Recursively Generated Neural Networks

Eric Mjolsness, David H. Sharp, and Bradley K. Alpert

Research Report YALEU/DCS/RR-549
June 1987

Recursively Generated Neural Networks

Eric Mjolsness
Computer Science Department, Yale University

David H. Sharp
Theoretical Division, Los Alamos National Laboratory

Bradley K. Alpert
Computer Science Department, Yale University

Abstract

One difficulty with existing methods of learning neural networks is that the networks which are learned often have no regular pattern. As a result, learning large networks requires a prohibitively large search and the learned networks cannot be automatically generalized to larger sizes. When a network to perform an algorithmic computation is to be learned, however, regularity is possible and desirable. We introduce a method, the Recursive Learning Network formalism, using the ideas of divide-and-conquer and superposition, to produce hierarchically structured networks which scale automatically. Initial experiments using the formalism are described.

Existing methods of learning neural networks suffer the difficulty (Rummelhart et al, 1986; Lapedes and Farber, 1985) that the synaptic weights comprising a network which is learned may be very irregular. Even when the network is to perform an algorithmic computation, it will usually become encoded as extremely complex connections of neurons. Suppose for instance a network is to learn to count the (maximal) sequences of contiguous 1's in a binary input ("blob" counting). There are some obvious ways to wire a network for this task, each of which is very regular and can be generalized to any length input. When current learning methods are applied to this problem, however, irregular, non-generalizable networks result (Denker, 1986).

Network irregularity presents two problems: first, a very large space of possible networks must be searched to find the final network; second, the network cannot be automatically generalized to larger networks.

Both of these problems can be prevented by requiring the network to be structured. We propose a method, the Recursive Learning Network formalism, to produce hierarchically structured networks which scale automatically. In this paper we describe the formalism and initial experiments in using the formalism to learn simple networks.

Formalism

The formalism is founded on the principles of divide-and-conquer and superposition. The first principle, divide-and-conquer, is fundamental in designing many types of algorithms, while the second, superposition, has proven useful in neural network design. To illustrate how these principles are embodied in the structured representation of networks, it is simplest to jump into an example.

Consider the following matrix which represents the connections of a simple chain of neurons:

$$\left(\begin{array}{cccc|cccc}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right)$$

Here a 1 in position (i, j) denotes a connection from neuron i to neuron j. Thus the matrix represents a chain of neurons in which the first is connected to the second, the second to the third, etc. The matrix may be viewed as four quadrants such that the upper-left and lower-right resemble the entire matrix, the upper-right contains a single 1 in its lower-left corner, and the lower-left quadrant is all zeroes. If a *template* is used to define the connection matrix, we may informally write

$$\mathcal{T}_1 = \begin{pmatrix} \mathcal{T}_1 & \mathcal{T}_2 \\ 0 & \mathcal{T}_1 \end{pmatrix}, \; \mathcal{T}_1^* = 0, \quad \mathcal{T}_2 = \begin{pmatrix} 0 & 0 \\ \mathcal{T}_2 & 0 \end{pmatrix}, \; \mathcal{T}_2^* = 1. \tag{1}$$

To expand a matrix represented by template $\mathcal{T}_1$, each of the four quadrants is expanded, according to the specified template. When a quadrant consists of just one element, the template's starred or "bottom" value is used. This notion of template may be generalized somewhat so that each quadrant is expressed as a linear combination of templates with real-number weights.

A limitation of the scheme described so far is that we have assumed that a matrix is successively divided in half horizontally and vertically, which restricts us to very regular $2^k \times 2^k$ connection matrices. This restriction is removed by adding the notion of a *lineage tree*, which specifies where the divisions in the matrix rows and columns occur. In the special case above, the lineage tree is a balanced binary tree, a member of the class of all balanced trees, which share the recursive description $B(n) = \langle B(n-1), B(n-1)\rangle$ with base case $B(0) = LEAF$. (We use the notation $\langle L, R\rangle$ to mean a tree with left subtree L and right subtree R.)

