Multiscale Optimization in Neural Nets

Eric Mjolsness, *Member, IEEE,* Charles D. Garrett, and Willard L. Miranker

***Abstract*—One way to speed up convergence in a large optimization problem is to introduce a smaller, approximate version of the problem at a coarser scale and to alternate between relaxation steps for the fine-scale and coarse-scale problems. We exhibit such an optimization method for neural networks governed by quite general objective functions. At the coarse scale there is a smaller approximating neural net which, like the original net, is nonlinear and has a nonquadratic objective function. The transitions and information flow from fine to coarse scale and back do not disrupt the optimization, and the user need only specify a partition of the original fine-scale variables. Thus the method can be applied easily to many problems and networks. We show positive experimental results including cost comparisons.**

I. Introduction

A RATHER general neural net objective function for continuous neural variables v_i is composed of a general sum of linear, quadratic, and cubic terms, by which neurons are connected and can interact with each other, along with nonlinear potential functions of just one neural variable each [1]. All higher order polynomial objectives can be reduced to this form [2]. If the cubic and potential function terms are absent, then the objective is quadratic and analogous to many numerical problems, defined on geometric domains, for which multigrid methods are successfully used as fast relaxation algorithms [3], [4]. Such algorithms proceed by introducing a smaller, approximate version of the problem at a coarser scale (i.e., using a coarser mesh) and alternating between relaxation steps for the fine-scale and the coarse-scale problem. This is done recursively, at many scales, and it is the two-way passage of information between scales which is responsible for the unusual effectiveness of the technique. Previously, multigrid methods have been applied to nonlinear objective functions arising in computer vision which are close to being neurally optimizable [5], [6], but the coarse-scale objective functions were taken to be quadratic approximations of the full objective functions. In this paper we generalize the multigrid approach and propose a multiscale relaxation method which does not require an underlying geometric domain and which incorporates the cubic and potential function nonlinearities of the neural net at all scales if those terms are present in the original fine-scale problem. Thus, we define and explore a multiscale optimization method appropriate for neural nets.

Manuscript received July 3, 1990; revised October 29, 1990. This work was supported in part by AFOSR Grant 88-0240.

E. Mjolsness and C. D. Garrett are with the Department of Computer Science, Yale University, P.O. Box 2158 Yale Station, New Haven, CT 06520-2158.

W. L. Miranker is with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598, and with the Department of Computer Science, Yale University, New Haven, CT 06520.

IEEE Log Number 9041475.

II. Theory

The general neural net objective function that we will use for continuous neural variables v_i is [1]

$$E[\boldsymbol{v}] = -\sum_{ijk} T_{ijk} v_i v_j v_k - \sum_{ij} T_{ij} v_i v_j - \sum_i h_i v_i + \sum_i \phi_i(v_i), \tag{1}$$

although many networks are designed without the cubic term $T_{ijk} v_i v_j v_k$. The potential function term is $\phi_i(v_i)$. We may consider the neural variable v_i to be the ith component of a vector devoted $\boldsymbol{v}$. The desired multiscale neural network is illustrated schematically in Fig. 1. To obtain such a design, let us consider the usual multigrid method for optimization.

The constituents of the multigrid method are (1) a map from the original variables v_i to fewer coarse-scale variables V_a, called the restriction or aggregation map $V = R[\boldsymbol{v}]$; (2) a map from coarse scale to fine scale, called the prolongation or disaggregation map $\boldsymbol{v} = P[V]$; (3) a coarse-scale objective function $\hat{E}[V]$ which is intended to approximate $E[\boldsymbol{v}]$ although $\hat{E}[V]$ is cheaper to evaluate and differentiate; (4) an algorithmic cycle by which $E[\boldsymbol{v}]$ is partially relaxed to produce a point $\boldsymbol{v}$, then $\boldsymbol{v}$ is restricted to produce V, then $\hat{E}[V]$ is partially relaxed by updating V, then V is prolonged to produce a new value of $\boldsymbol{v}$ for relaxation under E again; and (5) modification of the basic cycle to handle many scales recursively. We must supply versions of these constituents suitable for the neural nets. The prolongation map and the coarse-scale objective are particularly important.

The choice of $\hat{E}$ will be very simple for us: it is the restriction of E to a subspace parameterized by V. The subspace is given by the prolongation map P:

$$\hat{E}[V] = E[P[V]]. \tag{2}$$

Thus, relaxation of $\hat{E}[V]$ is equivalent to relaxation of $E[\boldsymbol{v}]$ in a subspace, namely the range of $P[V]$. This form of $\hat{E}$ will be modified slightly in subsection II-B (Eq. (13)).

One special case of (2) covers much of what is done under the name of multigrid methods. Assume $E[\boldsymbol{v}]$ is purely quadratic and that the maps P and R are matrix–vector multiplications; that is, they are linear and homogeneous maps ($P[V] = PV$ and $R[\boldsymbol{v}] = R\boldsymbol{v}$). If $R = P^T$, we get a form of $\hat{E}$ which occurs often in multigrid problems:

$$\begin{aligned}\hat{E}[V] &= E_{\text{quadratic}}[P[V]] \\ &= \sum_{ij} T_{ij} \left(\sum_a P_{ia} V_a\right)\left(\sum_b P_{jb} V_b\right) \\ &= \sum_{ab} (RTP)_{ab} V_a V_b. \end{aligned} \tag{3}$$

Thus, there is a corresponding quadratic form on the coarse scale with matrix $\hat{T} = RTP$. If P and R are fixed, the reduced-dimen-

1045-9227/91/0300-0263$01.00 © 1991 IEEE

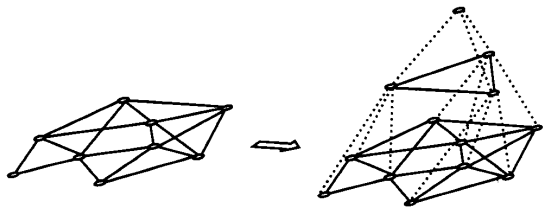


Fig. 1. Multiscale network. An ordinary neural network, depicted on the left in this figure, may be governed by an objective function $E[v]$ with numerous simple interactions (connections) between many nonlinear neurons (circles). Such a network may be transformed into a multiscale network, shown here on the right, by the addition of smaller and cheaper approximating networks at successive scales, with associated objective functions.

sion connection matrix $\hat{T}$ can be computed once and then used throughout the multigrid procedure. Of course if T is sparse, then the cost advantage of relaxing at the coarse scale requires that R and P be sparse as well, so that $\hat{T}$ will not suffer much fill-in. Since $\hat{T}$ is smaller than T, some fill-in can be tolerated.

The simplifying assumption $R = P^T$ describes the bulk of existing multigrid methods and it simplifies the analysis associated with them. For example if T is Hermitian and $R = P^T$, then $\hat{T}$ will also be Hermitian. Note, however, that we will avoid this assumption in the next section and use another description of R which is equally compatible with (2).

A. The Prolongation Map

Our problem now is to design the prolongation map $P[V]$. One might design it separately for each objective function E, so that the prolongation allows $\hat{E}$ to approximate E. For example, prolongations for canonical problems on two-dimensional grids have been extensively studied. This is very expensive in the designer's time. Or one might automatically learn an effective P for a particular E by optimizing some measurement of the degree to which $\hat{E}$ approximates E over a training set (after all, we are studying neural nets), but that involves quite an escalation of computational cost and must happen on a slow time scale: P would be nearly constant during one multiscale optimization. We look for a less complex and less general choice of P. Our choice of P is intended to require relatively little effort on the part of the user of the multiscale procedure.

