Generalization of Back-Propagation to Recurrent Neural Networks

Fernando J. Pineda
Applied Physics Laboratory, Johns Hopkins University, Laurel, Maryland 20707
(Received 10 June 1987)

An adaptive neural network with asymmetric connections is introduced. This network is related to the Hopfield network with graded neurons and uses a recurrent generalization of the δ rule of Rumelhart, Hinton, and Williams to modify adaptively the synaptic weights. The new network bears a resemblance to the master/slave network of Lapedes and Farber, but it is architecturally simpler.

PACS numbers: 87.30.Gy

The neural network approach is a paradigm for computation in which the traditional paradigm of a finite-state machine performing sequential instructions in a discrete state space is replaced with the paradigm of a dynamical system, in a discrete or continuous state space, which evolves under the control of a certain class of dynamics (*neurodynamics*). Although a precise definition of neurodynamics does not exist, it seems safe to characterize it by at least three salient features. First, the dynamical system has very many degrees of freedom. At the present time, most simulations of these systems are limited to less than 10^5 neurons. On the other hand, the human brain has at least 10^{11} neurons. The activity level and the time derivative of the activity of the neurons are the coordinates in the phase space of the system. This phase space plays the role of the state space in a conventional computing machine. The second feature of neurodynamics is nonlinearity. Nonlinearity is essential to create a universal computing machine. This follows because a network composed of linear units can always be reduced to an equivalent single-layer network which performs the same input/output transformation. But, as pointed out by Minsky and Papert,[1] a universal computing machine cannot be built from a single layer of finite-order neurons. The third feature of neurodynamics is that it is dissipative. A dissipative system is characterized by the convergence of the phase-space volume onto a manifold of lower dimensionality as time increases. Systems whose flow exhibits the property of *global asymptotic stability* play a particularly important role in neural-network modeling. Global asymptotic stability implies that the system will ultimately settle down to a steady state for any choice of initial condition. Systems which minimize an energy function, such as the Hopfield model, are guaranteed to be globally asymptotically stable.

The identification of stable fixed points with computational objects, e.g., memories, is one of the fundamental ideas of the paradigm. To implement this idea it is necessary to control the locations of the fixed points of the neural networks. A learning algorithm is a rule or dynamical equation which changes the locations of fixed points to encode information. One way of doing this is to minimize, by gradient descent, some function of the system parameters. This general approach is reviewed by Amari[2] and forms the basis of many learning algorithms. The algorithm described here is a specific case of this general approach.

The dynamics of the network considered in this Letter is based on the following system of coupled differential equations:

$$dx_i/dt = -\alpha x_i + \beta f_i\left(\sum_j w_{ij}x_j\right) + I_i, \tag{1}$$

where x_i represents the activity of the ith neuron, where the matrix element w_{ij} denotes the connection strength, or coupling, from the jth to the ith neuron, and where α and β are conveniently chosen positive constants. The functions f_i may have different forms for various populations of neurons. A commonly used form is the logistic function,

$$f(\xi) = (1+e^{-\xi})^{-1}.$$

The constant I_i represents an external input bias which may be included inside or outside $f(\xi)$. I chose the latter case arbitrarily. The fixed points of (1), which I denote as $\mathbf{x}^0$, are solutions of the nonlinear algebraic equations

$$\alpha x_i^0 = \beta f_i\left(\sum_j w_{ij}x_j^0\right) + I_i \tag{2}$$

and are implicit functions of the weight matrix $\mathbf{w}$ and initial state $\mathbf{x}^i$.

Suppose that $\mathbf{w}$ is lower triangular. Then it is clear that Eq. (2) can be solved recursively since to calculate x_i one needs only $x_1, \ldots, x_{i-1}$. Thus, when the units are properly labeled, this is just the forward propagation which occurs in the widely used feedforward network of Rumelhart, Hinton, and Williams.[3] I conclude that the feedforward network simply provides a direct method of calculating the fixed points of (1) when $\mathbf{w}$ is lower triangular.

The δ rule is a learning rule for feedforward networks. Strictly speaking, it is restricted to feedforward networks only. Nevertheless it has been applied to recurrent networks by taking advantage of the fact that for every recurrent network there exists an equivalent feedforward network (for a finite time). The cost for this strategy is

the manyfold duplication of the hardware for the feedforward version of the recurrent network.[3] The algorithm presented in this paper makes unnecessary the artifact of unfolding a recurrent network into a feedforward network.

