JOURNAL OF COMPLEXITY 4, 216–245 (1988)

Dynamics and Architecture for Neural Computation*

FERNANDO J. PINEDA

Applied Physics Laboratory, Johns Hopkins University, Johns Hopkins Road, Laurel, Maryland 20707

Received April, 1987

Useful computation can be performed by systematically exploiting the phenomenology of nonlinear dynamical systems. Two dynamical phenomena are isolated into primitive architectural components which perform the operations of continuous nonlinear transformation and autoassociative recall. Backpropagation techniques for programming the architectural components are presented in a formalism appropriate for a collective nonlinear dynamical system. It is shown that conventional recurrent backpropagation is not capable of storing multiple patterns in an associative memory which starts out with an insufficient number of point attractors. It is shown that a modified algorithm can solve this problem by introducing new attractors near the to-be-stored patterns. Two primitive components are assembled into an elementary machine and trained to perform invariant pattern recognition with respect to small arbitrary transformations of the input pattern, provided the transformations are sufficiently small. The machine realizes modular learning since error signals do not propagate across the boundaries of the components.

1. INTRODUCTION

Much of the recent interest in neural computation stems from the suggestion by Hopfield (1982) that the collective properties of physical systems might be used to directly implement computational tasks. This new paradigm for computation promises to yield a new class of computing machines in which the physics of the machine and the algorithms of the computation are intimately related.

The purpose of this paper is to show how to perform useful computation by systematically exploiting the phenomenology of a class of collec-

* This work was supported in part by the Air Force Office of Scientific Research under Grant AFOSR-87-0354 and by the Applied Physics Laboratory under IRAD-X8U.

0885-064X/88 $3.00

tive dynamical systems. Initially the model-independent behavior of the dynamical systems will be discussed and it will be shown how the phenomenology of the systems can be isolated into two primitive architectural components (filters) which perform the operations of continuous nonlinear transformation and autoassociative recall. These filters are primitive in the sense that they are fundamental building blocks from which one can build hierarchical architectures.

Backpropagation techniques for programming filters will be developed for the case of a simple model, however, the techniques apply to a broad class of neurodynamical models. The recurrent backpropagation algorithms will be presented in a formalism appropriate for implementation as a physical nonlinear dynamical system. One of the advantages of this formalism is that it uses continuous time and therefore does not exhibit certain kinds of oscillations which occur in discrete time models usually associated with backpropagation.

One of the results of the model-independent investigation will be applied to explain why the backpropagation algorithm is incapable of storing multiple patterns in a simple associative memory model. The solution to the problem will be to constrain the system during learning. The resulting algorithm not only changes the location of fixed points, it also creates new ones. This results in discontinuous learning behavior in the autoassociative memory.

As a demonstration of a simple hierarchical architecture, two primitive filters will be combined to produce an elementary pattern recognition machine. This machine is capable of recognizing patterns which have been corrupted by arbitrary transformations provided the transformations are sufficiently small. In other words the machine exhibits a limited amount of invariant pattern recognition. The two filters in the machine are capable of learning independently in the sense that error signals do not propagate across the filter boundaries. Thus the two-filter system is a simple example of the modular learning scheme proposed by Ballard (1987).

This paper is organized in the following way. In Section 2, a restricted definition of neurodynamics is given. The systems considered in this paper are subclasses of this dynamics. The way in which the dynamics is exploited to construct two kinds of filters is explained. In Section 3 the behavior of these two filters is discussed qualitatively. In Section 4 a specific neural model is chosen and discussed. This model provides a concrete system for illustrating the subsequent developments. Section 5 contains a derivation of a set of dynamical equations which are appropriate for training the model when it is used to make continuous nonlinear maps. Section 6 discusses how these equations are used to learn multiple input/output patterns. Section 7 discusses the role of time scales in the

learning dynamics. In Section 8 the dynamical equations for training an associative memory are presented and discussed. Section 9 presents a simple hierarchically organized pattern recognition system based on the components discussed in the previous sections. Finally, the results are summarized and discussed in Section 10.

2. Neurodynamics and Primitive Filters

A universally agreed-upon definition of neurodynamics does not exist, but for the purposes of analysis it is useful to define the most general features of the dynamical systems which are to be considered in this paper. The entire discussion, unless otherwise specified, will be limited to systems which have continuous-valued states and equations of motion which can be expressed as differential equations. These systems possess three general characteristics. First, they generally have very many degrees of freedom. The human brain, for example, is believed to have between 10^{11} and 10^{13} neurons (depending on which cells are counted). The state of each of these neurons can be modeled by one or more dynamical variables. It is generally believed that the computational power and fault-tolerant capabilities of neural systems results from the collective dynamics of the system. Collective effects account for the properties of many physical systems including magnetism, superconductivity, and fluid dynamics. These systems are trivial in one respect. They can all be characterized by only one or two coupling constants. Neurodynamical systems on the other hand are characterized by very many coupling constants. In general, there is a different coupling constant for each interaction. In biological systems these different coupling constants correspond to the strengths of individual synaptic junctions. A well-studied physical system which does have very many coupling constants is the spin-glass (see, e.g., Binder and Young, 1986). Not surprisingly, this system has been used as the basis for discrete neural network models, e.g., Hopfield (1982) and Hinton *et al.* (1984).

Second, the neurodynamical systems are nonlinear. Linear dynamical systems are characterized by the fact that any two solutions of the system may be added together to produce a third solution. Accordingly, linear dynamical systems can perform linear mappings only and are therefore limited in their computational ability. This is one of the implications of a rigorous analysis by Minsky and Papert (1969) of single layer networks of linear threshold units called perceptrons. Nonlinearity is a required property in associative memories if they are to distinguish between two stored patterns. This issue is discussed further in Section 9 of this paper.

The final characteristic of the neurodynamical systems is that they are

dissipative. Dissipative dynamical systems are generally described by coupled sets of first-order differential equations of the form

$$dx_i/dt = G_i(\mathbf{x}). \tag{2.1}$$

If $\mathbf{G}$ and $\mathbf{x}$ have N components then the state space of the system is N-dimensional and the trajectory of $\mathbf{x}$ is called an N-dimensional flow. A dissipative system is characterized by the convergence of the flow onto a manifold of lower dimensionality as the system evolves. General dissipative systems can exhibit complicated behavior. For example, they may converge onto one-dimensional manifolds (periodic orbits) or manifolds with fractional dimensions (strange attractors). The discussion in this paper will be confined to systems whose only behavior is to converge onto point attractors for some range of parameters and initial conditions. Point attractors are important in neural computing because the corresponding state vector values can be used to represent computational objects, e.g., memories, data structures, or rules.

Now consider a general neurodynamical system with an unspecified dependence on internal dynamical parameters, external dynamical parameters, and state variables. The system is defined by the equation

$$dx_i/dt = G_i(\mathbf{w}, \mathbf{I}, \mathbf{x}). \tag{2.2}$$

The matrix $\mathbf{w}$ represents the set of internal dynamical parameters and the vector $\mathbf{I}$ represents the set of external dynamical parameters, i.e., a control vector or external bias. It will be assumed that trajectories of this system converge onto point attractors for values of $\mathbf{w}$, $\mathbf{I}$ and initial states $\mathbf{x}^0$ in some "operating region." The concept of an operating region of the system will be taken to mean the set of $\mathbf{x}$, $\mathbf{w}$, and $\mathbf{I}$ which are permitted by the dynamics of the system, the dynamics of the learning algorithm, or the dynamics of the external environment, respectively. This concept is not sharply defined, nevertheless it is useful for describing the phenomenology of general neurodynamical systems.