Consider as another example a class of Fibonacci trees, defined by $F(n) = \langle F(n-2), F(n-1)\rangle$ and $F(1) = F(2) = LEAF$. The first few trees of this class are shown in figure 1. In generating

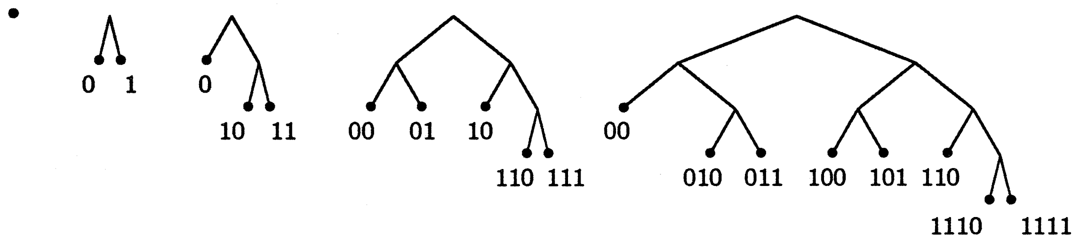


Figure 1: Fibonacci trees of size 1, 2, 3, 5, and 8, with leaf labels.

connection matrices of side length 1, 2, 3, 5, and 8, these trees may be used as lineage trees to label the entries. Rows and columns are labelled independently, with the n^{th} row or column labelled with the path to the n^{th} leaf in the tree. Sizes 3×3 and 5×5 are given as examples:

$$\begin{pmatrix} \mathcal{T}_{0,0} & \mathcal{T}_{0,10} & \mathcal{T}_{0,11} \\ \mathcal{T}_{10,0} & \mathcal{T}_{10,10} & \mathcal{T}_{10,11} \\ \mathcal{T}_{11,0} & \mathcal{T}_{11,10} & \mathcal{T}_{11,11} \end{pmatrix} \quad \begin{pmatrix} \mathcal{T}_{00,00} & \mathcal{T}_{00,01} & \mathcal{T}_{00,10} & \mathcal{T}_{00,110} & \mathcal{T}_{00,111} \\ \mathcal{T}_{01,00} & \mathcal{T}_{01,01} & \mathcal{T}_{01,10} & \mathcal{T}_{01,110} & \mathcal{T}_{01,111} \\ \mathcal{T}_{10,00} & \mathcal{T}_{10,01} & \mathcal{T}_{10,10} & \mathcal{T}_{10,110} & \mathcal{T}_{10,111} \\ \mathcal{T}_{110,00} & \mathcal{T}_{110,01} & \mathcal{T}_{110,10} & \mathcal{T}_{110,110} & \mathcal{T}_{110,111} \\ \mathcal{T}_{111,00} & \mathcal{T}_{111,01} & \mathcal{T}_{111,10} & \mathcal{T}_{111,110} & \mathcal{T}_{111,111} \end{pmatrix}$$

Matrix entry labels determine which templates apply. Before presenting an example, we introduce one more definition: if the occupant of the upper-right quadrant of $\mathcal{T}_1$ is $\mathcal{T}_2$, then we say $D^{1,2}_{0,1} = 1$. Here the subscripts refer to the quadrant row and column and the superscripts refer to the containing and contained templates. As an example, if the 5×5 connection matrix is specified by template $\mathcal{T}_2$ and the 5-leaf lineage tree, then $\mathcal{T}_{01,01} = \mathcal{T}^{(2)}_{01,01} = \sum_i D^{2,i}_{0,0}\mathcal{T}^{(i)}_{1,1}$ where i ranges over all possible templates. Here a superscript on a $\mathcal{T}$ denotes the template being expanded. Advancing one more step, $\mathcal{T}^{(i)}_{1,1} = \sum_j D^{i,j}_{1,1}\mathcal{T}^*_j$. In each step the most significant bit of the row and column labels is removed, until in the final step the template base values are used. Again, $D^{c,d}_{a,b}$ denotes the weight of template d inside template c in the quadrant with coordinates (a,b). For templates $\mathcal{T}_1$ and $\mathcal{T}_2$ above, $D^{1,1}_{0,0} = 1$, $D^{1,2}_{0,1} = 1$, $D^{1,1}_{1,1} = 1$, $D^{2,2}_{1,0} = 1$, and $D^{c,d}_{a,b} = 0$ otherwise.

