenergy predicted	% error	
0.60	1.00	0.824	3.01	0.831	3.83	0.839	4.84	0.800
0.70	1.00	0.857	0.86	0.864	1.69	0.872	2.62	0.850
0.80	1.00	0.879	2.31	0.886	1.55	0.895	0.63	0.900
0.90	1.00	0.908	4.41	0.898	5.44	0.907	4.51	0.950
0.40	1.00	0.568	2.11	0.570	1.77	0.570	1.81	0.580
0.40	1.40	0.852	5.33	0.865	3.85	0.879	2.30	0.900
0.50	1.45	0.943	3.28	0.953	2.29	0.967	0.79	0.975
0.55	1.40	0.949	2.66	0.959	1.60	0.973	0.19	0.975
0.60	1.90	1.228	1.77	1.240	0.78	1.256	0.51	1.250
0.95	1.90	1.382	3.00	1.408	1.18	1.418	0.46	1.425
0.50	1.90	1.160	3.37	1.168	2.69	1.185	1.22	1.200
0.60	1.90	1.220	2.42	1.233	1.39	1.248	0.14	1.250
0.70	1.90	1.274	2.02	1.291	0.66	1.305	0.39	1.300
0.80	1.90	1.319	2.31	1.341	0.69	1.353	0.19	1.350
0.90	1.90	1.355	3.20	1.379	1.48	1.390	0.73	1.400

[a] Actual eigenenergies were calculated using eq. (2).

Table 2
Results of predicted eigenenergies with corresponding actual eigenenergies and associated error for $-2.5 \leqslant x,y \leqslant 2.5$

ω_x	ω_y	2739 cycles		3740 cycles		Actual energy [a]
		eigenenergy predicted	% error	eigenenergy predicted	% energy	
0.60	1.00	0.811	1.39	0.803	0.39	0.800
0.70	1.00	0.865	1.75	0.854	0.44	0.850
0.80	1.00	0.920	2.17	0.906	0.66	0.900
0.90	1.00	0.975	2.63	0.960	1.01	0.950
0.40	1.00	0.588	1.33	0.586	1.01	0.580
0.40	1.40	0.897	0.28	0.904	0.46	0.900
0.50	1.45	0.967	0.80	0.979	0.37	0.975
0.55	1.40	0.966	0.88	0.977	0.18	0.975
0.60	1.90	1.259	0.71	1.254	0.28	1.250
0.95	1.90	1.413	0.81	1.402	1.61	1.425
0.50	1.90	1.211	0.91	1.198	0.15	1.200
0.60	1.90	1.254	0.28	1.245	0.37	1.250
0.70	1.90	1.297	0.26	1.291	0.68	1.300
0.80	1.90	1.340	0.74	1.334	1.21	1.350
0.90	1.90	1.383	1.24	1.372	2.02	1.400
2.00	2.00			3.91	2.17	4.00
3.00	3.00			8.61	4.35	9.00
4.00	4.00			16.62	3.89	26.00
5.00	5.00			24.46	2.17	25.00

[a] Actual eigenenergies were calculated using eq. (2).

4. Results and discussion

Table 1 presents predicted values of the eigenenergies for various potential energy surfaces. A grid of 49 points was produced for each set of ω_x and ω_y values listed. The grid was produced for integer values of x and y, being varied from -3 to $+3$ inclusive, using eq. (1).

In the training phase of the project, values of ω_x were varied between 0.5 and 0.9, and values of ω_y were varied between 1.1 and 1.8. One additional training surface was produced for $\omega_x=1.0$ and $\omega_y=1.0$. For each training surface, the appropriate ground state eigenenergy was provided as output.

Results from table 1 after 2738 cycles show that the predicted eigenenergies ranged in accuracy from 4.41 to 0.86% with an average error of 2.85%. After 4441 cycles, the error ranged from 5.44 to 0.66% with an average error of 2.08%. After 4501 cycles, the error ranged from 4.84 to 0.14% with an average error of 1.39%. It should be noted that this trend of more accuracy with more learning cycles does not necessarily hold for individual eigenenergy determinations. For example, at $\omega_x=0.60$ and $\omega_y=1.00$, the accuracy decreases as the number of learning cycles increased, from a value of 3.01% at 2738 cycles to 3.83% at 4441 cycles to a maximum of 4.84% at 4501 cycles. On the average, however, the trend is more accuracy with additional learning cycles.