Quantities evaluated at steady state will be denoted by a superscript $^\infty$. In particular the point attractors will be denoted by $\mathbf{x}^\infty$. These are solutions of

$$0 = G_i(\mathbf{w}, \mathbf{I}, \mathbf{x}^\infty). \tag{2.3}$$

For a given $\mathbf{w}$ and $\mathbf{I}$, the set of initial points $\mathbf{x}^0$ that evolve to a particular fixed point is called the basin of attraction of that fixed point. The locations of the fixed points and the basin boundaries are functions of $\mathbf{w}$ and $\mathbf{I}$.

The phenomenology of the general neurodynamical system described by Eq. (2.2) can be exploited to construct filters. The term "filter" refers

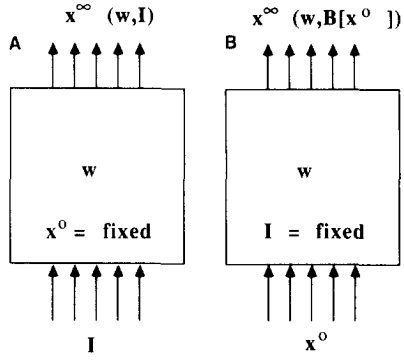

FIG. 1. For a continuous mapper (A) the filter input is the external dynamical parameter **I**. The output $\mathbf{x}^{\infty}(\mathbf{w}, \mathbf{I})$ is a continuous function of **I**. For an autoassociative memory (B) the filter input is the initial state $\mathbf{x}^0$. The output $\mathbf{x}^{\infty}(\mathbf{w}, B[\mathbf{x}^0])$ changes depending on which basin (denoted by B) contains the input state.

to an architectural component that performs a mapping operation from some input to some output. The architecture of a computer, the design of a program, or the organization of the brain can be described as a hierarchy of suitably designed filters. Adaptive filters, which will be discussed when learning algorithms are introduced, change their mapping operation according to the history of inputs.

If one restricts the discussion to filters with static outputs, there are two general ways of exploiting the dynamics of system (2.2) to obtain a filter. In both cases the final state $\mathbf{x}^{\infty}$ of the system is used as the output of the filter. In the first case, which is shown schematically in Fig. 1A, the bias **I** acts as the input to the filter. The initial state is set to some constant vector for all inputs. In the second case, which is shown schematically in Fig. 1B, the initial state $\mathbf{x}^0$ of the dynamical system represents the input to the filter and it is the bias **I** which is set to some constant vector for all inputs.

Two well-known neural network models are examples of these two filters. The Hopfield (1984) associative memory with analog neurons is an example of the second filter. In a Hopfield network, information is stored by locating point attractors at positions in the state space which correspond to memories. The system typically converges to a complete memory if an incomplete memory, e.g., a state vector in which only some of the components are the same as a stored memory, is presented as an initial state. In general the output of such a filter is a discontinuous function of the input. For the remainder of this paper a filter which performs autoassociative recall will be called an autoassociative memory or associative memory for short. On the other hand, the feedforward network used by Rumelhart *et al.* (1986) is a limiting case of the first filter. In this

case the bias **I** represents input into the bottom layer of the feedforward network. In general this second kind of filter will behave continuously provided that certain phenomena do not occur. These phenomena will be discussed in the following section. For the remainder of the paper this kind of filter will be called a continuous mapper or mapper for short.

The question of whether a given dynamical system can realize a given mapping is an important unanswered question. In the language of dynamics the question is whether the desired attractors are in the region of state space accessible to the system. Or, put another way, do the coupling constants exist which code the representation? This issue is beyond the scope of this paper and will not be addressed here. The approach to be followed is pragmatic: it will be assumed that a sufficiently large neurodynamical system with feedback loops is capable of representing the maps which are of interest.

Let us turn now to a discussion of how the dynamics of system (2.2) leads to either continuous or discontinuous behavior. A clear understanding of this behavior is important for the proper design of filters and learning algorithms. In the case of learning algorithms continuity is an important consideration because any learning algorithm that gradually adjusts the internal dynamical parameters is relying on the continuity of the state vector.

3. A Qualitative Description of Filter Behavior

The qualitative behavior of the continuous mapper and the autoassociative memory can be deduced from simple considerations. The presentation in this section is a schematic description of this behavior in the sense that many aspects of nonlinear dynamical systems will be simplified in the interests of clarity. For example, basins are often disconnected and have fractal boundaries. Accounting for these complications is not essential for a description of the basic phenomena. Therefore these details will be neglected. Recall that it has been assumed that the only permitted behavior of the solution of Eq. (2.2) is convergence onto point attractors, provided that the system stays within some operating region. Let us now consider the motion of these point attractors.

In a continuous mapper the location of the fixed point is usually a continuous function of **w** and **I**. The mapper will behave continuously provided that the point attractor does not vanish and provided that a basin boundary does not go past the initial point. The former situation is caused by a topological transition called a catastrophe (Arnold, 1986) which results in a discontinuous change in the attractor and basin structure (see, e.g., Amari, 1977). On the other hand, the latter situation involves no

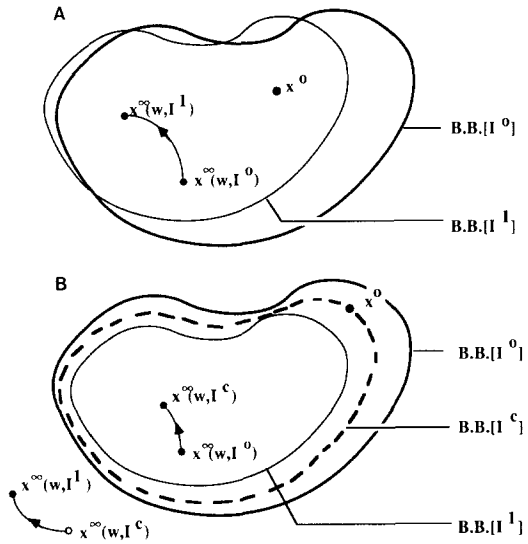

FIG. 2. If no catastrophes occur, a continuous mapper behaves continuously provided that the initial state is always inside the basin boundary (denoted by B.B.) for all values of **I**. This is shown in (A). On the other hand (B) shows that if the basin boundary crosses the location of the initial point, the behavior is discontinuous.

topological transition and results from the particular choice of initial state. To illustrate the latter situation consider a sequence of trials wherein one tracks the final state for a set of gradually changing inputs. The initial state is reset between trials. Figure 2A shows the schematic behavior of the final state and the basin boundary. Let the interior of the boundary represent one basin and the exterior represent another basin. If **w** is fixed and **I** changes continuously from $\mathbf{I}^0$ to $\mathbf{I}^1$, the fixed point moves continuously from $\mathbf{x}^\infty(\mathbf{w}, \mathbf{I}^0)$ to $\mathbf{x}^\infty(\mathbf{w}, \mathbf{I}^1)$. The boundary moves also, but at no time does the boundary cross the initial point $\mathbf{x}^0$. Thus $\mathbf{x}^\infty$ depends continuously on **I**. On the other hand consider Fig. 2B. In this case the fixed point moves continuously until the **I** reaches the value $\mathbf{I}^c$. As **I** goes through the value $\mathbf{I}^c$ the basin boundary goes past the initial state $\mathbf{x}^0$ and the initial point is suddenly in the exterior basin. Accordingly the steady state solution jumps discontinuously to the point attractor of this basin. One can turn the argument around and consider changing $\mathbf{x}^0$ while keeping **I** fixed. This is the case to consider if one is presenting multiple inputs. Suppose one examines $\mathbf{x}^\infty$ only whenever a specific input, say $\mathbf{I}^\alpha$, is presented to the system. Furthermore, suppose the "initial" state is not reset for each new pattern but instead it is taken to be the steady state from the previously presented pattern. Then, if a catastrophe does not occur, the final state for

pattern $\mathbf{I}^{\alpha}$ will be unique in the operating region, provided that none of the final states for the other patterns is outside the basin boundary for $\mathbf{I}^{\alpha}$.

