PROGRAMMING A MASSIVELY PARALLEL, COMPUTATION UNIVERSAL SYSTEM: STATIC BEHAVIOR

Alan Lapedes and Robert Farber

Theoretical Division
Los Alamos National Laboratory
Los Alamos, New Mexico 87545
March 27, 1986

Abstract

Massively parallel systems are presently the focus of intense interest for a variety of reasons. A key problem is how to control, or "program" these systems. In previous work by the authors, the "optimum finding" properties of Hopfield neural nets were applied to the nets themselves to create a "neural compiler." This was done in such a way that the problem of programming the attractors of one neural net (called the Slave net) was expressed as an optimization problem that was in turn solved by a second neural net (the Master net). The procedure is effective and efficient. In this series of papers we extend that approach to programming nets that contain interneurons (sometimes called "hidden neurons"), and thus we deal with nets capable of universal computation. Our work is closely related to recent work of Rummelhart et al. (also Parker, and LeChun), which may be viewed as a special case of this formalism and therefore of "computing with attractors." In later papers in this series, we present the theory for programming time dependent behavior, and consider practical implementations. One may expect numerous applications in view of the computation universality of these networks.

I. Introduction

Massively parallel systems are the focus of intense interest by numerous researchers spanning many disciplines. They have tremendous practical implications for new kinds of computing[1,2] and may give some insight into how massively parallel biological brains operate.[3,4] They are also theoretically interesting when analyzed as an example of a complex, nonlinear dynamical system.[5] Programming, i.e. controlling, these systems is a difficult task. Recent work of Hopfield[1,2] on a particular system, neural networks, has stimulated intense interest on the part of many researchers. He has shown that nets with symmetrical interactions posess a Lyapunov function, and that such a function is very useful in controlling the network behavior. In particular, he has shown that continuous state neural networks (as opposed to discrete, two state networks) may be used to perform a credible job of solving difficult optimization problems such as the classic Traveling Salesman Problem (see also M. Takeda, G. Goodman[6]). In these types of problems the solution to the optimization problem is a fixed point of the neural dynamics and the problem constraints and the data (e.g. the distances in the TSP problem) appears in the symmetrical neural connections. The solution to the problem, the fixed point of the net, is of course not known a priori, and thus one runs the nonlinear differential equations for the nets until it relaxes to a fixed point in order to find an optimum.

A second class of difficult computation problems, e.g. Content Addressable Memory, associative memory, pattern recognition (and many other forms of high level symbol processing) can also be attacked with neural networks. Here, one knows the desired fixed points a priori (they correspond, for example, to the patterns to be recognized), however, one does not know in general what neural connections will yield the known fixed points. A partial solution to determine the neural connections was given by Hopfield[2], which is based on a symmetric synaptic connection matrix formed by taking outer products of the desired fixed point vectors. Disadvantages of this approach include a very limited degree of control over the basins of attraction, the symmetry of the synaptic connection matrix (which is almost certainly unphysiological), inability to deal with time dynamic behavior, and the fact that interneurons necessary for universal computation are excluded. In spite of these limitations, Hopfield's pioneering work remains an important first step in the theory of computation for neural networks.

The present authors[7] have approached the problem of programming fixed points into a neural net by directly attacking the conditions that the synaptic connection matrix must satisfy if the desired fixed points are to be present. This was accomplished by posing the general problem of determining the connections of a Slave net as an optimization problem, and then using the "optimum finding" abilities of neural nets to define a second net, a Master net, that performed the optimization. This therefore links the two classes of problems currently under investigation in neural network theory. Advantages of this approach include greater control over the basins of attraction, removal of the symmetry restriction in the Slave net, and as we will show, the ability to control interneurons or "hidden neurons" that are necessary for universal computation. The method is effective and efficient[8] and may also be viewed as a learning algorithm for neural networks. We have also extended the method to allow programming of time dependent behavior (paper II of this series).

The extension of our original formalism that is required to deal with hidden neurons is closely related to recent work of Rummelhart et al.,[9] Parker[10], and Le Chun.[11] This work, performed independently of the Master/Slave approach is shown to be a special case of the more general formalism presented below. This comment is not intended in any way to detract from the very fine results of these authors, and we hope that by providing a general setting that the value of such approaches can be better appreciated.

The importance of hidden neurons has, of course, not gone unnoticed by other researchers. It was forcefully pointed out by Minsky et al.[12] that hidden neurons are necessary to solve certain classes of important problems. In fact, it was the difficulty in being able to program these important neurons that led to the decline of interest in the 1970's in the Perceptron formalism of neural networks. Hinton and Sejnowski et al.[13,14] attacked the hidden neuron problem in the early 1980s by developing the Boltzman machine formalism. In this work, simulated annealing[15] for a stochastic two-state neuron model was coupled with the construction of a clever learning algorithm to provide the first solutions for programming massively parallel networks to perform difficult computations. Hinton and Sejnowski and coworkers have provided many beautiful analyses of problems involving hidden neurons and used their procedure to find many solutions for small-scale model problems. However, there exist two severe disadvantages to their approach. These are the extreme computational slowness and the inability to directly incorporate time-varying neural behavior (e.g. direct programming of network limit cycles etc.).