The case in which the row and column labels of a matrix entry are different lengths has not been treated. What then is the value of $\mathcal{T}_{110,01}$? We pad the shorter label on the right with a new symbol, 2, until the lengths are equal, and extend the templates with the symbols $D^{c,d}_{a,2}$ and $D^{c,d}_{2,b}$, where a and b take the values 0, 1. To simplify the notation, we may further define $D^{i,i}_{2,2} \equiv \mathcal{T}_i{}^*$, $D^{i,j}_{2,2} \equiv 0$ for $i \neq j$, and $\mathcal{T}^{(i)}_{\epsilon,\epsilon} \equiv 1$, where ϵ is the empty string. Then adding a 2 to the longer label and padding the shorter label to the same length with 2's, we have $\mathcal{T}_{110,01} = \mathcal{T}_{110,012} = \mathcal{T}_{1102,0122}$. Using this convention, in general we have

$$\mathcal{T}^{(i)}_{r_0 r_1 \ldots r_n, c_0 c_1 \ldots c_n} = \sum_j D^{i,j}_{r_0,c_0}\mathcal{T}^{(j)}_{r_1 \ldots r_n, c_1 \ldots c_n}. \tag{2}$$

In the equation, j ranges over the set of templates, $r_0 r_1 \ldots r_n$ is the (extended) row label, and $c_0 c_1 \ldots c_n$ the (extended) column label. This equation may be rewritten with the recursion expanded, so

$$\mathcal{T}^{(i)}_{r_0 r_1 \ldots r_n, c_0 c_1 \ldots c_n} = \sum_{j_0} D^{i,j_0}_{r_0,c_0} \sum_{j_1} D^{j_0,j_1}_{r_1,c_1} \ldots \sum_{j_n} D^{j_{n-1},j_n}_{r_n,c_n} \tag{3}$$

In this form it is evident that the connection matrix is a generalization of a tensor product, the added generality coming from the summations. This equation is fundamental in defining the meaning of the templates and lineage trees. Through the recursive application of the templates as prescribed by the lineage tree, the formalism embodies the principle of divide-and-conquer. The summation makes it possible for several networks to be superimposed, a technique generally useful in network design.

The Recursive Learning Network formalism has great expressive power. It allows networks to contain arbitrary interconnections, including cycles. Among the networks which have been concisely expressed using the formalism are an n-dimensional cyclical grid of neurons and a feedforward base-2 to base-3 conversion network. The algebraic form for the connection matrix entries will permit conjugate gradient minimization of an objective function.

Experiments

We have begun to explore the learning of networks expressible with the formalism. We are particularly interested in producing networks to perform an algorithmic computation by training the templates on small-size inputs and then testing them on larger inputs. This approach is possible since a set of templates and a recursively-described class of lineage trees together describe a class of networks. Training on the small-size inputs is supervised through an objective function (energy function) to be minimized. Although conjugate gradient minimization is a potential search method, we have experimented with simulated annealing. In our procedure, a random change made to the templates is always accepted if it results in a decrease in energy; with an energy increase, it is accepted with probability

$$\Pr(accept \mid \triangle E \geq 0) = e^{-\triangle E/T} \tag{4}$$

where T is the *temperature*, according to the method presented by Kirkpatrick et al (1983). The temperature is varied as the simulation progresses, starting at a high value, where most changes are accepted, then slowly decreasing to a low value, where most accepted changes are very slight.

The particular task to be learned is a continuous coding task: given an input of length n consisting of 0's and exactly one 1, transform it to an output of length $2\log n$ such that when two inputs are geometrically close, their outputs are close in Hamming distance; likewise, if two outputs are close in Hamming distance the corresponding inputs should be close in geometric distance. We have tried both continuous- and discrete-valued neurons for this problem. The energy function in both versions is given as

$$E = \frac{1}{n^2}\sum_{i,j}\left[\left|\frac{i-j}{n}\right|^p - \frac{1}{2\log n} d_{Hamming}(o_i, o_j)\right]^2 \tag{5}$$

where

$$d_{Hamming}(o_i, o_j) = \sum_{\alpha}(o_{i\alpha} - o_{j\alpha})^2 \tag{6}$$

where α ranges over the code positions, and p is a parameter of the task ($0 \leq p \leq 1$). Thus the goal is a sort of gray code. A more complete description of the task is found in Mjolsness and Sharp (1986). The task is significant because it is one of a class of conversions from sparse to dense representations which preserve information. In this case, an acceptable dense representation will also be noise resistant, so that a small error in the representation produces only a small change to the value represented.

If the goal of the search were to minimize solely the task energy function, there would be no assurance that the succinctness possible with the templates would be achieved. In addition to the task energy, there should be an energy associated with the complexity of the templates found. We define the *parsimony* energy to be

$$P = \frac{1}{3^2 M^2}\sum_{a,b,c,d}\sqrt{|D^{c,d}_{a,b}|} \tag{7}$$

where M is the number of templates. The summand is intended as a continuous version of the function which is zero at 0 and one elsewhere, hence penalizes non-zero template entries. Adding the parsimony energy to the task energy increases the likelihood that the templates will be appropriate for inputs larger than those present in the training set.