At the other extreme in complexity for P, one could just take $P[V] = BV$, where B is a 0–1 matrix representing the partition of the v_i variables into blocks each of which is summarized by V_a. (Note that $\Sigma_a B_{ia} = 1$ since, in our treatment, we require that each fine-scale neuron is a member of exactly one block of the partition. We will exploit this fact in equation (10) of subsection II-B). This scheme would be very inexpensive since B is very sparse, and it seems reasonable to request the user of the multiscale algorithm to guess a relevant partition of the variables. The problem with this simple scheme is that B is a 0–1 partition matrix, so when a particular coarse-scale variable V_a is updated there is a corresponding motion of v induced by P. This motion of v is along the $(1, 1, \cdots, 1)$ direction within block a of the partition of the variables $\{v_i\}$. So, within each block the $(1, 1, \cdots, 1)$ direction is always favored as the relaxation direction during coarse-scale relaxation, though for most objective functions E that particular direction is without special merit.

We therefore propose a compromise between generality and cost in the choice of P. We will partition the variables and define P so that the coarse-scale relaxation corresponds to fine-scale relaxation along the initial gradient direction of each partition block. That way, each block has just one degree of freedom during coarse-scale relaxation, but its direction is well chosen. This idea is expressed mathematically as follows. Letting the matrix B correspond to a user-supplied partition of the original variables, take

$$\begin{aligned} v_i &= v_i^0 + \Sigma_a P_{ia} V_a \\ P_{ia} &= B_{ia} z_i(v^0) \\ z(v) &= -\nabla_v E \end{aligned} \tag{4}$$

i.e.,

$$P[V] = v_i^0 - \sum_a B_{ia} \left(\frac{\partial E}{\partial v_i} \right)_{v^0} V_a. \tag{5}$$

(C.f. the form of the prolongation matrix P_{ia} in [7].) Notice that the prolongation (5), while linear, is not homogeneous. Here v^0 is the value of v obtained by the last fine-scale relaxation of E, just before the coarse-scale relaxation of $\hat{E}$. Note that v^0 is the image of $V = 0$ under $P[V]$. That is, we center our coarse-scale coordinate system so that the origin corresponds to the most recent fine-scale vector, v^0. Consequently there is no change in the value of E in the actual restriction step of the multiscale method. The restriction step may be written out as

$$\begin{aligned} v_i^0 &= v_i \\ V_a &= 0. \end{aligned} \tag{6}$$

Remark 1: Not only is there no change in E during the restriction step; after coarse-scale relaxation occurs, (2) guarantees that there is no change in E in the prolongation step of the multiscale method either. So E changes only during the fine-scale and coarse-scale relaxations. The conventional multigrid method, by contrast, may suffer an increase of E during the restriction or prolongation steps.

A second property of (5) is that the coarse-scale relaxation may be understood in terms of gradient descent on the fine scale. Each block of the partition relaxes in the direction specified by the projection of the gradient z onto that block's subspace in v; in other words, each block a undergoes its particular gradient descent, with V_a as the parameter describing the distance moved along the projection of the descent direction within that block. Initially $V_a = 0$ for all blocks. Notice that the prolongation map $P[V]$ adapts dynamically as the optimization of E proceeds.

Some analytical results concerning the reduction of error available at the coarse scale under (2) and (5) are presented in Appendix I. Experimental results on the behavior of the entire multiscale network will be presented in Section IV. In the remainder of this section we consider the computational cost of the multiscale network.

The proposed $P[V]$ is inexpensive because B is very sparse: it has just one nonzero entry per fine-scale variable (i.e., per row). On the other hand, P must be recomputed on successive cycles of the multiscale method—whenever an intervening fine-scale relaxation step alters v^0 and therefore the gradient $z(v^0)$ as well. So, unlike the coarse-scale network connection matrix RTP discussed earlier, the coarse-scale matrix here must be re-

computed for each aggregation phase of coarse-scale relaxation:

$$\begin{aligned}\Sigma_{ij}\, T_{ij} P[V]_i P[V]_j &= \Sigma_{ij}\, T_{ij}\left(\Sigma_a\, B_{ia} z_i(\boldsymbol{v}^0) V_a\right)\left(\Sigma_b\, B_{jb} z_j(\boldsymbol{v}^0) V_b\right) \\ &\quad + \text{linear terms} \\ &= \Sigma_{ab}\, \hat{T}_{ab} V_a V_b + \cdots\end{aligned}$$

where

$$\hat{T}_{ab} = \Sigma_{ij}\, B_{ia} B_{jb} T_{ij} z_i(\boldsymbol{v}^0) z_j(\boldsymbol{v}^0). \tag{7}$$

The cost of recomputing $\hat{T}_{ab}$ and the rest of $\hat{E}$ is small compared with the cost of the fine-scale partial relaxation which immediately proceeds it. This network construction cost is due to the variable $\boldsymbol{v}^0$ and $\boldsymbol{z}(\boldsymbol{v}^0)$ vectors only. B remains constant and defines a fixed topology for the construction algorithm, which can be considered to be a feedforward neural net that computes $\hat{T}$ according to (7). In comparison, the fine-scale relaxation net for E has about the same number of connections but it contains feedback and requires many iterated relaxation steps. Even for a hard-wired circuit implementation, the wiring cost of recomputing $\hat{T}_{ab}$ would about the same as that of the fine-scale relaxation and therefore in balance with it.

The cubic, quadratic, and linear terms of $\hat{E}$ may be computed from the corresponding terms of E as in (7). It remains to consider the effect of the prolongation map $P[V]$ on E's single-neuron potential term, $\Sigma_i\, \phi_i(v_i)$.

B. The Potential Term

Low-order polynomial summands in $\phi_i(v_i)$ can be efficiently transferred to the rest of the objective; assume this has been done. We confine our discussion to singularities in ϕ_i, such as those of the barrier functions that correspond to sigmoidal neural transfer functions $v = g(u)$ through $g^{-1}(v) = \phi'(v)$ [1]. (See Fig. 2 for a typical barrier function expressed as a sum of two singular functions $\phi(v) = \phi_-(v) + \phi_+(v)$.) It may be possible to handle some nonsigmoidal neural transfer functions in a similar way.

We will assume a restricted form of the potential term's dependence on the neuron index i:

$$\begin{aligned}E_\phi[\boldsymbol{v}] = \sum_i \phi_i(v_i) = \sum_i \big[&c_{i-}\phi_-(a_{i-}v_i + b_{i-}) \\ &+ c_{i+}\phi_+(a_{i+}v_i + b_{i+})\big] \quad (a_{i\pm} \geq 0)\end{aligned} \tag{8}$$

where $\phi_-(w)$ has a singularity at the origin and provides an infinite penalty for negative values of w, so that relaxation algorithms are restricted to positive w; similarly $\phi_+(w) = \phi_-(-w)$ restricts its argument to negative values. For example, one could take $\phi_-(w) = w^{-p}$ or $-(1/2)\log w$ for $w > 0$. We assume $-b_{i-}/a_{i-} \leq 0 \leq -b_{i+}/a_{i+}$ for consistency, so that there is an allowable region of v_i. Equation (8) imposes a standard form for $\phi_i(v_i)$ (in terms of $\phi_\pm$) that can easily be generalized to a small number of standard "species" of potentials, $\phi_\pm^\lambda$, indexed by λ, one of which is used for each neuron.