A necessary condition for the learning algorithm discussed in this Letter to exist is that system (1) reach steady state (I will not discuss limit cycles here). Except for some theorems concerning collective quantities,[4] little is known about the stability of system (1) for arbitrary $\mathbf{w}$. However, there are special cases for which (1) can be proved to be globally asymptotically stable. The set of equations (1) is stable if $\mathbf{w}$ is symmetric because (1) can be transformed into the equations studied by Hopfield[5] under the coordinate transformation,

$$u_i = \sum_k w_{ik} x_k.$$

Hopfield's equations are globally asymptotically stable if $\mathbf{w}$ is symmetric and has zeros along the diagonal. Stability in this case is proved because there exists a Liapunov function. A general theorem concerning stability of networks with symmetric weights is given by Cohen and Grossberg.[6] The set of equations (1) is also globally asymptotically stable if $\mathbf{w}$ is lower triangular because in such a case the network is a pure feedforward network. In other words the nth unit can only receive input from the mth unit if $n > m$. The stability of the feedforward case follows from a recursive agreement which goes as follows. Suppose that the activations x_i (where $i = 1, \ldots, m$) are constant. Then from the feedforward constraint the nth unit (where $n = m + 1$) receives only constant input. With constant input Eqs. (1) converge exponentially to a constant value, and hence x_{m+1} becomes constant. Thus it is clear that if the inputs are constant, the activation of the entire network will ultimately become constant. Equations (1) are also stable in the limit of infinite $\mathbf{w}$ since if $\mathbf{w}$ is infinite the function $f(u_i)$ becomes constant and the solutions to (1) simply decay exponentially to constants.

Numerical simulations conducted by this author strongly suggest that in practice the system is stable for most $\mathbf{w}$ and initial $\mathbf{x}$. Oscillatory solutions can occur when there exists substantial self-excitation. It shall be assumed, for the purpose of deriving the back-propagation equations, that the system ultimately settles down to a stable state. With this caveat in mind I present the recurrent back-propagation (RBP) algorithm.

Consider a system of neurons, or units, whose dynamics is determined by Eqs. (1). Of all the units in the network we will arbitrarily define some subset of them, A, as *input* units and some other subset of them, Ω, as *output* units. Units which are neither members of A or Ω are denoted *hidden* units. A unit may be simultaneously an input unit and an output unit. If a unit is an input unit, the corresponding component of the vector $\mathbf{I}$ is nonzero and represents an external input to the system, i.e.,

$$I_i = \begin{cases} \xi_i, & \text{if } i \in A, \\ 0, & \text{otherwise,} \end{cases}$$

where ξ_i is an external input.

Our goal will be to find a local algorithm which adjusts $\mathbf{w}$ so that a given fixed initial state $\mathbf{x}^i$ and a given set of input values ξ_i result in a fixed point, $\mathbf{x}^0$, whose components along the output units have a desired set of values, τ_j (where $j \in \Omega$). This will be accomplished by our minimizing a function which measures the error between the desired fixed point and the actual fixed point. Consider the positive definite function

$$E(\mathbf{x}^0) = \tfrac{1}{2} \sum_{i=1}^{N} J_i^2,$$

where

$$J_i = \begin{cases} \tau_i - x_i^0, & \text{if } i \in \Omega, \\ 0, & \text{otherwise.} \end{cases}$$

It is an implicit function of the weight matrix $\mathbf{w}$ because the fixed point $\mathbf{x}^0$ is implicitly dependent on the weight matrix. $E(\mathbf{x}^0)$ has a family of minima which exist on the hyperplanes which satisfy $x_i^0 = \tau_i$, where $i \in \Omega$.

A formal learning algorithm consists of an algorithm which drives the fixed point towards one of these hyperplanes. Dynamically, this is accomplished by our letting the system evolve in the weight space along trajectories which are antiparallel to the gradient $\partial E/\partial w_{ij}$. In other words,

$$dw_{ij}/dt = -\eta\, \partial E/\partial w_{ij}, \tag{3}$$

where η is a numerical constant which defines the (slow) time scale on which $\mathbf{w}$ changes. *η must be small so that x is always essentially at steady state* (i.e., $\mathbf{x} \cong \mathbf{x}^0$). On performing the differentiations in (3) one immediately obtains

$$dw_{rs}/dt = \eta \sum_k J_k\, \partial x_k^0/\partial w_{rs}. \tag{4}$$

The derivative of x_k^0 with respect to w_{rs} is obtained by our differentiating both sides of (2) with respect to w_{rs} and solving for the derivatives. The result is