Similar arguments apply when one considers the qualitative behavior of the continuous mapper in the learning process. The only difference is that **w** is varied gradually. To be more precise it changes slowly compared to the relaxation time of **x**. Therefore the instantaneous **x** can be approximated by the steady state solution of Eq. (2.2). In physics this is known as the adiabatic approximation. Suppose one is training the system on a single pattern by changing **w** adiabatically. Then **I** is a constant for all time the system is always in steady state. By definition, the fixed point is always inside its basin so the only way a discontinuity can occur is for a catastrophe to occur. Barring such an occurrence the steady state solution will move gradually and continuously toward the desired final state. Now suppose one is training the system on multiple patterns so that **I** is no longer a constant for all time but instead makes transitions between some set of input vectors. If the transitions occur adiabatically then only discontinuities due to catastrophes can occur. If, on the other hand, the transitions occur suddenly, as is the case in most neural network simulations, then discontinuities due to basin boundaries crossing "initial" states can occur, where the "initial" state is the steady state solution for the previously presented input. The general conclusion is that for the mapper to have continuous outputs and for learning to take place, the accessible states of the system should be in a region of the state space which has no basin boundaries for all **w** and **I** which will be encountered. This is the case if the point attractor is unique for fixed **w** and **I**.

Associative memories have completely different requirements. **I** is a fixed constant and for the purposes of this discussion it can be taken to be zero without loss of generality. Accordingly the point attractors and basin boundaries depend on **w** alone. An associative memory relies on having very many basins distributed over the operating region of the state space. Each of the basins is associated with a memory. If the untrained associative memory does not have enough attractors it may be difficult to store memories. To see this consider Fig. 3. Suppose one has a learning algorithm which moves attractors by adiabatically changing the internal parameters **w** in response to some measure of the separation between the location of the desired memory and the point attractor. Then, if there is more than one to-be-stored memory within a basin of the untrained system, the learning algorithm may not be able to store both memories because they both converge to the same final state. To store the memories the learning algorithm must somehow get the two memories into different basins by either going through a catastrophe or by a basin boundary going across a to-be-stored memory. The latter cannot be guaranteed to occur in a learning algorithm which responds to the location of a point attractor

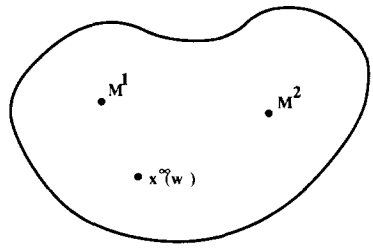

FIG. 3. Two memories, $\mathbf{M}^1$ and $\mathbf{M}^2$ are in the basin of a single point attractor $\mathbf{X}^\infty(\mathbf{w})$. This point attractor cannot be moved to two locations by adjusting $\mathbf{w}$. Either a catastrophe must occur so that the memories end up in the basins of different point attractors or else the basin boundary must go across one of the memories and thereby place the memory in a different basin.

only, because such a learning algorithm does not have direct control over the basin boundaries.

The general conclusion of this analysis is that continuous mappers and associative memories have conflicting requirements. For fixed $\mathbf{I}$ and $\mathbf{w}$, the continuous mapper requires a unique point attractor in the operating region whereas the associative memory requires multiple point attractors. This conflict must be resolved if one is to build and train hierarchical systems with both filters. A specific example of this conflict and a way of circumventing it is given in Section 8.

4. A Simple Nonlinear Neural Model

The discussion now narrows to a specific neural model. The intent is to use a simple model so as not to obscure the subsequent discussions with mathematical complications. The reader should keep in mind that the techniques discussed throughout this paper apply equally well to high-order models (Pineda, 1987b).

A simple nonlinear model for an interacting system of N neurons which ignores propagation time delays between neurons is specified by the system of equations

$$du_i/dt = -u_i + \sum_j w_{ij} g(u_j) + I_i, \tag{4.1}$$

where $\mathbf{u}$ is the state vector of the system. The summation convention in this and all other equations in the paper is that the sum runs from $j = 1$ to N where N is the number of nodes. The weight-matrix elements w_{ij} repre-

sent the synaptic strengths of the connections between the neurons and I_i represent external biases. A commonly used form for the function g is the logistic function,

$$g(u) = (1 + e^{-u})^{-1}. \tag{4.2}$$

This simple model is well studied and is the basis of many current neural network models. The biological motivation for this model is reviewed briefly by Sejnowski (1981) and an interpretation in terms of electronic components is given by Hopfield (1984).

This author is aware of three useful operating regions for the system given in Eq. (4.1). First, when the weight matrix is lower triangular the network is of the feedforward type. Hence, there are no recurrent loops and it converges to a unique point attractor for all initial conditions. Second, when the weight matrix is symmetric, the system possess a Liapunov function and it must converge to one of many point attractors (Hopfield, 1984) and (Cohen, 1983). Finally, Atiya (1987) has shown that the system always converges to a unique point attractor provided that

$$\sum_i \sum_j w_{ij}^2 < 1/(\max_i |g_i'|)^2. \tag{4.3}$$

It is doubtful that these three cases exhaust the possible operating regions. Indeed, arguments given by Geman (1981) suggest that the system will converge "almost always" to a point attractor provided that a chaotic hypothesis holds. The chaotic hypothesis allows the weights and the activations to be treated as independent random variables (Amari, 1972).

5. Learning in a Continuous Mapper

Now let us turn to a specific technique for programming the dynamics of system (4.1). The learning algorithms to be presented are continuous and recurrent generalizations of the backpropagation algorithms discussed by Parker (1982), Le Cun (1985), and Rumelhart *et al.* (1986). Backpropagation is a very useful and general tool for training neural network models with arbitrary connectivity. The literature is now rich with recurrent, higher order, and stochastic variations of the original algorithms, e.g., Almeida (1987), Atiya (1987), Parker (1987), Rohwer and Forrest (1987) and Samad and Harper (1987), to name a few. It is worthwhile to point out that the first-order finite difference approximation of

Eq. (4.1), with $\Delta t = 1$, is the form of the model which is often associated with backpropagation. In fact, the difference equations suffer from oscillations which are artifacts of the discrete time approximation. This is discussed briefly in Appendix C.

An algorithm for training system (4.1) which allowed recurrent connections was first introduced by Lapedes and Farber (1986a, 1986b). This algorithm did not use the backpropagation technique for calculating the gradient. A more efficient algorithm which used backpropagation to reduce the amount of calculation was introduced by Pineda (1987a). A later paper by Pineda (1987b) emphasized that the method for obtaining the algorithm was quite general and applied to an entire class of neural network models, including higher order networks. This formalism is now reviewed.

The adaptive dynamics adjusts the weights $\mathbf{w}$ of the dynamical system (4.1) when it is used as a continuous mapper. Rather than using the form (4.1) it has been found convenient to transform these equations into the form

$$dx_i/dt = -x_i + g_i\left(\sum_j w_{ij}x_j + I_i\right). \tag{5.1}$$

Equation (4.1) is transformed into (5.1) by an affine transformation of the form $\mathbf{u} = \mathbf{wx} + \mathbf{I}$. If $\mathbf{w}$ is nonsingular the transformation is invertible and the attractor structure of Eq. (4.1) is the same as Eq. (5.1). If g_i is the logistic function, x_i is bounded between $0 \leq x_i \leq 1$ and the operation region is containing within the unit N-cube.

To simplify the subsequent mathematical manipulations it is useful to introduce a notational convention. Suppose that Φ represents some subset of units in the network. Then the set function $\Theta_{i\Phi}$ is defined by

$$\Theta_{i\Phi} = \begin{cases} 1 & \text{if } i\text{th unit is a member of } \Phi \\ 0 & \text{otherwise.} \end{cases} \tag{5.2}$$

In the continuous mapper there are three important subsets of units. The first is the subset (A) of input units. The second is the subset (Ω) of output units. Note that a unit can be simultaneously an input unit and an output unit. Finally, the set (H) of hidden units contain all the units which are neither members of A nor Ω.