Using (2) directly for the potential term in $\hat{E}$ is too expensive for a multiscale method but can be simplified as follows:

$$\begin{aligned}E_\phi[P[V]] = \sum_i \Bigg[&c_{i-}\phi_-\Bigg(a'_{i-}\sum_a B_{ia}V_a + b'_{i-}\Bigg) \\ &+ c_{i+}\phi_+\Bigg(a'_{i+}\sum_a B_{ia}V_a + b'_{i+}\Bigg)\Bigg]\end{aligned} \tag{9}$$

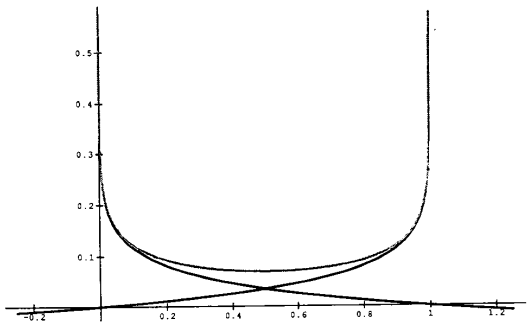


Fig. 2. The barrier function $\phi(w)$ shown as a sum of two singular functions $\phi_-(w)$ and $\phi_+(w)$ which have singularities at $w = 0$ and $w = 1$, respectively.

where $a'_{i\pm} = a_{i\pm}z_i^0$ and $b'_{i\pm} = a_{i\pm}v_i^0 + b_{i\pm}$. Using the fact that B is a partition matrix (as discussed in the previous section), (9) may be written as

$$\begin{aligned}E_\phi[P[V]] = \sum_{ia} \big[&B_{ia}c_{i-}\phi_-(a'_{i-}V_a + b'_{i-}) \\ &+ B_{ia}c_{i+}\phi_+(a'_{i+}V_a + b'_{i+})\big].\end{aligned} \tag{10}$$

Since $a_{i\pm} \geq 0$, a'_{i-} and a'_{i+} have the same sign. If they are negative we can change their signs using $\phi_+(w) = \phi_-(-w)$, in the process interchanging the roles of ϕ_+ and ϕ_-. Then we obtain

$$\begin{aligned}E_\phi[P[V]] = \sum_{ia} \big[&B_{ia}\tilde{c}_{i-}\phi_-(\tilde{a}_{i-}V_a + \tilde{b}_{i-}) \\ &+ B_{ia}\tilde{c}_{i+}\phi_+(\tilde{a}_{i+}V_a + \tilde{b}_{i+})\big] \quad (\tilde{a}_{i\pm} \geq 0)\end{aligned} \tag{11}$$

where

$$\begin{aligned}\tilde{a}_{i\pm} &= \begin{cases} a'_{i\pm} & \text{if } a'_{i-} > 0 \\ -a'_{i\mp} & \text{otherwise} \end{cases} \\ \tilde{b}_{i\pm} &= \begin{cases} b'_{i\pm} & \text{if } a'_{i-} > 0 \\ -b'_{i\mp} & \text{otherwise} \end{cases} \\ \tilde{c}_{i\pm} &= \begin{cases} c'_{i\pm} & \text{if } a'_{i-} > 0 \\ c'_{i\mp} & \text{otherwise.} \end{cases}\end{aligned} \tag{12}$$

Equation (11) has a form similar to its fine-scale counterpart, (8), but takes just as many ϕ evaluations to compute. Fortunately there is another strategy available. To guarantee a favorable result for a phase of coarse-scale relaxation, (2) is not necessary. It suffices to choose $\hat{E}$ so that

$$\hat{E}[V] \geq E[P[V]] \tag{13a}$$

and

$$\hat{E}[\mathbf{0}] = E[P[\mathbf{0}]]. \tag{13b}$$

Subtracting (13a) from (13b) gives

$$0 \geq \Delta\hat{E} \geq \Delta E[P[V]] \tag{14}$$

so relaxation in $\hat{E}$ implies *at least as much relaxation* in E when V is prolonged back to $\boldsymbol{v}$ (see Remark 1 of subsection II-A).

We establish (13a) and (13b) for each summand $\phi_\pm$ of $E_\phi[P[V]]$ in (11). The idea is to bound all the ϕ_- terms in one partition block by the ϕ_- term whose singularity is *closest* to

the initial value $V_a = 0$, and likewise with the ϕ_+ terms. So we look for a function $\hat{\phi}$ for which

$$\tilde{c}_{i-}\phi_-(\tilde{a}_{i-}V_a + \tilde{b}_{i-}) \le \hat{\phi}_{i-}(V_a) \equiv C_{i-}\phi_-(\hat{a}_{a-}V_a + \hat{b}_{a-}) + D_{i-} \quad (15a)$$

$$\tilde{c}_{i-}\phi_-(\tilde{b}_{i-}) = \hat{\phi}_{i-}(0) \equiv C_{i-}\phi_-(\hat{b}_{a-}) + D_{i-} \quad (15b)$$

where $-\hat{b}_{a-}/\hat{a}_{a-} = \max_{i\in\pi(a)} -\tilde{b}_{i-}/\tilde{a}_{i-}$; $\pi(a)$ denotes the set of i indices in the ath partition block; and C_{i-} and D_{i-} are to be adjusted to satisfy equations (15). Then summing over i and a as in (11) shows that (15) implies (13). Incidentally, the $\max_{i\in\pi(a)}$ operation is neurally implementable by means of a winner-take-all subnet [2]. For a maximum over n variables, $O(n)$ connections and time proportional to $O(\log n)$ [8] is typically required by such an analog neural network.

For some potentials ϕ_- (e.g., $\phi_-(w) = w^{-p}$ or $-(1/2)\log w$), equations (15) can be ensured by demanding equality of the two functions and their V_a derivatives at $V_a = 0$ and solving for C_{i-} and D_{i-}:

$$C_{i-} = \tilde{c}_{i-}\tilde{a}_{i-}\phi'_-(\tilde{b}_{i-})/[\hat{a}_{a-}\phi'_-(\hat{b}_{a-})]$$

$$D_{i-} = \tilde{c}_{i-}\phi_-(\tilde{b}_{i-}) - C_{i-}\phi_-(\hat{b}_{a-}). \quad (16)$$

Then

$$\begin{aligned}\hat{E}_\phi[V] &= \Sigma_{ia} B_{ia}[\hat{\phi}_{i-}(V_a) + \hat{\phi}_{i+}(V_a)] \\ &= \Sigma_a(\Sigma_i B_{ia}C_{i-})\phi_-(\hat{a}_{a-}V_a + \hat{b}_{a-}) + \Sigma_i D_{i-} \\ &\quad + \Sigma_a(\Sigma_i B_{ia}C_{i+})\phi_+(\hat{a}_{a+}V_a + \hat{b}_{a+}) + \Sigma_i D_{i+} \\ &\equiv \Sigma_a[\hat{c}_{a-}\phi_-(\hat{a}_{a-}V_a + \hat{b}_{a-}) \\ &\quad + \hat{c}_{a+}\phi_+(\hat{a}_{a+}V_a + \hat{b}_{a+})] + \hat{d} \quad (\hat{a}_{i\pm} \ge 0) \quad (17)\end{aligned}$$

which is the coarse-scale version of (8). Notice that calculating $\hat{E}_\phi$ now requires as many evaluations of ϕ as there are coarse-scale variables, not fine-scale variables; thus the cost of the coarse-scale neural net has become affordable for a multiscale method.