$$\partial x_k^0/\partial w_{rs} = \beta(\mathbf{L}^{-1})_{kr} f_r'(u_r) x_s^0, \tag{5}$$

where the matrix $\mathbf{L}$ is given by

$$L_{ij} = \alpha\delta_{ij} - \beta f_i'(u_i) w_{ij},$$

and where δ_{ij} is the Kronecker δ symbol. On substituting (5) into (4) one immediately obtains

$$dw_{rs}/dt = \eta y_r x_s^0, \tag{6}$$

where

$$y_r = \beta f_r'(u_r) \sum_k J_k (\mathbf{L}^{-1})_{kr}. \tag{7}$$

Equations (6) and (7) specify a formal learning rule. Equations (7) require a matrix inversion to calculate the error signals, y_k. Direct matrix inversions are necessarily nonlocal calculations and therefore this learning algorithm is not suitable for implementation as a neural network. A local method for the calculation of y_r is obtained by the introduction of an associated dynamical system. Consider the vector $\mathbf{z}$ whose components are defined in terms of the components of $\mathbf{y}$ according to

$$y_r = \beta f_r'(u_r) z_r. \tag{8}$$

Equations (7) and (8) imply that z_r satisfies

$$\sum_r L_{ri} z_r = J_i. \tag{9}$$

Now observe that the solutions of Eqs. (9) are the steady-state solutions of

$$dz_i/dt = -\sum_r L_{ri} z_r + J_i.$$

In terms of the explicit variables in the problem, these equations are

$$dz_i/dt = -\alpha z_i + \beta \sum_r \{f_r'(u_r) w_{ri} z_r\} + J_i. \tag{10}$$

This leads to a learning rule of the form

$$dw_{rs}/dt = \eta f'(u_r) z_r^0 x_s^0. \tag{11}$$

Equations (1), (10), and (11) completely specify the dynamics for an adaptive neural network, provided that (1) and (10) are convergent. It is known that the convergence of (1) is a sufficient condition for the convergence of (10).[7] This follows from the observation that the back-propagation network is obtained from the forward-propagation network (linearized about a fixed point) and that a linear network is stable in both directions if it is stable in either direction. It is quite easy for one to obtain the δ rule from the RBP algorithm by expressing Eqs. (1), (10), and (11) as difference equations with $\Delta t = 1$ and with $\mathbf{w}$ lower triangular.

I have conducted preliminary numerical experiments with exclusive OR (XOR) networks to verify the correctness of the algorithm. These were performed by my approximating the differential equations with first-order finite-difference equations and requiring that Eqs. (1) and (10) converge before taking an integration step in Eq. (11). The XOR network is shown in Fig. 1. Each input unit receives one digit of a two-digit binary number. The target x_i for the output unit is 1 if the number of 1's in the input is odd and 0 otherwise. Unit 5 is a threshold unit, i.e., it biases the total input to units 3 and 4 so as to provide a threshold which must be exceeded if these units are to turn on. Unit 5 feeds back on itself so as to stay turned on always. The feedforward exclusive OR network used by Rumelhart, Hinton, and Williams is completely equivalent to this network if the backward connection from unit 4 to unit 3 is omitted and if the feedback loop in unit 5 has an infinite positive magnitude. In practice I made the magnitude of the loop merely large and was able to reproduce the behavior of the Rumelhart network.

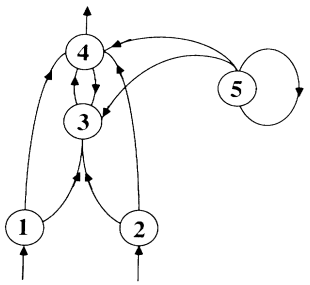

FIG. 1. XOR network with recurrent connections.

The network with the backward connection performed only modestly faster than the network without this connection.[8] The main difference in the networks was in the distribution of final weights. Both networks had similar attractors which could be characterized by the final root mean square weight per connection ($w_{\rm rms}$). The attractor which I denote by A had $w_{\rm rms} = 3.4$, while the attractor denoted by B had $w_{\rm rms} = 10.7$. The network without the backward connection converged onto attractors A and B approximately 85% and 15% of the trials, respectively, whereas the network with the backward connection converged onto attractors A and B approximately 52% and 48% of the trials, respectively. Only in one trial out of 480 did the recurrent network fail to converge onto a global minimum. Each pattern was presented to the recurrent network approximately 200 times. The final solutions were insensitive to the initial value of $\mathbf{x}$ which indicates that the attractors of Eqs. (1) have large basins of convergence.