Recently, Sejnowski has used the Rummelhart procedure (in favor to the Boltzman machine procedure) to solve a moderately large-scale problem - that of converting typed text to speech, including correct context dependent pronunciations of phonemes.[16] As we will see in the general formalism below, computational slowness is no longer an issue, and that the work of Rummelhart, Parker, Le Chun, Sejnowski and the present authors are closely related to the neural networks popularized by Hopfield, and are examples of computing with attractors in computation universal systems. In view of the widespread interest by engineers in constructing Hopfield-type nets in hardware (VLSI or optical)[17], it would seem that actual construction of massively parallel, computation universal devices may closely follow the development of theory for controlling these systems.

In this paper (I) we confine our attention to developing the theory for control of static behavior and point out the relations to work of Rummelhart, Parker, Le Chun. Their work inspired the present development. However, our results are completely general in that we allow all possible connections of hidden, input and output neurons, both forward and backward, including fully recurrent hidden connections. "Clamping" of inputs is not necessary - back connections can allow inputs to remain "on" as part of the net's fixed-point configuration. Adjustment of basin boundaries may be accomplished in the manner of Ref. 7. The formalism clearly shows that the method is a procedure for controlling attractors in Hopfield type nets. In paper II we present the theory for controlling time dependent behavior. In subsequent papers we discuss specific applications.

II. Summary of Master/Slave Formalism

The Master/Slave formalism was developed as a general[7] attack on determining what synaptic connection matrix, T_{ij}, yields desired fixed points of the neural net equations

$$U_i + \mu\dot{U}_i = \sum_j T_{ij} g(U_j) + I_i \qquad (1)$$

Here, U_i represents the membrane potential of the i^{th} neuron, μ is the cell summation time constant (or the RC time constant of an electric circuit implementation), T_{ij} is the synaptic connection matrix, $g(U_j)$ is the sigmoidal firing rate curve of the j^{th} neuron (or transfer function of an op-amp), and I_i is an external current (or equivalently an internal threshold). For concreteness we take

$$g(x) = \tfrac{1}{2}(1 + \tanh(\beta x)) \qquad (1a)$$

where β determines the slope of the sigmoid curve. The exact form that the sigmoiu assumes is not crucial to the results. These equations were introduced by Hopfield[1,2] in the context of "problem solving with neural nets" (e.g. the Traveling Salesman Problem), but related equations have also been written down by many researchers[18,19] as an acceptably crude model of neurophysiology. Hopfield showed that if $T_{ij} = T_{ji}$, then a Lyapunov function, E, exists:

$$E = -\tfrac{1}{2} \sum_{ij} T_{ij} V_i V_j - \sum_i I_i V_i + \sum_i \int^{V_i} g^{-1}(x)\, dx \tag{2}$$

where $V_i = g(U_i)$. Equation (2) implies that

$$\frac{dE}{dt} = \sum_i g'(U_i) \cdot \dot{U}_i^2 \tag{3}$$

and since g'(x) is positive semidefinite, E will decrease. E is bounded however, so that it will eventually stop decreasing and $\dot{U}_i$ will be zero, at which point a fixed point is reached. In the high gain limit (large β), the integral in Eq. (2) is negligible and the minima of E that is obtained will be close to a minima of E_1:

$$E_1 = -\tfrac{1}{2} \sum_{ij} T_{ij}\, g(U_i)\, g(U_j) - \sum_i I_i g\,(U_i) \tag{4}$$

Hopfield chose

$$E_1 = -\tfrac{1}{2} \sum_{s=1}^{m} (V^{(s)} \cdot V)^2 \tag{5}$$

where $V_i^{(s)} = g(U_i^{(s)})$, $s = \{1,2...m\}$ are the desired choice of m memory states or fixed points. Although this choice works, subject to limitations discussed in the Introduction, it is ad hoc and it is not a general solution to the problem of inserting desired fixed points into the neural dynamics. The T_{ij} determined by Eq. (5) is a sum of the outer products of the fixed-point vectors and is, of course, symmetric.

In Ref. (7) we considered the general problem of finding T_{ij}'s that produce fixed points at desired states $V_i^{(s)}$, $s = \{1,2...m\}$ with no a priori restriction on the T_{ij}. To do so we reformulated Eq. (1) by introducing a new variable, $V_i^1(t)$ such that

$$U_i = \sum_j T_{ij} V_j + I_i \ . \tag{6}$$

It is easily checked that V_i satisfies

$$V_i + \mu \dot{V}_i = g(\sum_j T_{ij} V_j + I_i) \tag{7}$$

by multiplying Eq. (7) by T_{ij}, summing over i, and comparing to Eq. (1). It may be shown that V_i is essentially a short time average of the firing rate $g(U_i)$, and that at a fixed point of Eq. (7) ($\dot{V}_i = 0$) the value of V_i is the firing rate, $g(U_i)$, of the ith neuron.