III. Discussion

Having presented the proposed multiscale optimization method for neural networks, we now make a few observations about it before presenting experimental results in Section IV.

A. Benefits

We have presented a very conservatively designed multiscale method: regardless of the particular optimization problem being solved, the method can be applied and the restriction, prolongation, and coarse-scale relaxation steps are each guaranteed to have $\Delta E[v] \le 0$ so that they at least do no harm to the fine-scale minimization process. Also, we showed that the cost of the method is low. The potential benefit is in speedier convergence: if the coarse-scale relaxations make a significant amount of progress in minimizing E, they can be called upon to do most of the work at very little computational cost. For quadratic objectives (i.e., solving linear systems of equations) there are several techniques for proving that some speedup will occur when a multigrid technique is used. When the problem is derived from a partial differential equation on a spatial domain, the modal techniques of Fourier analysis are often used to prove speedup [9]. Much weaker assumptions are required by more recent speedup proofs [10], though a quadratic objective is still assumed. We know of no such proof for our method; it must simply be tried out. But in Appendix I we derive an analytic expression for the reduction in error obtained by complete relaxation at the coarse level:

$$\varepsilon^1 = (I - \Pi + \lambda\Pi J[\hat{v}])^{-1}(I - \Pi)\cdot\varepsilon^0 \quad (18)$$

where Π is a projection operator, J is the Jacobian of the objective at a certain point $\hat{v}$, and λ is any nonzero real number.

Spatial-domain multiscale techniques can also lead to better local minima for problems with nonquadratic and multimodal objective functions; for example, scale-space continuation methods in computer vision may have this desirable property [11], [12]. We do not expect such an improvement in the local minima reached by our multiscale method because it never takes any uphill steps in the original, multimodal E. This suggests using the multiscale method to speed up convergence within a continuation method for minimizing E.

B. Saddle Points

As mentioned in the Introduction, any polynomial summand of an objective function can be reduced to a cubic polynomial, so (1) is rather general [2]. But this reduction occurs at the expense of replacing minima with saddle points which have the characteristic that each variable is classified ahead of time as requiring maximization or minimization. Thus one can consider a two-phase algorithm for finding such saddle points: alternately maximize E with respect to all the maximization variables, then minimize E with respect to all the minimization variables, and iterate. For each phase one can use the multiscale method we have presented to speed up the calculation. But the number of max/min cycles required for convergence may be large. Alternatively one could seek saddle points rather than minima or maxima within the multiscale algorithm, but this would destroy the built-in property that the restriction, prolongation, and coarse-scale relaxation steps do not undo any of the optimization progress made at the fine scale. So the algorithm may be less effective on saddle point problems than on minimization problems.

C. The Partition, B_{ia}

One may obtain the required 0/1 partition matrix B_{ia} in several ways. For example, one might preprocess the original network of (1) so as to group together neurons that are strongly connected (large $|T_{ij}|$). Finding the best possible graph partition is an NP-complete problem [13], but nearly optimal partitions may be found by another optimizing neural net such as the graph-partitioning networks studied in [14] and [15]. A particularly cheap (and approximate) way to do this is to bisect the net into two "modules" with minimal intermodule connections and recursively bisect the modules. So we must consider the relative costs of the preprocessing stage, involving a non-multiscale neural net for graph partitioning, and the accelerated multiscale neural net. An expensive preprocessing step is allowed if the resulting partition will be used unchanged on many different runs of the multiscale net. If the amortized cost of such preprocessing is still regarded as too high for its benefits, one might simply guess a partition of the neurons based for example on a partition of some spatial domain loosely associated with the net, as we do in the experiments reported in Section IV. Finally, one might attempt to learn an effective partition through experience in repeatedly running the network for different prob-

lems and optimizing B on a slow time scale while respecting its sparseness.

D. An Alternate Network Notation

We will not consider in great depth the question of implementing our multiscale neural nets as analog circuits or other special-purpose hardware. But a slight change of notation can bring this question into the domain of a circuit-design method that uses rewrite rules by which one objective function can be algebraically transformed into another, more implementable one [2]. We will describe this notation in detail in Appendix II.

IV. Experimental Results

We have applied the proposed multiscale optimization techniques to several nonquadratic objective functions. To illustrate the method's independence of a continuous spatial domain, we used an objective function for inexact graph matching based on purely structural similarity of two graphs [16], [17]. This problem may have application to problems of model matching in high-level computer vision. The objective is related to the traveling salesman problem objective of [18], and both suffer from a strong increase in the number of local minima as the problem size increases. So we also considered a less problematical but spatially structured nonquadratic objective function from low-level vision [19]. It may be used for smooth two-dimensional surface reconstruction from sparse data, modified by nonlinear discontinuity detection processes, all defined on a discretized two-dimensional grid.

The inexact graph-matching problem is defined by two 0/1 incidence matrices g and G which specify the graphs to be matched, and the answer is a sparse variable 0/1 matrix M whose elements specify a permutation of nodes of g onto nodes of G. The objective function for graph matching is taken to be

$$E[M] = \frac{A}{2}\Sigma_i(\Sigma_\alpha M_{\alpha i} - 1)^2 + \frac{A}{2}\Sigma_\alpha(\Sigma_i M_{\alpha i} - 1)^2 + \frac{B}{2}\Sigma_{\alpha i}(1 - M_{\alpha i})M_{\alpha i} - C\Sigma_{\alpha\beta ij} G_{\alpha\beta} g_{ij} M_{\alpha i} M_{\beta j} + \Sigma_{ij}\phi_s(M_{ij}) \quad (19)$$

where

$$\phi_s(x) = -\frac{1}{2g_0}[\ln x + \ln(1 - x)] \quad (20)$$

and $A = 30$, $B = 1$, $C = 3$, and $g_0 = 10$. The first two terms favor unique matches between nodes in the two graphs; the B term favors $M \approx 0$ and $M \approx 1$ over the intervening values $M \in (0, 1)$; the C term favors consistent matches between neighboring nodes in the two graphs; and the final term restricts each M variable to the interval $(0, 1)$ by raising infinite barriers at the border. See Fig. 3 for an example of a graph-matching neural net corresponding to this objective, including graph nodes and links as well as the neurons and connections in the network.

For matching two n-node graphs of constant degree, this objective function has n^2 variables (neurons) and $O(n^3)$ monomial interactions, each corresponding to a connection in the neural net which we simulated. It is known how to reduce this to $O(n^2)$ connections in a saddle-point objective function [2] (with different temporal behavior). But to avoid the complications of studying the multiscale algorithm in the context of saddle points, we will compare the multiscale and single-scale versions of the $O(n^3)$ network just described. The multiscale method requires a partition of the $M_{\alpha i}$ variables. We used the outer product of partitions of the graph nodes in G and g, which are indexed by α and i respectively. We chose the graph partitions heuristically, intending to minimize the number of graph links that cross the partition boundaries.