Equipped with this combined energy function, the search routine is given all inputs of lengths 4, 8, 16, and 32. In addition, for these early experiments, a class of lineage trees is also supplied. The n^{th} lineage tree provides for 2^n input neurons and $2n$ output neurons and no interneurons. The inputs tree is balanced and the outputs tree is nearly balanced, adding new leaves from left to right. The recursion describing this class of trees is $A(n) = \langle B(n), C(2n)\rangle$, where $B(n)$ is defined above, and $C(n) = \langle C(k), C(n-k)\rangle$, where $k = \max\{2^{\lfloor \log_2 n\rfloor - 1}, n - 2^{\lfloor \log_2 n \rfloor}\}$, with $C(1) = LEAF$. (The definition of a simpler class of nearly balanced trees could have been $C(n) = \langle C(\lceil n/2\rceil), C(\lfloor n/2\rfloor)\rangle$, with $C(1) = LEAF$.)

After the search terminates, the discovered templates are used to produce solutions for inputs of size 64 and 128. The performance of the searches for the discrete-valued neuron case is summarized in the graphs of figure 2.

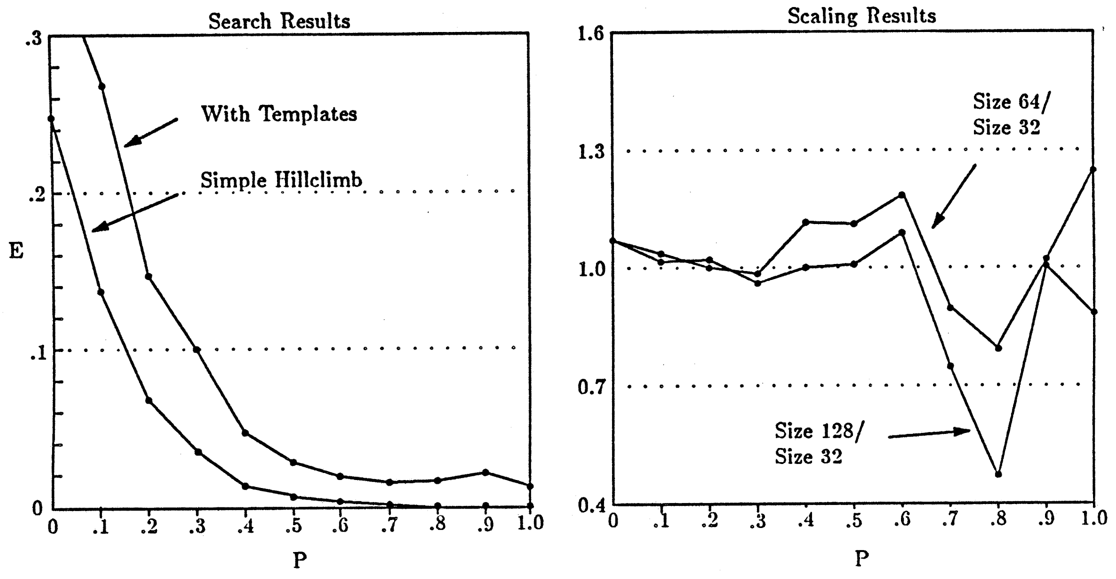


Figure 2: The left graph compares the solutions obtained by a search using the templates to one without them, by comparing the resulting energies. In the right graph, the templates show their value by scaling the solutions to larger input sizes, on which no training occurred.

The first graph shows a comparison of the result for size 32 using templates with that employing a simple hillclimbing procedure. Both lines represent the average results from three searches for each value of p. It is apparent that the hillclimbing procedure finds somewhat better solutions as measured by the energy function (though the solutions are irregular networks which have no larger-size analogs). We feel this reflects a shortcoming of the current search procedure, rather than trouble with the formalism, because we are able ourselves to describe better solutions succinctly within the formalism.