The requirement that a particular set of m fixed points $V_i^{(s)}$,

$\{s = 1,2...m\}$ of Eq. 7 exists may be expressed as

$$E_1 = \sum_{s,i} [V_i^{(s)} - g(\sum_j T_{ij} V_j^{(s)} + I_i)]^2 \tag{8}$$

with $E_1 = 0$. From Eq. 7 we see that this requires that $\dot{V}_i = 0$ at each $V_i^{(s)}$. The $V_i^{(s)}$ are the known fixed points that one desired to insert into the net and the T_{ij}'s are unknown but subject to Eq. (8). Note that no restrictions such as symmetry have been placed on the T_{ij}'s. In Ref. 7 we rewrote Eq. (8) slightly as

$$E_1 = \sum_{s,i} [g^{-1}(V_i^{(s)}) - \sum_j T_{ij} V_j^{(s)} - I_i]^2 \tag{9}$$

and noticed that finding the T_{ij}'s satisfying $E_1 = 0$ requires finding the minimum of the quadratic form in T_{ij} that results from expanding the square in Eq. (9).

It is desirable to keep T_{ij} bounded and a natural parameterization that accomplishes this is

$$T_{ij} = 2g(U_{ij}) -1 \tag{10}$$

where $g(U_{ij})$ is another sigmoidal function (between 0 and 1) of the new variable U_{ij}. T_{ij} ranges over -1 to +1. I_i may be parameterized in a similar fashion but for simplicity let us take I_i to be zero for the moment. We consider non-zero I_i in the next section, but wish to keep the following summary as simple as possible and will not consider non-zero I_i until later. There will be no problems with non-zero I_i - it just adds some more terms.

Inserting (10) in (9) and expanding the square yields

$$E_1 = - \sum_{ijk\ell} T_{ijk\ell}\, g(U_{ij})\, g(U_{k\ell}) - \sum_{ij} I_{ij}\, g(U_{ij}) + \text{constant} \tag{11}$$

where

$$T_{ijk\ell} = - 4 \sum_s \delta_{ki}\, V_j^{(s)} V_\ell^{(s)} \quad , \quad T_{ijk\ell} = T_{k\ell ij} \tag{12a}$$

$$I_{ij} = \sum_s 4\, V_j^{(s)} [g^{-1}(V_i^{(s)}) + \sum_\ell V_\ell^{(s)}] \quad . \tag{12b}$$

Comparison of Eq. (11) with Eq. (4) shows that Eq. (11) is related to the Lyapunov function for <u>another</u> network, the Master net, with membrane potential U_{ij} and synaptic matrix and external currents given by Eq. 12. Adding a suitable integral places Eq. (11) exactly in the same form as Eq. (2) with the Master net neurons indexed by two indices, i and j, instead of just one index. Therefore, solving the two index form of Eq. (1)

$$U_{ij} + \mu U_{ij} = \sum_{k\ell} T_{ijk\ell}\, g(U_{k\ell}) + I_{ij} \tag{13}$$

will result in a fixed point of the Master net, $g(\bar{U}_{ij})$ that is a minimum of Eq. (11) and equivalently Eq. (9) (for large β). At the Master fixed point the Master neurons are firing at a sustained rate $g(\bar{U}_{ij})$ and the Slave net T_{ij} will be given by Eq. (10), i.e.,

$$T_{ij} = 2g(\bar{U}_{ij})-1 \quad . \tag{14}$$

The Master net firing rates, therefore, modulate the synaptic connections in the Slave.

Simulations[7] show that a good approximation to the global minimum of Eq. (11) is generally found, and that the resulting Slave net T_{ij}'s generally produce the desired fixed points. For further discussion, and extensions to add control over basins of attraction, see Ref. 7. More discussion and an analysis of the efficiency of the above procedure may be found in Ref. 8.

III. Extension of Master/Slave Procedure: Hidden Neurons

The above procedure was originally developed for auto-associative memory.[20] That is, fixed points were inserted in a Slave net such that configurations in the basins of attraction evolved to the desired fixed points. Hetero-associative memory[20], i.e. true associative memory, seeks to find T_{ij}'s such that particular configurations A_1, $A_2 \ldots A_m$ will evolve to other specific patterns B_1, $B_2 \ldots B_m$. Thus, pattern $A_1 \rightarrow B_1$, $A_2 \rightarrow B_2$ etc. A trivial psychological interpretation is that recalling fact A_1 ("a dog") evokes fact B_1 ("dogs bark"). Although this is amusing (and potentially very powerful in, say, a relational data base or massively parallel expert system) the psychological interpretation is the least important aspect of hetero-associative memory.

In our opinion the prime significance of hetero-associative memory, when accomplished with interneurons or hidden neurons, is the ability to perform universal computation. For example, the following bit pattern associations 00 → 0, 01 → 1, 10 → 1, 11 → 0 may only be realized in a neural network with the aid of hidden neurons[12,13,14] and are the logical expression of XOR ("exclusive or"). Negation and other logical functions may also be expressed as simple associations and can only be accomplished with hidden neurons. Therefore networks with hidden neurons are capable of space-bounded universal computation. To realize this capability one must be able to control the attractors in such networks.