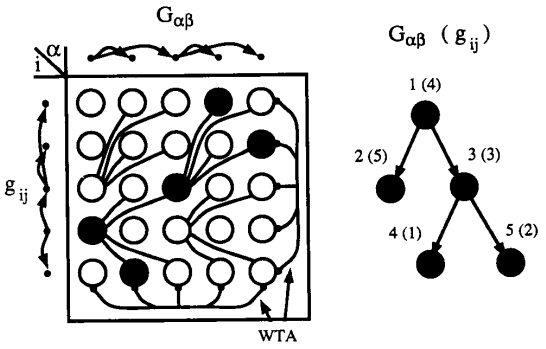


Fig. 3. Left, neurons and connections in the graph-matching problem corresponding to the two graphs consisting of two different labelings of the tree graph on the right. The matrix of neurons (circles) are match neurons $M_{\alpha i}$, shaded for those neurons that should be "on" (should have value near unity) in the correct final configuration. Each row and column has a winner-take-all subnet (indicated schematically for the last row and column and labelled "WTA"). Additional connections arise from the graph-matching term, in which neuron $M_{\alpha i}$ is connected to neuron $M_{\beta j}$ if and only if both $G_{\alpha\beta} = 1$ and $g_{ij} = 1$. Unlike the tree and two-dimensional graphs used in our experiments, the isomorphic tree graphs shown here are directed graphs.

Two families of arbitrary-size graphs were considered. First, we constructed a size-n nearly balanced tree with incidence matrix g and the same tree under a different labeling of the nodes with incidence matrix G; in this case the graph partition was obtained by grouping together equal-length segments of the chain of nodes resulting from an in-order traversal of the tree. Second, we considered the sparse two-dimensional graphs of [20], obtained by independently choosing n points from the unit square with the uniform probability distribution and connecting up all points within a distance d determined by the requirement that the average degree of connectivity be given by $4nd^2 = 3$. This graph was partitioned by a uniform rectangular subdivision of the unit square. G was obtained similarly, after a randomly selected 20% of the points had been displaced by random vectors with x and y components between -0.1 and $+0.1$. In this case g and G were often not exactly isomorphic. In Fig. 4 we show an example of two isomorphic tree graphs with different labelings, and also two nonisomorphic two-dimensional graphs which the network may be employed to match.

The objective function for two-dimensional surface interpolation with discontinuities was

$$E[f, l] = A\Sigma_{ij}(1 - l^v_{ij})(f_{i+1,j} - f_{ij})^2 + A\Sigma_i(1 - l^h_{ij})(f_{ij+1} - f_{ij})^2 + B\Sigma_{ij}(f_{ij} - d_{ij})^2 + C\Sigma_{ij}(l^h_{ij} + l^v_{ij}) + \Sigma_{ij}(\phi_s(l^h_{ij}) + \phi_s(l^v_{ij})) \quad (21)$$

where d is the data, f is the real-valued interpolated function, and l^h and l^v form a set of 0/1 variables (called line processes) indicating the presence of a discontinuity of the reconstructed surface (the f's) in the horizontal or vertical directions. The parameters were $A = 1$, $B = 1$, $C = 0.1$, $D = 1$, and $g_0 = 100$. The first two terms in E favor smoothness of the recon-

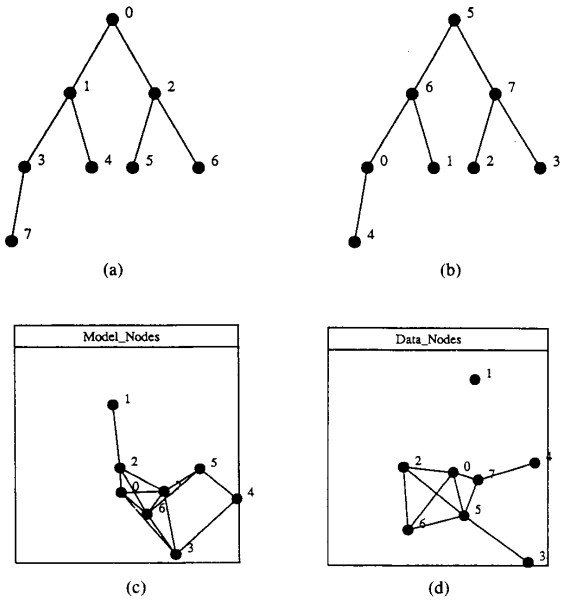


Fig. 4. In (a) and (b) are shown two isomorphic binary tree graphs with different labelings. The graph-matching network of Fig. 3 may be employed to match them. In (c) and (d) we display two nonisomorphic two-dimensional graphs, generated as described in the text. Part (c), labeled model nodes, represents a possible structural model of an object stored in a library for use in model-based vision; in this application (d), labeled data nodes, would be a graph derived from image data.

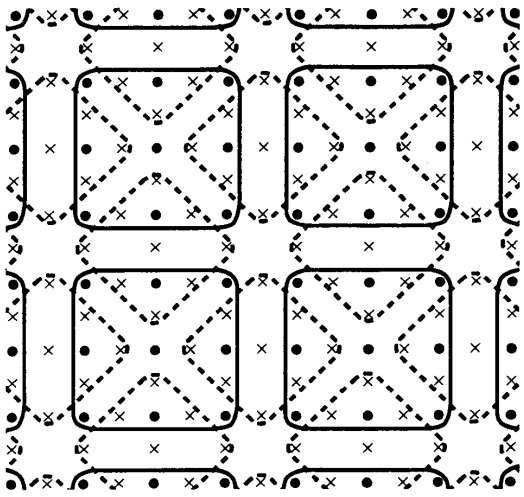


Fig. 5. Partition of variables used in the multiscale version of the surface reconstruction network. Dots represent f_{ij} neurons in the plane. Crosses represent the associated horizontal and vertical "line process" neurons l^h_{ij} and l^v_{ij}. Each l neuron is placed symmetrically between the two f neurons with which it interacts. All these neurons occur in the original net and therefore at the finest scale of the multiscale net. The f neurons are aggregated by means of a partition, here into 3×3 blocks with solid outlines. (the block size could be any integer; 3 is just an example.) The l neurons are aggregated into their own 3×3 partition blocks, separate from those of the f neurons, as shown by the dotted outlines. Both the l neurons and, at the larger scale, their partition blocks, occur on lattices tilted at 45° to the f_{ij} lattice. In this way the fine-scale network structure is reproduced at the coarse scale.

structed function f in the horizontal and vertical directions, unless interrupted by a line process variable l. The B term favors consistency between f and the original data d. The C term penalizes a large number of active line processes or discontinuities. And the final term again restricts l to $(0, 1)$. The f and l variables were partitioned differently; since they occur on the sites and links (respectively) of a fine-scale grid, their partition blocks were chosen to occur on the sites and links (respectively) of a coarse-scale grid. This partition scheme is shown in Fig. 5.

In principle, the method imposes no restrictions on the block size used to partition the variables; the 3×3 blocks used in Fig. 5 are merely illustrative. In conventional multigrid investigations, 2×2 blocks in a deep hierarchy often result in the fastest convergence. So far we have had better results with a shallow (three-level) hierarchy; future experiments with larger problem sizes may reveal an optimal block size.

Fig. 6 shows the reconstruction of a simple discontinuous surface by this neural network, starting from random values for the f neural variables and small uniform values for the l variables.

Another multiscale method has been applied to the minimization of this objective by Terzopoulos [5], [6]. The differences are illuminating: Terzopoulos's algorithm turns the nonquadratic objective into a quadratic one by holding the line process variables l fixed, as control parameters. A conventional multigrid method performs the quadratic optimization of the interpolation variables f; then in a separate phase the fine-scale line processes are relaxed as continuous or discrete variables, and the two-phase cycle is iterated. By contrast our procedure includes both sets of variables in the multiscale algorithm, relaxing both simultaneously at any given scale. It has the drawback of greater complication in the restriction and prolongation steps. In our method the structure of the original objective is more faithfully preserved at the coarse scale, but the resulting coarse-scale objective function is not as easy to minimize as a quadratic approximation would be. A detailed comparison of the relative merits of the two multiscale algorithms for the surface interpolation problem is beyond the scope of this paper, but we speculate that the relative performance depends on the number and nature of the reconstructable discontinuities in the data d_i. These interesting issues could be explored in further work.