The external environment influences the system through the bias term, $\mathbf{I}$. If the ith unit is an input unit then $I_i = \xi_i$, otherwise $I_i = 0$, where ξ_i is the value of the external input. This is expressed concisely by the following relation

$$I_i = \xi_i \Theta_{iA}. \tag{5.3}$$

Initially it will be assumed that the inputs ξ_i and outputs η_i are constants in time, thus only a single input/output association (or pattern) is considered. The goal will be to find a local algorithm which adjusts the weight matrix $\mathbf{w}$ so that a given initial state $\mathbf{x}^0$ and a given set of inputs ξ_i result in a point attractor, the components of which have a desired set of values η_i on the output units, i.e.,

$$\eta_i = x_i^{\infty} \Theta_{i\Omega} \tag{5.4}$$

along the output units. This will be accomplished by minimizing a function E which measures the Euclidean distance between the desired position of the point attractor and the actual position of the point attractor. This function is

$$E = \frac{1}{2} \sum_i J_i^2, \tag{5.5}$$

where

$$J_i = (\eta_i - x_i^{\infty}) \Theta_{i\Omega}. \tag{5.6}$$

The function E depends on the weight matrix $\mathbf{w}$ through the fixed point $\mathbf{x}^{\infty}(\mathbf{w})$. A dynamical way of minimizing E is to let the system evolve in the weight space along a "learning trajectory" which descends against the gradient of E. Thus the equation of motion for a weight matrix element w_{rs} is

$$\tau_w dw_{rs}/dt = -\frac{\partial E}{\partial w_{rs}}, \tag{5.7}$$

where τ_w is a numerical constant which defines the (slow) time scale over which $\mathbf{w}$ changes. τ_w must be large so that the weights change adiabatically. If this condition is not satisfied then E is not a function of the point attractor.

It is important to stress that the system evolves both in the space of activations (state space) and in the space of weights (weight space or parameter space). The evolution in the state space is determined by Eq. (5.1) whereas the evolution in the parameter space is determined by Eq. (5.7).

The choice of Eq. (5.7) for the learning dynamics is by no means unique, e.g., Lapedes and Farber (1986b). Other learning dynamics which

employ second-order time derivatives, e.g., the momentum method (Rumelhart *et al.*, 1986) or which employ second-order space derivatives (Parker, 1987) may be more useful in particular applications or hardware implementations. Equation (5.7) does have the virtue of having the simplest functional form which minimizes E by use of a gradient.

The remainder of this section discusses the derivation of a dynamical neural algorithm for efficiently calculating the gradient. Begin by performing the differentiations in Eq. (5.7). One immediately obtains

$$\tau_w dw_{rs}/dt = \sum_k J_k \frac{\partial x_k^\infty}{\partial w_{rs}}. \tag{5.8}$$

The derivative of x_k^∞ with respect to w_{rs} is obtained by first noting that $\mathbf{x}^\infty$ is a fixed point of Eq. (5.1) and hence the kth component must satisfy the nonlinear algebraic equation

$$x_i^\infty = g_i\left(\sum_j w_{ij} x_j^\infty + I_i\right). \tag{5.9}$$

Upon differentiating both sides of this equation with respect to w_{rs} and solving for $\partial x_k^\infty/\partial w_{rs}$ one obtains

$$\frac{\partial x_k^\infty}{\partial w_{rs}} = (\mathbf{L}^{-1})_{kr} g_r'(u_r^\infty) x_s^\infty, \tag{5.10}$$

where

$$u_r = \sum_j w_{rj} x_j + I_r. \tag{5.11}$$

The details of the derivation of Eq. (5.10) are given in Appendix A. The function g_r' is the derivative of g_r and the elements of the matrix $\mathbf{L}$ are given by

$$L_{ij} = \delta_{ij} - g_i'(u_i^\infty) w_{ij}, \tag{5.12}$$

where δ_{ij} are the elements of the identity matrix. On substituting (5.10) into (5.8) one obtains the simple outer product form

$$\tau_w dw_{rs}/dt = y_r^\infty x_s^\infty, \tag{5.13}$$

where y_r^∞ is defined by

$$y_r^\infty = g_r'(u_r^\infty) \sum_k J_k(\mathbf{L}^{-1})_{kr}. \tag{5.14}$$

Equations (5.13) and (5.14) specify a formal learning rule for modifying the weights. Unfortunately, Eq. (5.14) requires a matrix inversion to calculate the "error signals" y_r^∞. 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 calculating y_r^∞ can be obtained by the introduction of an associated dynamical system. To obtain this dynamical system first rewrite Eq. (5.14) as

$$\sum_r \mathbf{L}_{rk}\{y_r^\infty / g_r'(u_r^\infty)\} = J_k. \tag{5.15}$$

Then multiply both sides by $g_k'(u_k^\infty)$ and substitute the explicit form for $\mathbf{L}$. Finally, sum over r. The result is

$$0 = -y_k^\infty + g_k'(u_k^\infty)\left\{\sum_r w_{rk} y_r^\infty + J_k\right\}. \tag{5.16}$$

One now makes the observation that the solutions of this linear equation are the fixed points of the dynamical system given by

$$dy_k/dt = -y_k + g_k'(u_k^\infty)\left\{\sum_r w_{rk} y_r + J_k\right\}. \tag{5.17}$$

The reader familiar with numerical techniques will recognize that this method of obtaining y_k^∞ is little more than a relaxation method for inverting a matrix. It is convenient to borrow the terminology of feedforward networks and refer to Eq. (5.1) as the forward propagation equation and to Eq. (5.17) as the backward propagation equation.

Equations (5.1), (5.13), and (5.17) completely specify the dynamics for an adaptive neural network, provided that (5.17) converges to point attractors. Almeida (1987) has pointed out that the local stability of (5.1) is a sufficient condition for the local stability of (5.17). The proof of this result is given in Appendix B.

The objective function which was chosen is based on Euclidean distance. The most general objective function, however, only has to be separable and bounded from below if it is to lead to local equations. If the function is separable it has the form

$$E = \sum_i e_i(x_i). \tag{5.18}$$

The only change to the adaptive equations is in the definition of J_i which generally has the form

$$J_i = -\partial e_i/\partial x_i. \tag{5.19}$$

A useful objective function based on probability is discussed by Baum, (1987).

Recall that up to this point the entire discussion has assumed that $\boldsymbol{\xi}$ and $\boldsymbol{\eta}$ are constants in time. Thus, no mechanism has been obtained for learning multiple input/output associations. The question of how to learn multiple patterns is addressed in the next section.

6. Learning Multiple Patterns

A method for learning multiple patterns in a gradient descent algorithm was suggested by Amari (1977). His solution was to present the patterns randomly to the network. This corresponds to solving (5.1), (5.13), and (5.17) simultaneously as the patterns change randomly with time. Therefore, the system is subject to a sequence of random forces, each of which attempts to minimize the error for a single pattern. Under certain technical conditions this may result in convergence onto a global minimum. The precise statement is the following. Suppose that each input/output pair is labeled by a pattern label α, i.e., $\{\boldsymbol{\eta}^\alpha, \boldsymbol{\xi}^\alpha\}$. Then define the quantity $E_m(\mathbf{w})$ to be the error $E[\alpha]$ averaged over the distribution of patterns, i.e.,

$$E_m(\mathbf{w}) = \langle E[\mathbf{w}, \boldsymbol{\xi}^\alpha, \boldsymbol{\eta}^\alpha] \rangle. \tag{6.1}$$

If the sequence of random patterns is stationary and if the function $E_m(\mathbf{w})$ has a unique minimum then the theory of stochastic approximation guarantees that the solution of a gradient descent equation will converge to the minimum point $\mathbf{w}_{\min}$ of $E_m(\mathbf{w})$ to within a small fluctuating term which vanishes as $1/\tau_w$ tends to zero.