In the following we show that a minor extension of the Master/Slave procedure (section II and Ref. 7) allows one to present a collection of inputs and outputs to a Master/Slave neural net, where the outputs may be complicated logical functions of the inputs, so that the net will adapt its attractors to "deduce" an algorithm that reproduces the input/output pairs. The net has limited information capacity and is clearly not just recording the input/output pairs, but is instead developing a collective algorithm to associate input and output. The net has certain capabilities of deduction and generalization (this seems to be related to overloading the memory capacity) and may even be operated in a "backward mode" where it is told the output and can back propagate to "correct" a corrupted input. We allow all possible connections of hidden, input, and output neurons, both forwards and backwards, including fully recurrent connections among the hidden neurons. "Clamping" of inputs is not necessary. Adjustment of basin boundaries, if desired, may be accomplished in a similar manner to Ref. 7. A natural extension of this method, to deal with time dependent behavior is presented in the next paper in this series.

Development of the Master/Slave procedure for hidden neurons begins with Eq. (7) and (8). In the previous section, and in Ref (7), we introduced the g^{-1} function so that the energy expression in Eq. (8) could be written as a quadratic form (Eq. (9)) with known coefficients related to the desired fixed points of the Slave net. This was done for pedagogical purposes to clarify the connection to a Hopfield Lyapunov function for a Master net, which was originally presented as a quadratic form. However, it is not mathematically necessary to do this, and to clarify the relation to Rummelhart et al., we consider Eq.(8)instead of Eq.(9).

Let us now parameterize T_{ij} as before (Eq. 10) and introduce a similar parameterization for I_i that bounds the allowed current values,

$$T_{ij} = 2g(U_{ij})-1 = \tanh(\beta U_{ij}) \tag{15a}$$

$$I_i = 2g(S_i)-1 = \tanh(\beta S_i) \tag{15b}$$

Actually, the β appearing in 15a,b does not have any relation to the β of the Slave net, so for clarity let us retain the notation of Eq. (1a) for the Slave net

$$g(x) = \tfrac{1}{2}(1 + \tanh(\beta x)) \text{ -- Slave net} \tag{16a}$$

and introduce a better notation $g_M(x)$, for the Master net

$$g_M(x) = \tanh(\beta_M x) \text{ -- (Master net)} \tag{16b}$$

with

$$T_{ij} = g_M(U_{ij}) \ \varepsilon[-1,1] \tag{17a}$$

$$I_i = g_M(S_i) \ \varepsilon[-1,1] \quad . \tag{17b}$$

Let us now add two integrals to Eq. (8) to produce a Lyapunov function for the Master net that is <u>not</u> a quadratic form:

$$E = E_1 + \sum_{ij} \int^{g_M(U_{ij})} g_M^{-1}(x)dx + \sum_i \int^{g_M(S_i)} g_M^{-1}(x)dx \tag{18a}$$

with

$$E_1 = \sum_{s,i} [V_i^{(s)} - g(\Sigma\, T_{ij}\, V_j^{(s)} + I_i)]^2 \tag{18b}$$

and T_{ij}, I_i parameterized as Eq. 17a,b. If

$$U_{ij} + \mu \dot{U}_{ij} = - \frac{\partial E_1}{\partial T_{ij}} \tag{19a}$$

$$S_i + \mu \dot{S}_i = - \frac{\partial E_1}{\partial I_i} \tag{19b}$$

then it is easily verified that

$$\frac{dE}{dt} \leq 0 \tag{20}$$

and an identical argument to the previous section shows that E will decrease to near a minimum of E_1, at which point the Master neurons (with membrane potentials U_{ij} and S_i) are firing at a constant rate determined by the fixed points of 19a,b. The Slave net synapses and currents are then determined by (17a,b) evaluated at the fixed point of (19a,b).

To summarize, the logical argument so far is virtually identical to the previous section and Ref. 7. The only change is that we chose to use the nonquadratic form of Eq. 18 and have also included contributions from currents, I_i. It may be worth emphasizing that the above demonstrates that restrictions to quadratic Lyapunov functions are totally unnecessary, and that Hopfield's "energy minimization" arguments apply to a much wider class of functions than merely quadratic. There seems to be a misconception present in the literature that Lyapunov functions must be restricted to be quadratic, which in turn restricts the class of optimization problems that can be attacked. This restriction is unnecessary.

Up until now we have always assumed that all the $V_i^{(s)}$ fixed-point components were known because they were specified by the user and that there were no hidden neurons. Let us now consider a neural net where a subset of the neurons are arbitrarily labeled as Input, another subset as Output, and the remaining neurons as Hidden (see Fig. 1). Only pairs of associations of Input/Output are now specified by the user, however, we will now show that at a fixed point of the Slave, the states of the hidden neurons are known as well. Knowing the state of all the neurons at the fixed point allows us to determine the synaptic matrix, T_{ij} and currents I_i by the procedure of Ref. (7) (see previous section).