In our experiments the numerical relaxation step at any given scale consisted of repeated univariate minimization along the gradient direction (a "line search") [21]. This strategy is standard in parallel optimization algorithms but slightly different from the continuous steepest methods often used in analog neural nets. Rather than continuously computing the gradient direction and moving the current state vector in that direction, a line search holds the state vector fixed while the gradient is computed, and then the descent direction is held fixed while a continuous displacement along that direction is calculated and taken. Such a two-phase minimization procedure could be implemented as continuous steepest descent in a clocked objective function similar to that described in Appendix II.

Also, the control scheme we used for the multiscale relaxation algorithm is a standard one for multigrid algorithms: multiscale relaxation at level l of the scale hierarchy (where level number increases with coarseness) consists of ordinary relaxation at level l, then multiscale relaxation at level $l + 1$, then

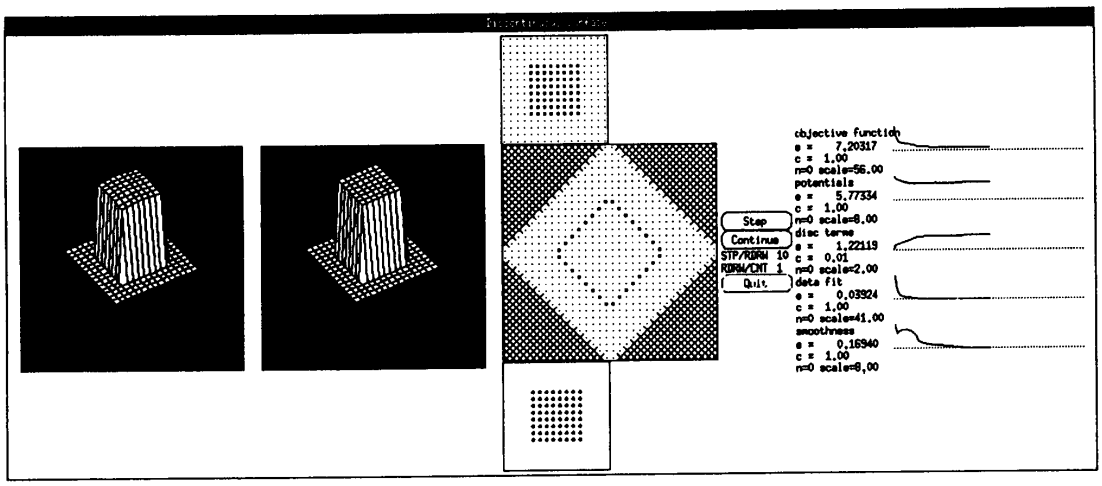


Fig. 6. Aspects of surface interpolation with discontinuities are shown. Leftmost is shown the final form of the network state of the surface interpolation f_{ij} based on the input d_{ij}, which is shown immediately to its right. Proceeding rightward is shown a large central square. The diamonds interior to this square are an encoding of the fine-scale line process neurons l_{ij} corresponding to the final form of the output. The neurons corresponding to the discontinuity in the surface are "on" (have value unity). These neurons form the boundary of the inner diamond. All other line process neurons are "off" (have value zero or nearly zero). Above, the smaller square is an alternate representation of d_{ij}, while the square below is an alternate representation of the fine-scale neurons f_{ij}. The time plots on the extreme right represent energy and energy term evolution. Topmost is the energy itself, shown as monotonically decreasing. The remaining plots correspond to energy summands which display a variety of behaviors but each of which converges to a particular value.

ordinary relaxation at level l, and a final step of multiscale relaxation at level $l + 1$. Of course, multiscale relaxation at level $l + 1$ is defined similarly in terms of ordinary relaxation at level $l + 1$ and multiscale relaxation at level $l + 2$. This recursive control scheme ensures that the smallest and cheapest networks are called upon most frequently in a completely serial algorithm, as discussed, for example, in [10]. When switching levels, information is passed using the prolongation and restriction maps ((5) and (6)) as appropriate. Naturally a parallel algorithm could omit the coarsest levels of the network, those which are too small to make effective use of the parallel machine. (Indeed such an algorithm might profitably relax each finer level of the network in optimal-size chunks, sequentially, with efficient relaxation of interactions that cross chunk boundaries delegated to the next coarser level.)

The graph-matching network of (19) was used to find the best match between two n-node graphs; we tried $n = 8, 16, 25$ for tree graphs and $n = 8, 16, 25, 36$ for two-dimensional graphs. This single-scale algorithm was the control experiment. The problem was also given to a three-level multiscale neural net of the design proposed in this paper; its three levels contained roughly n^2, $n^{4/3}$, and $n^{2/3}$ neurons respectively. The two algorithms resulted in about equally good solutions, for three runs that differed in their randomly selected starting points. But their computational costs were different.

The total number of univariate relaxation steps (line searches) required for convergence was similar for the multiscale and single-scale algorithms. But the multiscale relaxations are much cheaper to perform, on the average, than fine-scale relaxations of the full objective function because most multiscale relaxations occur at the coarsest scales. We estimate the magnitude of this effect by weighting the number of univariate minimization iterations by the number of nonzero connections in the network at each level of the multiscale scheme. This estimate omits the construction cost of the coarse-scale networks and the cost of computing the parameters of $\hat{\phi}(v)$ as well as the effects of highly parallel implementations (which would just truncate the coarsest levels of the network).

The construction cost of the coarse-scale networks and the cost of computing the parameters of $\hat{\phi}(v)$ were previously argued to be small compared with the cost of the fine-scale relaxation which they follow, if many steps of partial relaxation are involved. We tested the multiscale net for the extreme case in which just one univariate minimization was performed during each partial relaxation (more univariate minimizations seemed to be less effective per unit of computational cost in our graph-matching experiments). In this case the network construction cost may become comparable to (but not more than) the relaxation cost if a univariate minimization requires very few search steps. This is because each coarse network construction, and each search step in a univariate minimization procedure, require a number of operations proportional to the number of nonzero connections at any given network level. Also the cost of computing the parameters of $\hat{\phi}(v)$ is proportional to the number of neurons at the same level—generally a much smaller quantity. To avoid implementation-dependent assumptions about the ratio of the relevant proportionality constants, we omit the (usually smaller) cost of the coarse-scale network construction and potential calculation from the following cost estimates except when reporting on actual running times of our serial-computer implementation.