Random presentation has a built in mechanism for climbing out of local minima of $E_m(\mathbf{w})$ which suggests that it might be able to converge to the global minima of $E_m(\mathbf{w})$. White (1987) has concluded in a recent analysis that the existence of nonunique global minima in neural networks violates one of the assumptions of the convergence proof and therefore the method of random presentation is essentially a method for finding local minima only. It will, however, find the deepest minimum within some (unspecified) neighborhood.

The random presentation of patterns requires that each pattern be presented for a given amount of time. The length of this time influences

whether the patterns are learned or not. This is a special case of the more general issue of the role of time scales in the adaptive network. Let us now turn to a discussion of time scales.

7. Time Scales and Learning

For the system to learn it is necessary for the time scales to be properly chosen. To examine this it is useful to write the dynamical equations with explicit time scales. The equations are

$$\tau_x dx_i/dt = -x_i + g_i(u_i) + I_i(t/\tau_P), \tag{7.1}$$

$$\tau_y dy_k/dt = -y_k + g_k'(u_k)\Big\{\sum_r w_{rk} y_r + J_k(t/\tau_P)\Big\} \tag{7.2}$$

and

$$\tau_w dw_{ij}/dt = y_i x_j. \tag{7.3}$$

The relaxation time scale of the forward propagation is τ_x. The relaxation time scale of the backward propagation is τ_y and the adaptation time scale of the system is τ_w. It is straight forward to establish relationships which must be satisfied by the characteristic time scales of the system. First, note that $y_i x_j$ is the correct form of the gradient only if **x** and **y** are the steady state solutions of Eqs. (7.1) and (7.2). This will hold if the system parameters **w** and **I** change adiabatically. This condition constrains the time scales.

Let τ_P be the characteristic time scale over which the input and output patterns fluctuate. Then, for Eqs. (7.1) and (7.2) to operate near steady state it is necessary that the solution relax with a characteristic time much faster than τ_P, i.e., $\tau_P \gg \tau_x$ and $\tau_P \gg \tau_y$. Also, for Eqs. (7.1) and (7.2) to operate near steady state it is necessary for **w** to change slowly relative to the relaxation times of **x** and **y;** thus one must also have $\tau_w \gg \tau_x$ and $\tau_w \gg \tau_y$. Next, for the flow of Eq. (7.2) to be locally stable it is necessary for the right hand side of Eq. (7.2) to correspond to the transpose of the linearized Eq. (7.1) (see Appendix B). This is guaranteed if $\tau_y \gg \tau_x$.

The final constraint follows from the requirement that the system be able to learn multiple patterns. If the relaxation time of the weights is less than the characteristic time of the pattern fluctuations then the system will trivially learn and then forget each subsequent pattern. To keep the weights from tracking the fluctuations it is necessary that $\tau_w \gg M\tau_P$, where M is the number of patterns. These relationships imply that

$$(\tau_w/M) \gg \tau_P \gg \tau_y \gg \tau_x. \tag{7.4}$$

In practice it is found that these relationships need not be strictly satisfied for the system to learn.

8. Learning in an Autoassociative Memory

Now let us consider how to program autoassociative memory. Recall that a filter which performs autoassociative recall uses the initial state of the system as the input and the final state as the output. Therefore the set of input units and the set of output units are one and the same. This set of units will be called the set of visible units (V). All the other units are hidden (internal) units and the corresponding set is denoted by (H). With this notation the initial state of the system is simply

$$x_i^0 = \eta_i \Theta_{iV} + b_i \Theta_{iH}, \tag{8.1}$$

where the b_i are arbitrary constants and the η_i are components of a memory cue.

Recall also that the input bias $\mathbf{I}$ is an arbitrary constant vector for all patterns. Accordingly it is set equal to zero without loss of generality. Thus the dynamical equations have the form

$$dx_i/dt = -x_i + g_i\left(\sum_k w_{ik} x_k\right). \tag{8.2}$$

The goal of the learning algorithm is to position the point attractors of this dynamical system at the locations of the to-be-stored memories. Thus the point attractors must satisfy

$$\eta_i^\alpha = x_i^{\infty\alpha}(\mathbf{w}) \Theta_{iV} \tag{8.3}$$

for all patterns α. One might think it possible to train the associative memory with essentially the same gradient descent algorithm as in Section 5 except with the modification of resetting the state $\mathbf{x}$ for each memory instead of resetting $\mathbf{I}$. This investigator has never succeeded in training a network this way. The reason for this breakdown is the mechanism described in Section 3. All the initial states are in a single basin and all the trajectories converge onto a single degenerate point attractor. This attractor is a function of $\mathbf{w}$ only. The system output becomes the mean of all the memories since the function being minimized is

$$E(\mathbf{w}) = \frac{1}{2} \sum_\alpha \sum_{i \in V} [\eta_i^\alpha - x_i^\infty(\mathbf{w})]^2, \tag{8.4}$$

where α is a pattern label. The minimum of this function occurs when the point attractor $\mathbf{x}^{\infty}(\mathbf{x})$ equals the average output pattern, e.g., when $\langle \eta_i^{\alpha} \rangle = x_i^{\infty}$. Clearly this is a local minimum of the objective function. The random presentation method discussed in Section 6 might enable the system to climb out of the minimum after a sufficient number of presentations but this cannot be guaranteed. A deterministic algorithm has no mechanism for escape.

This problem can be circumvented by constraining the network during learning. As shall now be shown this leads to a dynamical system with different point attractors for each to-be-stored memory. To make sure that there is no possibility of confusion, the dynamical variables in the constrained system are called **z** rather than **x**. Accordingly consider the constrained system

$$dz_i/dt = -z_i + g_i\left(\sum_k w_{ik} Z_k\right), \tag{8.5}$$

where **Z** is defined to be

$$Z_i = \eta_i \Theta_{iV} + z_i \Theta_{iH}. \tag{8.6}$$

As in Section 5 the discussion will first consider the storage of a single memory which is represented by η_i.

To see that constraining the system has broken the degeneracy of the point attractors substitute Eq. (8.6) into (8.5) to obtain

$$dz_i/dt = -z_i + g_i\left(\sum_{k \in H} w_{ik} z_k + I_i\right), \tag{8.7}$$

where

$$I_i = \sum_{k \in V} w_{ik} \eta_k. \tag{8.8}$$

The point attractors of Eq. (8.7) (if they exist) are again functions of an externally determined bias vector **I**. This breaks the degeneracy of the attractors and it is again possible to train the system on multiple patterns. The effect is to transform an associative memory into a continuous mapper during the training process. As before it will be assumed that the constrained system has only point attractors in its operating region.

Equation (8.7) is useful for training (8.2), because if the weights can be adapted so that the visible units relax to the correct memories, then a fixed point of (8.7) is also a fixed point of (8.2). Therefore, by training the

constrained system one is simultaneously training the unconstrained system. The local stability of a fixed point in the constrained system does not imply the local stability of the same fixed point in the unconstrained system. Nevertheless, in practice the algorithm seems to produce stable fixed points. Recently it has been shown that memories are stored when unstable fixed points become stable (Simard, 1988). This is an issue which needs to be investigated more carefully.