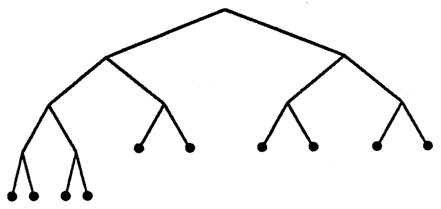
The second graph shows how well the template solutions scale up to sizes 64 and 128, by comparing the energies for those sizes with the energies for size 32. For most values of the problem parameter p, energies on the larger sizes are approximately equal or less than those for size 32. This demonstrates that although optimal solutions were not found, the solutions scale well to handle larger inputs than were in the training set.

The results from the continuous-valued neuron case were less consistent. Many of the solutions obtained were of low quality, but occasionally there were favorable surprises. For $p = 1$, we know the optimal code to take the form of a triangle, when the codes for increasing inputs are placed in sequence. For $n = 32$, this pattern would look something like

```
***
******
*********
************
***************
******************
*********************
************************
***************************
******************************
```

where a code bit of 1 is shown as * and 0 as a blank. The difficulty in general is that $2 \log n$ does not evenly divide n, so the triangle will have a jagged edge. One code actually found (twice), along with the output side of the lineage tree, looked like this:

```
********************************
  ******************************
    ****************************
      **************************
********
************
****************
********************
************************
****************************
Energy: 0.00557
```



Complementing the bits in one row does not alter the code quality, so this code is close to optimal. Observe that in the solution found, each output leaf at the greatest depth produced an "increment" of 2 and each output leaf at the next depth produced an increment of 4, or twice as much. This procedure always produces increments which sum to $n = 2^k$. A look at the code produced by the same templates for $n = 64$ confirms the pattern:

```
****
********
************
****************
********************
************************
****************************
                                ********************************
                                        ************************
                                                ****************
                                                        ********

Energy: 0.00604
```

The templates corresponding to the codes above are quite simple. Expressed in the augmented informal notation used above, they are:

$$\mathcal{T}_1 = \begin{pmatrix} \mathcal{T}_1 & \mathcal{T}_1 + \mathcal{T}_4 & 0 \\ -\mathcal{T}_1 + \mathcal{T}_4 & \mathcal{T}_1 & 0 \\ -\mathcal{T}_1 & -\mathcal{T}_1 & 11\mathcal{T}_1 \end{pmatrix}, \quad \mathcal{T}_4 = \begin{pmatrix} \mathcal{T}_4 & 0 & 0 \\ 3\mathcal{T}_1 + 3\mathcal{T}_4 & \mathcal{T}_4 & 0 \\ 0 & 0 & 6\mathcal{T}_4 \end{pmatrix} \tag{8}$$

In the search, four templates were available; only the first and fourth were used. This solution scales up indefinitely, with pretty good results. We were encouraged that this solution could be found and feel that an improved search procedure would be better at finding such solutions consistently. Apparently the simulated annealing search requires a very slow annealing schedule to work properly, slower than was practical for us. In its place we propose a search which uses simulated annealing for just the combinatorial problem of determining which template entries should be non-zero, combined with conjugate gradient minimization over the templates with specified non-zero entries. This method would also allow the discontinuous parsimony function to be used.

Summary

We introduced the Recursive Learning Network formalism to provide a language for succinctly representing connection matrices. It is based on the ideas of divide-and-conquer and superposition and has the benefit that elaborate, yet regular, connection matrices are represented with only a few parameters. This simplicity reduces the search necessary to construct networks of moderate size and makes possible automatic scaling to larger sizes. In initial experiments using the formalism to construct networks for the continuous coding problem, some quite good solutions were found for the case where the optimum is known and these scale very nicely. The consistency of solutions found could be improved, though, with a search procedure combining simulated annealing with conjugate gradient minimization. We are quite optimistic that with this better search procedure, the Recursive Learning Network formalism will enable the discovery of quality solutions that continue to scale well, a goal which has not be achieved by methods lacking structured representations.

Bibliography

Denker, John, in talk presented at Institute for Theoretical Physics, Santa Barbara, California, December, 1986

Kirkpatrick, S., C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by Simulated Annealing," Science, vol. 220, p. 671, May 13, 1983

Lapedes, Alan, and Robert Farber, "Programming a Massively Parallel, Computation Universal System: Static Behavior," Los Alamos National Laboratory, LAUR 86-1179, 1986

Mjolsness, Eric, and David H. Sharp, "A Preliminary Analysis of Recursively Generated Networks," in *Neural Networks for Computing*, John Denker, editor; American Institute for Physics, 1986.

Rummelhart, D. E., G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (Volume I), MIT Press, 1986