Under this estimate of the cost of the two algorithms, the multiscale algorithm is strongly favored over the single-scale control experiment for matching tree graphs as shown in Fig. 7. Similar results were obtained for matching two-dimensional graphs, as shown in Fig. 8. The simulations generally exhibit punctuated descent toward local minima. Note the roughly constant ratio of convergence times, as a function of n, between

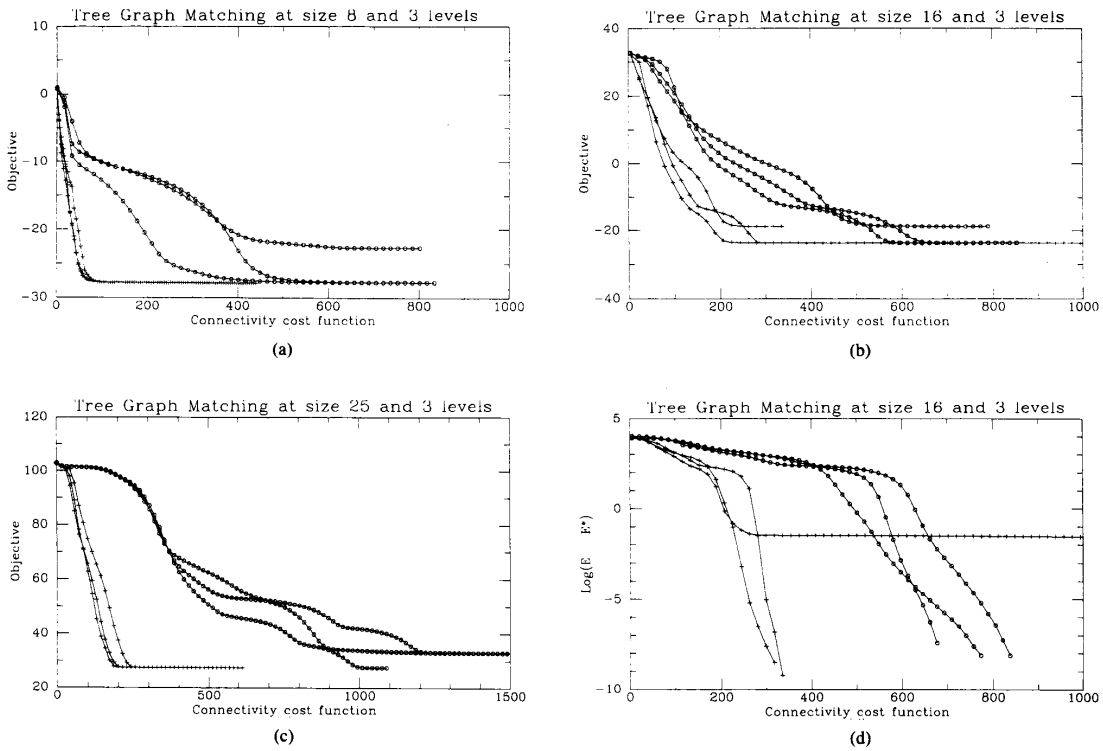


Fig. 7. Graph matching network using tree graphs. Values of E versus estimated cumulative computational cost (total work done so far), as the multiscale and single-scale minimization algorithms proceed. Three runs are shown for each size n. Single-scale runs are plotted with circles and multiscale runs are plotted with '+' signs. Under the multiscale algorithm, E is substantially minimized after roughly 20% to 50% of the effort required in the single-scale control experiment. (a) $n = 8$. Ratio of convergence times ≈ 6. (b) $n = 16$. Ratio of convergence times ≈ 2.5. Multiply by correction factor 0.65 to get observed running-time cost ratio. (c) $n = 25$. Ratio of convergence times ≈ 5. Cost correction factor = 0.51. (d) $n = 16$, same as (b), but we plot $\log_e (E[v] - E[v^*])$, and v^* is the local minimum towards which each trajectory is headed, as estimated by the last configuration of each long run shown. Exponential convergence shows up on this plot as a nearly constant slope, but the time to practical convergence and the quality of local minima are harder to compare than in (b).

single-scale and multiscale simulations. (For larger sizes there may be an additional benefit in cutting off a long tail of convergence at late times.) There is generally about a fivefold improvement in estimated cost under the multiscale method. For three scales and the recursive control scheme, a fivefold improvement is quite reasonable since it corresponds to roughly equal effectiveness of the fine-scale and coarse-scale nets. But our attempts to improve this ratio by moving to four levels resulted in only marginal improvements even for $n = 36$. As can be seen from the figures, the ratios of estimated cost are sensitive to convergence criterion.

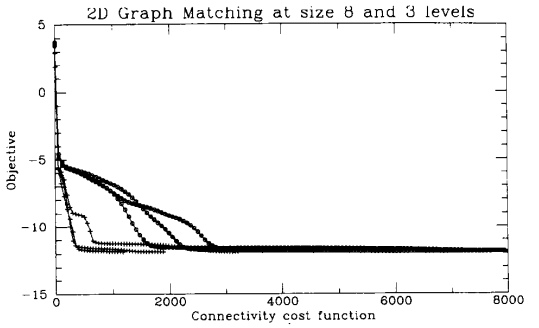
Including the costs of coarse-scale network construction will reduce the observed fivefold savings in an implementation-dependent way. For our serial implementation using sparse data structures and the C programming language on a Sparcstation 1 computer, the ratio of running times for single-scale and multiscale tests was observed to be well predicted by the estimated cost function used in the figures in the following sense: for each problem there is a robust "cost correction factor" which can be used to multiply the estimated cost ratio to get the observed running time ratio, independent of iteration number and, to a lesser extent, problem size. The correction factors are a little less than unity, so the observed cost ratios are a little less favorable to the multiscale method than estimated ones. For two-dimensional graph matching the observed cost correction factor is 0.79 ($n = 16$) or 0.89 ($n = 25$). For tree graph matching the correction factor is 0.65 ($n = 16$) or 0.52 ($n = 25$).

In the case of the two-dimensional discontinuous interpolation network of (21), the multiscale method cut off a long tail of slow convergence, allowing much quicker convergence in the final stages of minimization, as shown in Fig. 9.

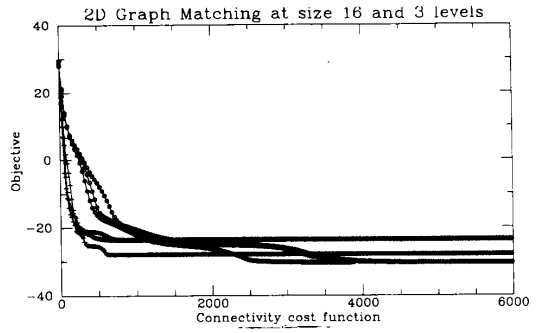
To summarize the experimental results, the multiscale method offers a decrease in computational cost by a factor that depends on the problem and on the convergence criterion, is roughly independent of problem size for the three problems we tested, and is between 2 and 5 for these problems. The advantage may be larger than this when the original network has a slow convergence tail.

V. Conclusion

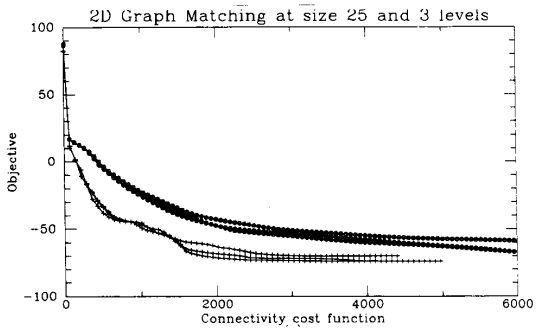
We have developed a fairly general, low-cost multiscale method for neural net optimization. It transforms a neural network into a multiscale neural network of a similar form. In the particular networks to which it was applied, we observed a nontrivial speedup by a constant factor (between 2 and 5) indepen-



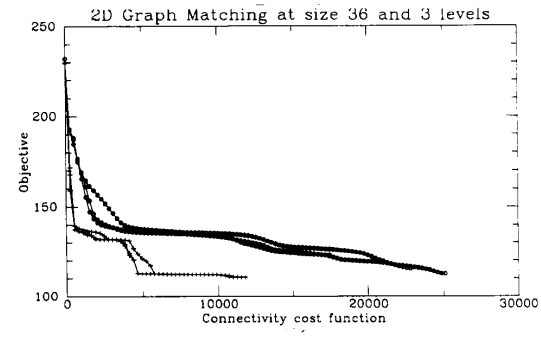
(a)