The steps required to derive the new adaptive equations are similar to the steps in Section 5 except that one considers the new objective function

$$E_c = \frac{1}{2} \sum_i J_i^2, \tag{8.9}$$

where

$$J_i = Z_i^\infty - z_i^\infty. \tag{8.10}$$

The mathematical details will be omitted since they are essentially the same as in Section 5. The new gradient descent equation is

$$\tau_w dw_{ij}/dt = y_i^\infty Z_j^\infty, \tag{8.11}$$

where $\mathbf{y}^\infty$ is the steady state solution of

$$dy_k/dt = -y_k + g_k'(v_k^\infty)\left\{\Theta_{kH} \sum_r w_{rk} y_r + J_k\right\} \tag{8.12}$$

and where

$$v_i^\infty = \sum_{k \in H} w_{ik} z_k^\infty + I_i. \tag{8.13}$$

Equations (8.7), (8.11), and (8.12) define the dynamics of the constrained network. The previous discussions concerning the stability of the three Eqs. (5.1), (5.14), and (5.17) apply to Eqs. (8.7), (8.11), and (8.12) as well. The discussions in Sections 6 and 7 concerning multiple patterns and time scales apply to these equations as well.

Once the weights are determined, memory recall is performed by starting the unconstrained network in a state which represents a partial memory cue. The system converges to the previously stored memory whose basin of attraction contains this initial state. It is also conceivable that the system could converge to a spurious memory, i.e., a point attractor that

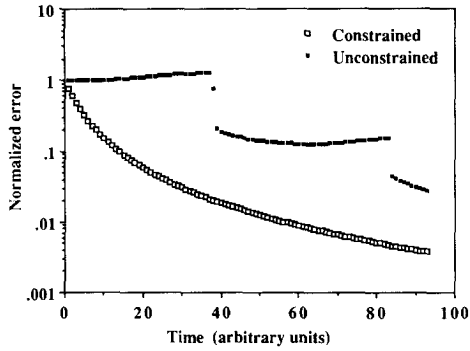

FIG. 4. Typical discontinuous behavior in the associative memory when it is unconstrained and tested at each time step in a learning process. The network contained 10 visible units, 5 hidden units, and 225 connections. The training set consisted of four arbitrarily selected binary vectors with the magnitudes of the vectors adjusted so that $0.1 \leq T_i \leq 0.9$. Learning was performed using the deterministic method in Appendix C which also contains the definition of normalized error.

does not correspond to a stored memory. In practice this occurrence has not been observed by this investigator.

The learning behavior of this associative memory is remarkable and novel. If the associative memory is tested at each time step during the training process by unconstraining it and presenting all the patterns, it is seen that at some time the unconstrained system spontaneously jumps to a point attractor near one of the to-be-stored memories. This occurs for each of the remaining to-be-stored memories in turn, provided that there are not too many of them. Figure 4 illustrates this behavior in a simple case in which four memories are stored. The error function of the unconstrained system undergoes discontinuous jumps as the degeneracy of the single final state is continually broken until there is one final state for each of the four to-be-stored memories. The mechanism for breaking the degeneracy can be (1) a catastrophe in the neighborhood of the to-be-stored memory, i.e., the spontaneous creation of a point attractor, (2) a basin boundary going past a to-be-stored memory, i.e., an already existing attractor is brought into the operating region, or (3) both of the above. The precise mechanism is currently a subject of investigation.

9. A SIMPLE HIERARCHICAL SYSTEM

This section is concerned with an elementary hierarchical system. The primitive components in this hierarchy are the filters whose behavior and programming has been the focus of the above discussions.

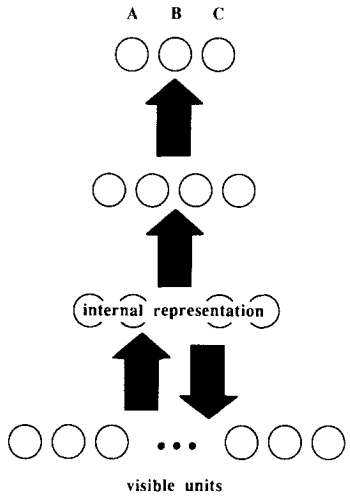

FIG. 5. This shows the topology of the two-filter machine. The layers have 1320 units, 50 units, 4 units, and 3 units, respectively. The first two layers are connected in both directions randomly with approximately 20% sparsity, so there are 25,406 total connections. The remaining layers are fully connected. There are no connections within a layer. The state of each node is determined by an equation of the form (5.1).

Large neural networks based on backpropagation do not scale well, i.e., the convergence time for learning grows faster than the number of layers. Ballard (1987) has suggested that this problem can be overcome by isolating the learning to individual modules. The filters described in this paper can be used to realize this hierarchical approach. The associative memory presented here forms internal representations when provided with hidden units. As suggested by Ballard, these internal representations can be passed to other modules in the hierarchy. In particular suppose the internal representation from an associative memory is passed to a continuous mapper. This mapper can be trained to map this representation to some external representation. The resulting two-filter machine is a primitive pattern recognition device. Figure 5 shows the topology for a four-layer implementation of a two-filter machine. The first layer is the visible layer of an autoassociative memory. The second layer is simultaneously the internal layer of the memory and the input layer of a three layer feedforward mapper. As required by Ballard, error signals are not propagated across the boundary of the two filters. Therefore the associative memory can learn independently of the continuous mapper. Furthermore the continuous mapper can be trained simultaneously with the associative memory or it can be trained after the associative memory has been trained. This network has been trained to recognize the three digitized

FIG. 6. Three images learned by the pattern recognition machine. Each image consists of 55×24 eight-bit pixels. Each image was preprocessed so as to have a mean pixel intensity of 0.5 and a standard deviation of 0.2.

images shown in Fig. 6. (More images can be stored by increasing the connectivity.) Only one of the three output units, denoted A, B, and C, turns on, depending on which image is present.

Despite its simplicity this system is capable of a limited amount of invariant visual pattern recognition, i.e., the output of the trained machine is invariant with respect to small arbitrary transformations of the input image. The invariance is due to the fact that any transformation of a stored pattern can be described as a displacement away from the corresponding fixed point. If this displacement is sufficiently small, the displaced point will not change basins and the associative memory filter will converge to the stored pattern. The amount of allowed displacement is difficult to quantify since it depends on the detailed shape of the basin boundaries. Nevertheless these displacements can be large in a perceptual sense as is seen in Figs. 7A–7C. These examples are correctly recognized by the network and illustrate a limited degree of invariant pattern recognition with respect to obscuration, translation, and noise. From the previous discussion it is also clear that the system will exhibit invariant pattern recognition with respect to small rotations and small scale changes. Furthermore the system will also disambiguate two patterns presented simultaneously. This latter feature is a direct consequence of the nonlinear nature of the associative memory and deserves a brief discussion.

FIG. 7. Three images used to demonstrate invariant pattern recognition. Image (A) is partially obscured. Image (B) has been shifted to the left by approximately 10% (2 pixels). Image (C) has uniform noise in the range ± 0.5 added to it.

The role played by nonlinearity in the associative memory filter becomes clear when one compares the behavior of a dynamical associative memory with a linear associative memory, e.g., Kohonen's model (1984). Consider the case where the memory cue consists of a linear combination of two stored memories. In this case the nonlinear dynamical memory will converge onto the dominant memory provided the contribution from the secondary memory is sufficiently small. More precisely any linear combination of two memories can be represented as a point on a line segment joining the two memories in the state space. If the basins are not overly convoluted the dominant memory will be recalled reliably. If the basins are overly convoluted the dominant memory will be recalled only if the contribution from the secondary memory is sufficiently small. On the other hand, the linear associative memory cannot separate the two images because the recall is based on a projection operator. This linear operator simply projects the initial state (the memory cue) onto the subspace **S** of the state spanned by the stored memories. Now suppose that the initial state is the sum of a stored memory $\mathbf{x}_s$ plus a perturbation ε, i.e., $\mathbf{x} = \mathbf{x}_s + \varepsilon$. The projection operator will be able to retrieve the "correct" memory, only to the extent that ε is contained in the space orthogonal to **S.** On the other hand, if ε is contained in **S** the projection operator simply performs the identity operation. One concludes that a linear associative memory is not an appropriate input filter for a pattern recognition neural network.