To summarize, the idea is to present the known input/output pairs to the Master net, which then determines Slave net T_{ij}'s and I_i's related to the firing rates of the Master at its own fixed point. A later presentation of an input to the Slave (with the hiddens and outputs set to states midway between "on" and "off") will place the Slave in the basin of attraction of a Slave fixed point

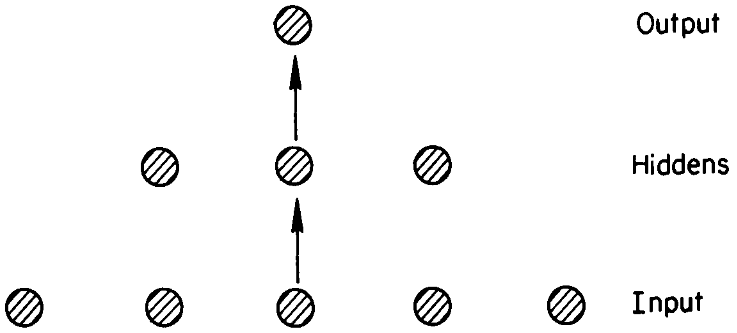

Figure 1

that then completes (i.e. "associates") the appropriate output. Because neural nets will only "complete correctly" if there is sufficient initial information, we would expect that if the inputs do not constitute a majority of the neurons in the net then completion may be incorrect. For these special situations one may disallow back connections to the inputs. However, for most cases the output neurons typically signal "yes/no" answers about complicated input patterns, and therefore completion of the much fewer output neurons will occur. Utilization of the completion capability of neural networks (which is a consequence of the attractor structure) adds new possibilities to neural net information processing.

We will first consider almost all possible connections of the Slave net, both forwards and backwards, including recurrent connections among the Inputs and Outputs. For the moment the only case we will not consider is recurrent connections among the hiddens. Recurrent hidden connections require a slight additional argument that we provide later on so as not to unnecessarily confuse the logical argument. This will then complete all possible cases.

The global minimum of Eq. 18b and 18a expresses the condition that Eq. (7) has fixed points at desired locations, $V_i^{(s)}$. Because the fixed-point values are to be specified only for the Input/Output neurons, we restrict the sum over "i" in E_1 (Eq. 18b) as follows

$$E_1 = \sum_s \sum_{i\varepsilon I/O} [V_i^{(s)} - g(\sum_j T_{ij} V_j^{(s)} + I_i)]^2 \tag{21a}$$

or expanding the $\sum_J$

$$E_1 = \sum_s \sum_{i\varepsilon I/O} [V_i^{(s)} - g(\sum_{j\varepsilon I/O} T_{ij} V_j^{(s)} + \sum_{j\varepsilon H} T_{ij} V_j^{(s)} + I_i)]^2 \quad . \tag{21b}$$

The $V_j^{(s)}$ for j ε I/O are known, while the $V_j^{(s)}$ for jεH are <u>determined</u> by the known values of $V_j^{(s)}$ for jεI/O because at a fixed point

$$V_j^{(s)} = g(\sum_{k\varepsilon I/O} T_{jk} V_k^{(s)} + I_j) \quad , \quad \text{for } j\varepsilon H \tag{22}$$

if we disallow hidden-hidden connections for the moment. Inserting Eq. (22) in Eq. (21b) yields (for no hidden-hidden connections)

$$E_1 = \sum_s \sum_{i\varepsilon I/O} [V_i^{(s)} - g(\sum_{j\varepsilon I/O} T_{ij} V_j^{(s)} + \sum_{j\varepsilon H} T_{ij} g(\sum_{k\varepsilon I/O} T_{jk} V_k^{(s)} + I_j) + I_i)]^2 \quad . \tag{23}$$

If we now allow the hidden-hidden connections to be only feed-forward connections then we may split the sum over jεH in Eq. (21b) into a sum over $j\varepsilon H_1$ (the first hidden layer), and a sum over $j\varepsilon H_2$ (the second hidden layer), see Fig. 2.

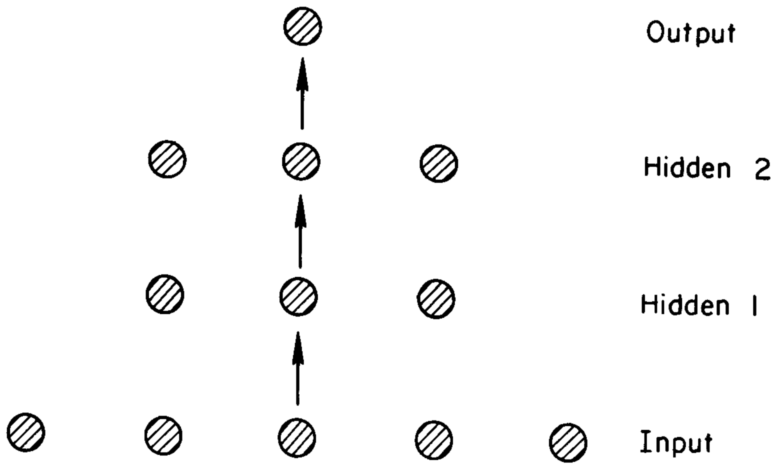

Figure 2

In this case Eq. (21b) becomes

$$E_1 = \sum_s \sum_{i\varepsilon I/O} [V_i^{(s)} - g(\sum_{j\varepsilon I/O} T_{ij}V_j^{(s)} + \sum_{j\varepsilon H_1} T_{ij}V_j^{(s)} + \sum_{j\varepsilon H_2} T_{ij}V_j^{(s)} + I_i)]^2 \tag{24}$$

where V_j for $j\varepsilon H_1$ is as before (Eq. 22) and V_j for $j\varepsilon H_2$ becomes:

$$V_j^{(s)} = g(\sum_{k\varepsilon I/O} T_{jk} V_k^{(s)} + \sum_{k\varepsilon H_1} T_{jk} V_k^{(s)} + I_j)$$

$$= g(\sum_{k\varepsilon I/O} T_{jk} V_k^{(s)} + \sum_{k\varepsilon H_1} T_{jk}\, g(\sum_{\ell\varepsilon I/O} T_{k\ell} V_\ell^{(s)} + I_k) + I_j) \quad .(25)$$

This clearly generalizes for any number of hidden layers $H_1, H_2 \ldots H_n$.