It is worthwhile to compare the RBP network with the master/slave network of Lapedes and Farber.[9] The slave network corresponds to my forward-propagation network. If we suppose it has N nodes then the master network determines the weights of the slave network by integrating N^2 equations, each of which has a form similar to Eqs. (1), but with slave weight matrix elements as dynamical variables and a rank-4 matrix as the master's weight matrix. The weight matrix of the master network has a simple symmetric form with at most $N(N+1)/2$ nonvanishing independent components. These components require additional storage beyond the N^2 components of the slave's weight matrix. The RBP network, on the other hand, requires the integration of N^2+2N equations and no additional storage. $2N$ of these equations correspond to Eqs. (1) and (10). The remaining N^2 equations have a simple outer product form [cf. Eq. (11)] and are quite trivial to implement. The conclusion is that the RBP network is an architecturally simpler network than the master/slave network and requires less memory.

The master/slave network directly minimizes the average of E over all input/output associations. This average is denoted by $\langle E \rangle$. The master equations are guaranteed to converge to at least a local minimum of $\langle E \rangle$ because $\langle E \rangle$ is a Liapunov function for the equations.[10] On the other hand, E is a Liapunov function for Eq. (3) of the RBP network only in the case of a single input/output association. For multiple associations the RBP network is guaranteed to converge only in a probabilistic sense and under certain technical conditions. It was noted by Amari[2] that gradient-descent algorithms, such as RBP, converge to a minimum point of $\langle E \rangle$ to within a small fluctuating term provided that the input/output sequence is an ergodic random sequence and provided that $\langle E \rangle$ has a unique minimum. Experimentally it is found that RBP, like standard back-propagation, converges robustly albeit after very many iterations. A detailed computational comparison of RBP and master/slave has yet to be performed.

The RBP algorithm is better suited for hardware implementation than the δ rule for two reasons. First, the algorithm is expressed completely in differential equations and therefore can be implemented in analog very large-scale integration. This eliminates the timing and synchronization problems which appear in digital implementations of the standard δ rule. Second, the RBP algorithm vectorizes naturally. This is because the units are homogeneous, i.e., the input, hidden, and output units all obey the same differential (difference) equations—only the components of the constant vectors **I** and **J** serve to distinguish the roles of the units.

The author wishes to acknowledge very fruitful discussions with Robert Jenkins and Ben Yuhas. Liam Healy also contributed in the early discussions. This work was supported in part by Grant No. AFOSR-87-0354 from the U.S. Air Force Office of Scientific Research.

[1]M. Minsky and S. Papert, *Perceptron* (M.I.T. Press, Cambridge, MA, 1969).

[2]Shun-Ichi Amari, in *Systems Neuroscience,* edited by Jacqueline Metzler (Academic, New York, 1977).

[3]D. E. Rumelhart, G. E. Hinton, and R. J. Williams, in *Parallel Distributed Processing,* edited by D. E. Rumelhart and J. L. McClelland (M.I.T. Press, Cambridge, MA, 1986).

[4]Shun-Ichi Amari, IEEE Trans. Systems Man Cybernet. **2**, 643–657 (1972).

[5]J. J. Hopfield, Proc. Natl. Acad. Sci. U.S.A. Bio. **81**, 3088–3092 (1984).

[6]Michael A. Cohen and Stephen Grossberg, IEEE Trans. Systems Man Cybernet. **13**, 815–826, 1983.

[7]After this Letter was submitted, the author learned of the independent work of Luis B. Almeida, who derived a discrete version of the RBP algorithm to appear in the Proceedings of the IEEE First Annual International Conference on Neural Networks, San Diego, California, June 1987, edited by M. Caudil and C. Butler (to be published).

[8]The parameters used were $\Delta t = 1$, $\alpha = 1$, $\beta = 1$, and $\eta = 0.5$. The integration was terminated when the χ^2 summed over all four patterns reached 0.1. A new pattern was presented on each iteration of Eq. (11). The difference equation corresponding to (11) was modified by the inclusion of a momentum term to accelerate the convergence.

[9]Alan Lapedes and Robert Farber, Physica (Amsterdam) **D22**, 247–259, 1986, and in *Neural Networks for Computing—1986,* edited by J. S. Denker, AIP Conference Proceedings No. 151 (American Institute of Physics, New York, 1986), pp. 283–298.

[10]The master/slave Liapunov function actually contains an extra term which adds a passive decay term to the learning equations. Our function E could be modified to include such a term, but the inclusion of such a term is not essential to the discussion.