The recognition task performed by this simple system could be performed by an associative memory alone by adding three visible units which label each picture. However, this would not have served the purpose of demonstrating hierarchical architecture. Furthermore, since the simple system described here already exhibits, to a limited degree, many of the properties which are important for practical pattern recognition devices, it is not unreasonable to assume that by building up a suitable hierarchy of filters, a system with more robust invariant pattern recognition properties will emerge (e.g., Fukushima (1987)).

10. Discussion and Conclusions

This paper has presented a systematic approach for exploiting the dynamics of a general class of neurodynamical systems for the purpose of neural computation. The starting point was an understanding of the model-independent properties of these systems. Two filters were identified, one which performed continuous nonlinear transformations and another which performed autoassociative recall. It was shown that the continuous mapper and the autoassociative memory make conflicting demands on the neurodynamical system. The former requires a unique attractor in the operating region whereas the latter requires multiple attractors in the operating region.

It was shown that these conflicting demands cause the failure of conventional recurrent backpropagation when an attempt is made to store multiple patterns in an associative memory which starts with an insufficient number of point attractors. A backpropagation technique was developed for the associative memory which effectively converts it into a continuous mapper during the training process. This results in an algorithm which introduces additional fixed points near the to-be-stored memories.

The identification of primitive filters and the development of programming techniques for them is a first step in a systematic approach for the construction of hierarchically organized networks. This approach was demonstrated by the construction of a simple hierarchically organized network for pattern recognition. The simple system exhibited a limited degree of invariant pattern recognition.

There are several outstanding research questions which still need to be addressed: first is the question of stability. It is not difficult to start a recurrent network with weights which satisfy one or another of the stability constraints which are given in Section 4. However, it is usual for the weights to eventually violate these constraints as the system learns. Nevertheless, this investigator rarely experiences the onset of oscillations in any of his simulations. It is difficult to believe that this is due to chance.

Since the onset of instability must be associated with a catastrophe it seems reasonable to conjecture that a detailed study of Eq. (5.1) might reveal conditions under which catastrophes cannot occur. This could explain the remarkable stability of the trained networks. Work on this question is underway.

A second research question is how to interface the various filters. This was not a problem in the example given in the previous section. A more difficult case occurs when one wishes to connect two associative memories. The question is: "How does one reset the memory in the second layer of a hierarchy?" One solution is to introduce time-dependent oscillations to latch the output of one filter into the input of another. Such a mechanism would introduce a system-wide "clock." This solution is theoretically feasible, but the question is open as to whether this is a desirable solution for physical dynamical systems.

In closing, it is the belief of this investigator that the promise of neural computation will be realized when two lines of research converge. One line of research, which is the subject of this paper, is to understand the underlying dynamics and architecture to the point that it becomes possible to model neural computational systems in a simple and systematic way. The second line of research is to understand how to exploit the properties of real collective physical systems to implement these models in native hardware.

Appendix A

In this appendix the steps required to derive Eq. (5.10) are given. Begin by differentiating Eq. (5.19) with respect to w_{rs} on both sides. The result is

$$\frac{\partial x_i^\infty}{\partial w_{rs}} = g_r'(u_r^\infty) \sum_j \left\{ \frac{\partial w_{ij}}{\partial w_{rs}} x_j^\infty + w_{ij} \frac{\partial x_j^\infty}{\partial w_{rs}} \right\}. \tag{A.1}$$

Now, since the elements of the matrix $\mathbf{w}$ are independent it follows that the partial derivative $\partial w_{ij}/\partial w_{rs}$ is one if and only if $i = r$ and $j = s$ and zero otherwise, i.e.,

$$\frac{\partial w_{ij}}{\partial w_{rs}} = \delta_{ir}\delta_{js}, \tag{A.2}$$

where δ_{ij} are the elements of the identity matrix. One can simplify the right hand side of Eq. (A.1) by substituting Eq. (A.2) into Eq. (A.1) and performing the summation over j. Also the left hand side can be expressed

as the product of the identity matrix times the matrix of partial derivatives of x_j^∞. The result is

$$\sum_j \delta_{ij} \frac{\partial x_j^\infty}{\partial w_{rs}} = g_r'(u_r^\infty)\left\{\delta_{ir} x_s^\infty + \sum_j w_{ij} \frac{\partial x_j^\infty}{\partial w_{rs}}\right\}. \tag{A.3}$$

On collecting all the partial derivatives in (A.3) on the left hand side, one obtains

$$\sum_k L_{ij} \frac{\partial x_j^\infty}{\partial w_{rs}} = \delta_{ir} g_r'(u_r^\infty) x_s^\infty, \tag{A.4}$$

where L_{ij} is given by

$$L_{ij} = \delta_{ij} - g_i'(u_i^\infty) w_{ij}. \tag{A.5}$$

Finally, multiply both sides of (A.4) by $(\mathbf{L}^{-1})_{ki}$, i.e., by the matrix elements of the inverse of $\mathbf{L}$, and sum over i. The result is

$$\frac{\partial x_k^\infty}{\partial w_{rs}} = (\mathbf{L}^{-1})_{kr} g_r'(u_r^\infty) x_s^\infty, \tag{A.6}$$

This is the desired result.

Appendix B

Almeida (1987) has shown that the convergence of the forward propagation implies the local stability of the backward propagation. To see why this must be the case it suffices to linearize Eq. (4.1) or Eq. (5.1) about a point attractor, i.e., $\mathbf{x} = \mathbf{x}^\infty + \boldsymbol{\varepsilon}$. The resulting linear equation has the form (expressed in vector notation)

$$d\boldsymbol{\varepsilon}/dt = -\mathbf{L}\boldsymbol{\varepsilon}, \tag{B.1}$$

where $\mathbf{L}$ is given by Eq. (5.12). Now observe that the backward propagation Eq. (5.17) has the form

$$d\mathbf{y}/dt = -\mathbf{L}^{\mathrm{T}}\mathbf{y} + \mathbf{b}, \tag{B.2}$$

where $b_k = g_k'(u_k) J_k$. Now, from (B.1) it is clear that the local stability of the forward equations depends on the eigenvalues of the matrix $\mathbf{L}$ and from Eq. (B.2) it is clear that the local stability of backward propagation

equations depends on the eigenvalues of the transposed matrix $\mathbf{L}^T$. But, $\mathbf{L}$ and its transpose $\mathbf{L}^T$ both have the same eigenvalues, hence if a fixed point of Eq. (5.12) is stable so is the corresponding fixed point of Eq. (5.17). A similar result holds for higher order neural network models.

Appendix C

This appendix discusses several issues relating to the implementation of the recurrent backpropagation algorithms on digital computers. The equations of motion can be solved with the usual numerical techniques for integrating differential equations, e.g., Euler or Runge-Kutta. In practice it turns out that the Euler method (also called first-order finite difference) suffices for converging onto the steady state solution. With a time step of $\Delta t = 1$ the forward propagation equations reduce to the forms used by Almeida (1987), Rohwer and Forrest (1987), Parker (1982), and others. These finite difference equations can exhibit oscillations which do not occur in the differential equations or in finite difference simulations with $\Delta t < 1$. This investigator typically uses $\Delta t = 0.9$ to suppress these oscillations.

A full dynamical simulation requires that the forward equations, the backward equations, and the weight update equations be solved simultaneously and that the patterns be presented randomly. In practice this integration is prohibitively time consuming. The usual approach is to integrate the forward propagation equations until they converge, then to integrate the backward propagation equations until they converge, and then to calculate the gradient. This is repeated for all the patterns while accumulating the gradient. The accumulation over all patterns makes the learning dynamics deterministic rather than stochastic and guarantees that the system will converge. However, it may only converge to a local minimum.