Eq. (19a,19b) will determine the appropriate synaptic connections and currents given the expressions E_1 (above). It should already be clear that this procedure incorporates the Rummelhart et al. algorithm as a special case if one uses a linear function for $g_M(x)$ (Eq. 16b) and allows only feed forward connections. The substitutions performed in Eq. 22 → 25 replace the "back propagation" of Rummelhart while the Eq. (19a,b) are identical to Rummelhart's when one "smooths the gradient" by overrelaxation (see Sejnowski Ref. 16). In actual practice, Rummelhart changes the synaptic weights (using an Euler discretization of Eq. 19) after a presentation of each pattern. He notes, however, that he is really attempting to perform gradient descent, which requires weight changes to be made only after the complete set of patterns is "presented" (as we do above). His simulations verify his claim that changing the weights after presentation of each pattern does not destroy the gradient descent process. A more detailed analysis of the relation of our procedure to that of Rummelhart is given in the next section.

We now consider the case where the hidden-hidden connections are allowed to be recurrent. This will complete the generalization to all possible types of connections. The case of feed forward H-H connections was considered above by appropriately generalizing Eq. (22). We now return to Eq. (22) and generalize it for the case of recurrent H-H connections. If we have some recurrent H-H connections and some purely feed forward (or feed backward) H-H connections then we would need to add additional terms as in Eq. 24, 25. The separate cases are additive (one just adds in more terms for the separate cases), so we see little expository value in writing down a huge complicated looking expression that sums up all the separate cases. We therefore return to Eq. (22) and add in only recurrent H-H connections.

In this situation Eq. (22) becomes

$$V_j^{(s)} = g(\sum_{k\varepsilon I/O} T_{jk} V_k^{(s)} + \sum_{k\varepsilon H} T_{jk} V_k^{(s)} + I_j) \tag{26}$$

for $j\varepsilon H_{\text{recurrent}}$. Because there are now feed forward and feed backward H-H connections, we are not able to substitute an expression in the known I/O values for the second summation in Eq. (26) as we did in Eq. (25). We therefore leave the second summation as an extra unknown variable, $X_j^{(s)}$, where:

$$X_j^{(s)} = \sum_{k\varepsilon H} T_{jk} V_k^{(s)} \quad . \tag{27}$$

Thus

$$V_j^{(s)} = g\Big(\sum_{k\varepsilon I/O} T_{jk} V_k^{(s)} + X_j^{(s)} + I_j \Big) \quad . \tag{28}$$

We now proceed exactly as in the above and form the analogue to Eq. (23) by substituting Eq. 28 in Eq. (21b).

$$E_1 = \sum_s \sum_{i\varepsilon I/O} [V_i^{(s)} - g(\sum_{j\varepsilon I/O} T_{ij} V_j^{(s)} + \sum_{j\varepsilon H} T_{ij} g(\sum_{k\varepsilon I/O} T_{jk} V_k^{(s)} + X_j^{(s)} + I_j) + I_i)]^2 \quad . \tag{29}$$

The unknown $X_j^{(s)}$ appears on virtually the same footing as the unknown, I_j, so that we may parameterize it in a similar fashion to Eq. 17b, add the analogous integral to Eq. 18a and discover all the unknowns by running Eq. 19 supplemented with a third equation for the third variable $X_j^{(s)}$. Of course $X_j^{(s)}$ is not in the range (-1,1), but is instead in the range (-H,H) (see Eq. 27), so that the parameterization similar to 17b uses $H \cdot g_M$ and not just g_M where H = (number of recurrent hiddens).

At the end of the above procedure we know the values of T_{jk} for ($j\varepsilon H$) and ($k\varepsilon I/O$), and also for ($j\varepsilon I/O$) and ($k\varepsilon I/O$), and for $j\varepsilon(I/O)$ and ($k\varepsilon H$). We also know all I_j and the $X_j^{(s)}$. All that remains is the determination of the values of T_{jk} for ($j\varepsilon H$) and ($k\varepsilon H$). We have all the information we need to do this. In view of the already known variables and Eq. 28, we also know the values of $V_j^{(s)}$ for all j, including $j\varepsilon H$. Equation (27) now determines the unknown values of T_{jk} because we may now form a new E_1,

$$E_1 = [X_j^{(s)} - \sum_{k\varepsilon H} T_{jk} V_k^{(s)}]^2 \quad ,$$

and proceed in the usual fashion to determine T_{jk}. Note the interesting consistency condition that arises (over, under, or exact determination of T_{jk}) when one includes recurrent hidden connections.