Similar results can be seen for the second set of calculations where the interval over which we varied x and y was -2.5 to 2.5 in increments of plus one (see table 2). The average error was 1.06% after 2739 cycles and 0.70% after 3740 cycles. It should be noted that the errors in our second series of calculations were considerably lower than in our first set, and the error after only 3740 cycles in our second series (0.70%) was about half the error after 4501 cycles for our first series of calculations (1.39%). This strongly indicates that the region of the energy surface sampled influences the accuracy of predictions based on these training surfaces, since we used the same delta x and delta y increments in all calculations and the exact same energy surfaces and number of surfaces in each set of runs.

A key feature to note in all these calculations is that we ask the trained neural network to predict values of eigenenergies for surfaces in which it was not trained. However, some of the eigenvalues it predicted were within the range in which we trained the network, and some of the eigenvalues were outside this range. For example (see tables 1 and 2), we trained the network for values of ω_x between 0.50 and 0.90 and for values of ω_y between 1.10 and 1.80. We asked the network to predict values of $\omega_x=0.50$ and $\omega_y=1.90$, or similarly $\omega_x=0.55$ and $\omega_y=1.40$. Both these sets of values would be within our range of training. However, we also chose values such as $\omega_x=0.40$, $\omega_y=1.00$, or $\omega_x=0.95$ and $\omega_y=1.90$, which lie outside the boundaries for training. The accuracy for determining the eigenenergies did not seem to depend on whether the regime of interest was within the training range or slightly outside the training range. However, we did check for the accuracy considerably outside the training range. For example (see table 2), for $\omega_x=2.00$ and $\omega_y=2.00$, we obtained an error of 2.17%. For $\omega_x=3.00$ and $\omega_y=3.00$, we obtained an error of 4.35%. However, for the values $\omega_x=4.00$ and 5.00 and $\omega_y=4.00$ and 5.00, we obtained a calculated error of 3.89% and 2.17%. It can therefore be noted that the network has appeared to learn how to "guess" eigenvalues reasonably well in regions considerably far from the regions it trained on. It should be pointed out, however, that these last several calculations were performed with the second neural network for which $-2.5 \leqslant x,y \leqslant 2.5$. Although this last set of values were all less than 4.35%, they were considerably higher than the average error (0.59%) calculated for the values of ω_x and ω_y closer to the values used for training.

5. Summary

In this initial study, we have explored the possibility of using the neural network method to learn the mathematical transformation of quantum mechanics. Because of our limited computer resource (simulation of a concurrent many-node network on a sequential computer), the input level was extremely limited. Hopefully, as more powerful neural computers are built, it will be possible to learn more examples at a greater detail so that more quantitative results are obtained. It should be noted that the transformation is reversible so that a potential function can be associated with a set of eigenvalues. Also,

after the neural network has learned the example, further calculations require only trivial amounts of CPU time.

Acknowledgement

We wish to thank the Nuclear Engineering Department for allowing us the use of their neural network codes and computer at The University of Tennessee. The authors would also like to thank Professor L.H. Tsoukalas and Professor R.E. Uhrig of the Nuclear Engineering Department of The University of Tennessee/Knoxville for their time and support during the course of this project, and especially to G. Mathai for extensive help with the computer systems and the use of his neural network program BPN.

References

[1] M. Caudill, AI Expert, December 1987, pp. 46–52.
[2] M. Caudill, AI Expert, February 1988, pp. 55–61.
[3] R.P. Lippmann, IEEE ASSP Magazine 4 2 (1987) 4.
[4] B.R. Upadhyaya, E. Evyurek and G. Mathai, Proceedings of the 1990 International Meeting on Fast Reactor Safety, Snowbird, UT, August 1990, Vol. 3, pp. 349–358.
[5] W.S. McCulloch and W. Pitts, Bull. Math. Biophys. 5 (1943) 115.
[6] D. Rumelhart and J. McClelland, Parallel distributed processing, Vol. 1 (Bradford Books/MIT Press, Cambridge, MA, 1986).
[7] D.W. Noid, M.L. Koszykowski and R.A. Marcus, Ann. Rev. Phys. Chem. 32 (1981) 267.