The mathematical form of the associative memory learning equation leads to certain computational efficiencies. First, it is not necessary to relax either the forward or backward propagation equations for the visible units because the fixed points of (8.12) can be calculated analytically for the visible units. Second, if there are no connections between hidden units it is not necessary to relax any of the equations in the network. The fixed points of the forward and backward propagation equations may be calculated in two iterations each, as if the system were a conventional feed-forward network with a single hidden layer. Finally, if there are no hidden units the fixed points of the backward propagation equation can be calculated analytically without any iterations. This final case is the learning algorithm used by Samad and Harper (1987) in an associative memory which used an annealing scheme to recall memories rather than Eq. (5.1).

In networks with units with widely disparate fan-ins it is useful to multiply the initial weights by the inverse fan-in. If this is done, the gradient in the learning equations should be multiplied by the same factor. This suppresses saturation of the nodes with large fan-in and leads to improved performance of the learning algorithm.

A useful measure of progress during the learning process is the normalized error, E_n. This is defined to be

$$E_n = \frac{E}{E_{mean}}, \tag{C.1}$$

where E is the error function summed over all patterns and E_{mean} is

$$E_{mean} = \frac{1}{2} \sum_{\alpha} \sum_{i} [\eta_i^{\alpha} - \langle \eta_i \rangle]^2, \tag{C.2}$$

where η_i^{α} is the αth target output for the ith unit and where $\langle \eta_i \rangle$ is the average over patterns of the target values at the ith unit. E_n is useful because, independent of network topology and problem domain, the backpropagation algorithm learns the average output pattern very rapidly. Therefore the network initially goes to $E_n = 1$. As the distinctions between the patterns are learned, E_n gradually drops below one.

ACKNOWLEDGMENTS

The author thanks Liam Healy for considerable assistance during the production of this paper and for providing crucial insight into the behavior of dissipative systems. Thanks also to Robert Jenkins, Juan Pineda, W. Jack Rugh, Terry Sejnowski, Kim Strohbehn, and Ben Yuhas for productive discussions. The two referees also contributed very constructive comments. This work was supported in part by the Air Force Office of Scientific Research under Grant AFOSR-87-0354 and by the Applied Physics Laboratory under IRAD-X8U.

REFERENCES

ALMEIDA, L. B. (1987), A learning rule for asynchronous perceptrons with feedback in a combinatorial environment, *in* "Proceedings of the IEEE First Annual International Conference on Neural Networks" (M. Caudil and C. Butler, Eds.), pp. 609–618, San Diego, CA.

AMARI, SHUN-ICHI (1972), Characteristics of random nets of analog neuron-like elements, *IEEE Trans. Syst. Man Cybernetics* **2**, 643–657

AMARI, SHUN-ICHI (1977), A mathematical approach to neural systems, *in* "Systems Neuroscience" (J. Metzler, Ed.), pp. 67–118, Academic Press, New York.

ARNOLD, V. I. (1986), "Catastrophe Theory," Springer-Verlag, Berlin.

ATIYA, A. (1987), Learning on a general network, *in* "Proceedings of IEEE Conference on Neural Information Processing Systems" (D. Z. Anderson, ed.), Denver, CO, Nov. 8–12, to appear.

BALLARD, D. H. (1987), "Modular Learning in Neural Networks," AAAI Proceedings of 6th National Conference on Artificial Intelligence, pp. 279–284.

BAUM, E. B., AND WILCZEK, F. (1987), Supervised learning of probability distributions by neural networks, preprint.

BINDER, K., AND YOUNG, A. P. (1986), Spin glasses: Experimental facts, theoretical concepts, and open questions, *Rev. Mod. Phys.* **58,** 801–976.

COHEN, M. A., AND GROSSBERG, S. (1983), Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. IEEE Transactions on Systems, Man, and Cybernetics, SMC-13, pp 815–826.

FUKUSHIMA, K. (1987), A neural network model for selective attention in visual pattern recognition and associative recall, *Appl. Opt.* **26**(23), 4985–4992.

GEMAN, S. (1981), The law of large numbers in neural modelling, *in* "SIAM–AMS Proceedings," Vol. 13, pp. 91–105.

HINTON, G. E., SEJNOWSKI, T. J., AND ACKLEY, D. H. (1984), "Boltzmann Machines: Constraint Satisfaction Networks That Learn, Tech. Rep. No. CMU-CS-84-119, Carnegie-Mellon University, Dept. of Computer Science, Pittsburgh, PA.

HOPFIELD, J. J. (1982), Neural networks as physical systems with emergent collective computational abilities, *Proc. Natl. Acad. Sci. USA Bio.* **79,** 2554–2558.

HOPFIELD, J. J. (1984), Neurons with graded response have collective computational properties like those of two-state neurons, *Proc. Natl. Acad. Sci. USA Bio.* **81,** 3088–3092.

KOHONEN, T. (1984), "Self-Organization and Associative Memory," Springer-Verlag, Berlin.

LAPEDES, A., AND FARBER, R. (1986a), A self-optimizing, nonsymmetrical neural net for content addressable memory and pattern recognition, *Physica D* **22,** 247–259.

LAPEDES, A., AND FARBER, R. (1986b), Programming a massively parallel, computation universal system: Static behavior, *in* "Neural Networks for Computing" (J. S. Denker, Ed.), AIP Conference Proceedings, Vol. 151, pp. 283–298, Snowbird, UT.

LE CUN, Y. (1985), Une procedure d'appresentissage pour reseau a seuil assymetrique [A learning procedure for an asymmetric threshold network], *Proc. Cognitiva* **85,** 599–604.

MINSKY, M., AND PAPERT, S. (1969), "Perceptrons," MIT Press, Cambridge.

PARKER, D. B. (1982), "Learning-Logic," Invention Report, S81-64, File 1, Office of Technology Licensing, Stanford University.

PARKER, D. B. (1987), Second order backpropagation: Implementing an optimal $O(n)$ approximation to Newton's method as an artificial neural network, draft preprint obtained from author.

PINEDA, F. J. (1987a), Generalization of backpropagation to recurrent neural networks, *Phys. Rev. Lett.* **18,** 2229–2232.

PINEDA, F. J. (1987b), Generalization of backpropagation to recurrent and higher order networks, *in* "Proceedings of IEEE Conference on Neural Information Processing Systems" (D. Z. Anderson, Ed.), Denver, CO, Nov. 8–12, to appear.

ROHWER, R., AND FORREST, B. (1987), Training time dependence in neural networks, *in* "Proceedings of the IEEE First Annual International Conference on Neural Networks" (M. Caudil and C. Butler, Eds.), Vol. 2, pp. 701–708, San Diego, CA.

RUMELHART, D. E., HINTON, G. E., AND WILLIAMS, R. J. (1986), Learning internal repre-

sentations by error propagation, *in* "Parallel Distributed Processing" (D. E. Rumelhart and J. L. McClelland, Eds.), pp 318–362, MIT Press, Cambridge.

SAMAD, T., AND HARPER, P. (1987), Associative memory storage using a variant of the generalized delta rule, *in* "Proceedings of the IEEE First Annual International Conference on Neural Networks" (M. Caudil and C. Butler, Eds.), Vol. 3, pp. 173–183, San Diego, CA.

SEJNOWSKI, T. J. (1981), Skeleton filters in the brain, *in* "Parallel Models of Associative Memory" (G. E. Hinton and J. A. Anderson, Eds.), pp. 189–212, Erlbaum, Hillsdale, NJ.

SIMARD, P. (1988), Private communication.

WHITE, H. (1987), Some asymptotic results for back-propagation, *in* "Proceedings of the IEEE First Annual International Conference on Neural Networks" (M. Caudil and C. Butler, Eds.), Vol. 3, pp. 261–266, San Diego, CA.