It is now clear that a straightforward extension of the Master/Slave approach to computing with attractors allows one to program networks capable of universal computation. The only slight complication occurs when one allows recurrent hidden connections. It would be of great interest to have a "theory of computation" for neural networks, which would give some insight into the conditions on types of problems that require recurrent hidden connections (as well as other hidden connections).

IV. Relation to Rummelhart et al.

Rummelhart et al. consider a neural network with a "synchronous update rule" such that the output, O, of a neuron at time (t + 1) is given a function, f, of the other neurons at time t:

$$O_i(t + 1) = f(\sum_j T_{ij} O_j + I_i) \quad . \tag{31}$$

They consider primarily feed forward networks, with input feeding to hidden neurons, that in turn feed to output neurons. Recurrent networks are somewhat unnaturally handled by relating them to feed forward networks. They consider an "energy function"

$$E^{(s)} = \tfrac{1}{2} \sum_i (t_i^{(s)} - O_i^{(s)})^2 \tag{32}$$

where $t_i^{(s)}$ is a desired, or target, configuration for the output units (when given an input configuration labeled by s), and $O_i^{(s)}$ is the actual output that occurs. A "learning rule" for T_{ij} was developed by performing gradient descent on $E^{(s)}$ i.e.

$$\Delta T_{ij} = - \frac{\partial E^{(s)}}{\partial T_{ij}} \eta \tag{33}$$

where

$$\frac{\partial E^{(s)}}{\partial T_{ij}} = \delta_i^{(s)} O_j^{(s)} \tag{34}$$

and η is a "learning constant."

The $\delta_i^{(s)}$ is given by

$$\delta_i^{(s)} = (t_i^{(s)} - O_i^{(s)}) f'(\sum_j T_{ij} O_j + I_i) \tag{35a}$$

for i ε Outputs and

$$\delta_i^{(s)} = f'(\sum_j T_{ij} O_j^{(s)} + I_i) \sum_k \delta_k^{(s)} T_{ki} \tag{35b}$$

for i ε Hiddens. Equations (35a,b) define a recursive procedure for calculating ΔT_{ij} in their feed forward networks. Because of numerical problems, one can smooth the gradient by overrelaxation (c.f. Sejnowski, Ref. 16), so that Eq. (33) becomes:

$$T_{ij}(t + 1) = \alpha T_{ij}(t) + (1 - \alpha) (- \frac{\partial E^{(s)}}{\partial T_{ij}} \eta) \quad . \tag{36}$$

On the other hand, we consider a Slave network with subsets of neurons divided into Input, Hidden and Output classes with no restrictions on the connections. The evolution equation for our Slave network is

$$V_i + \mu \dot{V}_i = g(\sum_j T_{ij} V_j + I_i) \quad , \tag{37}$$

which may be written in an Euler discretized form as

$$V_i(t + \Delta t) = (1 - \frac{\Delta t}{\mu}) V_i(t) + \frac{\Delta t}{\mu} g(\sum_j T_{ij} V_j + I_i) \quad . \tag{38}$$

The units of time are arbitrary, and if we choose to use units of Δt then (38) becomes

$$V_i(t + 1) = (1 - \frac{1}{\mu}) V_i(t) + \frac{1}{\mu} g(\sum_j T_{ij} V_j + I_i) \quad . \tag{39}$$

The relation to Eq. (31) is clear given a slight change in notation. Our Lyapunov function Eq. (18a, b) is virtually identical to Eq. (32) (the integral makes a negligible contribution, but helps remove false minima[1,2]) and our "learning Equation" Eq. (19a, b) is

$$U_{ij} + \mu \dot{U}_{ij} = - \frac{\partial E_1}{\partial T_{ij}} \tag{40a}$$

$$S_i + \mu S_i = - \frac{\partial E_1}{\partial I_i} \quad . \tag{40b}$$

Note that the Euler discretized form of Eq. (40) corresponds to Eq. (36) with $\alpha = 1 - \frac{\Delta t}{\mu}$ and with g_M restricted to be a linear function. In this case a few lines of algebra using Eq. 35 show that they are identical. The constant, η corresponds to an overall constant multiplicative factor on E_1. In fact, as we pointed out in Ref. 7, the η's may actually be used to sculpt the basins of attraction by generalizing it to $\eta_i^{(s)}$ (c.f. the constants, $C_i^{(s)}$, in Ref. 7).

It is therefore clear that the two methods (programming attractors versus learning rules) are very closely related. Our evolution equation for the Slave net, Eq. 39) is a Hopfield style neural net equation and differs slightly from the evolution equation of Rummelhart (Eq. (31)) by the addition of a "forgetting" term. The fixed-point conditions for the two evolution equations are, however, identical. Rummelhart's learning equations, Eq. (33-35), are virtually identical to our Eq. (40), if the $g_M(s)$ is specialized to a linear function. The back propagation and recursive manipulations of Rummelhart are replaced by our determination of the fixed-point values of the hidden units in terms of the Input and Ouput. We allow full back propagation to the Inputs (and also recurrent Input connections) to keep the Inputs switched on to their correct values. This can replace the "clamping of inputs" performed by Rummelhart. It allows a new kind of information processing where the output may be set to, say, a "yes" value and information may flow backwards through the net to correct a corrupted input.

Because we have specified the learning problem in terms of evolution to attractors, we require that the network correctly "complete" partial information. As noted above, this is potentially quite powerful, but can be fairly delicate. The network will generally be unable to correctly "complete" partial information unless it has sufficient information to start with. In these situations, we may restrict certain connections to be feed forward, and/or clamp inputs, and perform the task in a manner similar to Rummelhart et al. Another feature of our implementation is the nonlinear form for $g_M(x)$. Restricting $g_M(x)$ to be linear (c.f. Rummelhart[9]) is possible, and changes our algorithm to a straight gradient descent procedure. A nonlinear $g_M(x)$, however, tends to smooth the energy landscape[1] and allows the possibility of annealing in β, This should be helpful in more complicated situations than those considered by Rummelhart, where false minima become a problem.

V. Summary

The Master/Slave approach[7] to controlling attractors in neural networks was extended to the case of networks with hidden neurons. Such networks are extremely important in view of their ability to perform space-bounded universal computations in a massively parallel manner. A collective method of programming such networks was developed as an extension of our original approach. A special case of this method is identical to recent work of Rummelhart[9], Parker[10] and LeChun[11]. Their results were analyzed in the broader context of "computing with attractors" of Hopfield-style neural nets. This results in a unification of these two approaches and clarifies many aspects of the algorithm, in distinction to emphasizing implementation of the algorithm.

All possible types of neural interactions were allowed, including recurrent Hidden connections, and also recurrent Input connections. For certain situations one need not clamp the Slave net Inputs thereby allowing a new kind of information processing. The "learning procedure" is replaced by the Master net evolution equation, which is identical to Rummelhart's learning equation in the case where the Master net is restricted to have a linear firing rate curve. Nonlinear (sigmoidal) Master net firing rate curves are also allowed and this helps to smooth the energy landscape resulting in fewer false minima (c.f. Hopfield[1], in another context). The basins of attraction may also be shaped by hand in the manner of Ref. 7. Computational speed is determined by the time needed to reach a fixed point of the Master Net. This is generally of order μ, which is fractions of a microsecond in hardware.

This analysis therefore results in a formalism for programming attractors in massively parallel Hopfield style neural nets that are capable of performing universal computation. It reduces to previous work of Rummelhart[9], Parker[10], and LeChun[11] in special cases, and they have demonstrated its power in small-scale model problems (see

Sejnowski[16] for applications to a larger-scale problem). Until devices can be built (either optical or VLSI) that allow for synaptic plasticity, one must restrict attention to problems where the synaptic connections can be precomputed and then hard-wired. Sejnowski's Net Talk[16] is such an example. Even in this restricted problem domain one may expect numerous applications in view of the computation universality of these networks.

References

1. J. J. Hopfield, Bio. Cyb. 52, 141 (1985).
2. J. J. Hopfield, PNAS 79, 2554 (1982), PNAS 81, 3088 (1984).
3. G. Hinton, J. Anderson, ed. "Parallel Models of Associative Memory," Erlbaum Assoc., Hillsdale, NJ (1981).
4. D. Rummelhart, J. McClelland, ed. "Parallel Distributed Processing: Exploration in the Microstructure of Cognition," MIT Press (1986).
5. S. Wolfram, Rev. Mod. Phys. 55, p. 601 (1983).
6. M. Takeda, G. Goodman "Neural Networks for Computation," Stanford Univ., E.E. Dept. preprint (1986).
7. A. Lapedes, R. Farber Proceedings: Los Alamos International Conference on Learning, Games, Evolution (May 1985) to be published, Physica D (1986).
8. J. Denker, accepted for publication, Physica D; E. Mjolsness (unpublished).
9. D. Rummelhart, J. McClelland in Ref. 4.
10. D. Parker "Learning Logic" (TR-47) MIT Center for Computational Research in Economics and Management Sciences preprint (1986).
11. Y. LeChun, Proceedings of Cognitiva 85, p. 599 (1985).
12. M. Minsky, S. Papert "Perceptrons," MIT Press (1969).
13. G. Hinton, J. Sejnowski, Proc. IEEE Soc. Conf. on Computer Vision and Pattern Recognition, Washington, D. C., p. 448 (1983).
14. D. Ackley, G. Hinton, and T. Sejnowski, Cog. Sci. 9, 147 (1985).
15. S. Kirkpatrick, C. Gelatt, M. Vecchi, Science 220 (1983).
16. T. Sejnowski and C. Rosenberg, "Net Talk: A Parallel Network that Learns to Read Aloud," Johns Hopkins EECS preprint (1986).
17. D. Psaltis, N. Farhart, Opt. Lett. 10, 98 (1985).
18. See e.g., J. Feldman, J. Cowan, Biol. Cyb. 17, 29 (1975).
19. T. Sejnowski in Ref. 3.
20. T. Kohonen "Self Organization and Associative Memory," Springer-Verlag Press (1984).