(b)

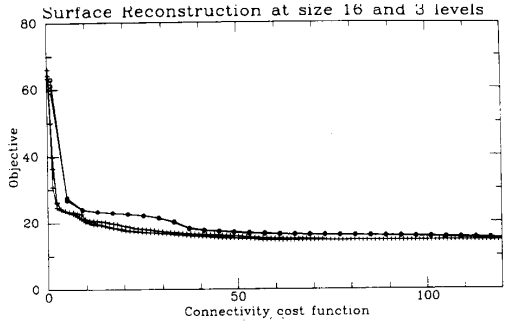


(c)

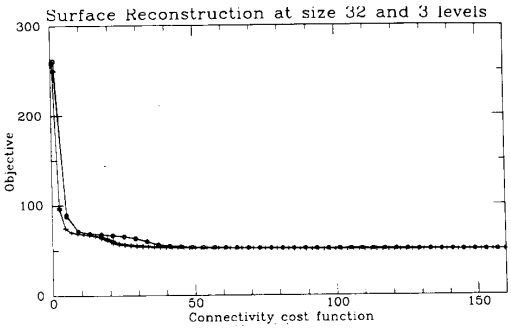


(d)

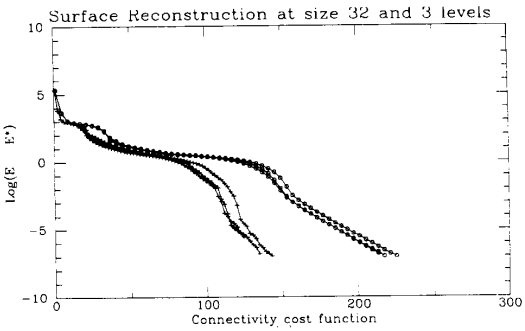
Fig. 8. As in Fig. 7, but here for 2-D graphs. (a) $n = 8$. Ratio of convergence times ≈ 4. (b) $n = 16$. Ratio of convergence times ≈ 5 (worse local minima). Cost correction factor = 0.79. (c) $n = 25$. Ratio of convergence times ≈ 3–10. Cost correction factor = 0.89. Note speedup of late-time convergence tail, which makes the speedup depend sensitively on the convergence criterion. (d) $n = 36$. Ratio of convergence times ≈ 5.



(a)



(b)



(c)

Fig. 9. $E(t)$ for smooth surface reconstruction network with discontinuity detection. Three runs are shown for each size n. Single-scale runs are plotted with circles and multiscale runs are plotted with '+' signs. The main effect of the multiscale algorithms for this network is to cut off long convergence tails, thus greatly reducing the amount of computational effort required for the final reduction in E. (a) $n = 16$. (b) $n = 32$. (c) $n = 32$, same as (b), but plotting $\log_e(E[v] - E[v^*])$ as in Fig. 7(d) to show the multiscale algorithm's effect on late-time convergence tail.

dent of problem size. Further improvements in computational cost are very likely to be available, especially for problem-specific multiscale neural net methods, since the method proposed here is conservatively designed in order to be more generally applicable. The conservative design ensures that the multiscale network never accepts steps that move the original objective function in the wrong direction; thus convergence is unobstructed. The method applies to highly nonlinear networks, without underlying geometric domains or known descent direc-

tions, and for which no particular problem structure is assumed or exploited except for a user-supplied hierarchical partition of the optimization variables.

Appendix I
Analysis of Error Reduction

In this appendix we derive an analytic expression for the reduction in error achieved by a complete relaxation at the coarse scale. The partial coarse-scale relaxations employed in our multiscale optimization algorithm can do no better than this. In principle the quality of the restriction/prolongation process could be graded (and improved) by exploiting this analytic representation of the error, but we leave such questions for future work.

Let $\boldsymbol{v}^*$ denote the limit of the entire optimization process (i.e., $\nabla E[\boldsymbol{v}^*] = 0$). Let $\boldsymbol{v}^0$ denote the state of this process (the value of an iterate) immediately prior to an aggregation step (c.f. (4)), and let $\boldsymbol{v}^1 = P[V]$ (c.f. (5)) denote the state immediately after coarse-scale optimization and prolongation. Then

$$\boldsymbol{\varepsilon}^0 = \boldsymbol{v}^* - \boldsymbol{v}^0 \tag{22}$$

and

$$\boldsymbol{\varepsilon}^1 = \boldsymbol{v}^* - \boldsymbol{v}^1 \tag{23}$$

are the errors in the process before and after a coarse-scale relaxation step. We seek a representation of $\boldsymbol{\varepsilon}^1$ in terms of $\boldsymbol{v}^0$ and $\boldsymbol{\varepsilon}^0$.

The equation for $\boldsymbol{v}^*$ is

$$\nabla_v E[\boldsymbol{v}^*] = 0. \tag{24}$$

By (4) and (5), the correction vector is

$$\boldsymbol{c} \equiv \boldsymbol{v}^1 - \boldsymbol{v}^0 = P \cdot V \tag{25}$$

where the matrix P is given by

$$P = \left(-B_{ia}\nabla_i E[\boldsymbol{v}^0]\right) = \left(B_{ia} z_i[\boldsymbol{v}^0]\right). \tag{26}$$

Now introduce the associated matrix Q:

$$Q = P \cdot \operatorname{diag}(1/w_a) = (P_{ia}/w_a) \tag{27}$$

where

$$w_a = \sqrt{\sum_i B_{ia} z_i^2} \tag{28}$$

which we assume is nonzero. We calculate

$$\begin{aligned}(P^T P)_{ab} &= \Sigma_i B_{ia} z_i z_i B_{ib} \\ &= \Sigma_i z_i^2 \delta_{ab} B_{ia}\end{aligned} \tag{29}$$

(since B is a partition matrix). Continuing,

$$\begin{aligned}(P^T P)_{ab} &= \delta_{ab} w_a^2 \\ &= \left(\operatorname{diag}(w_a^2)\right)_{ab}\end{aligned} \tag{30}$$

whence

$$Q^T Q = \operatorname{diag}(1/w_a) P^T P \operatorname{diag}(1/w_a) = I. \tag{31}$$

The vector $\boldsymbol{c}$ may also be expressed in terms of Q:

$$\boldsymbol{c} = Q \operatorname{diag}(w_a) \cdot V. \tag{32}$$

Now define

$$\Pi = QQ^T. \tag{33}$$

Since $Q^T Q = 1$, $\Pi\Pi = 1$, and Π is a projection operator. Also it preserves $\boldsymbol{c}$:

$$\Pi \boldsymbol{c} = QQ^T Q \operatorname{diag}(w_a) \cdot V = Q \operatorname{diag}(w_a) \cdot V = \boldsymbol{c}. \tag{34}$$

The equation $\Pi \boldsymbol{c} = \boldsymbol{c}$ implies that $(I - \Pi)(\boldsymbol{v}^0 + \boldsymbol{c}) = (I - \Pi)\boldsymbol{v}^0$. Subtracting $(I - \Pi)\boldsymbol{v}^*$ from both sides of this equation yields

$$(I - \Pi)\boldsymbol{\varepsilon}^1 = (I - \Pi)\boldsymbol{\varepsilon}^0 \tag{35}$$

which cannot be solved for $\boldsymbol{\varepsilon}^1$ since $I - \Pi$ is a projection operator.

The coarse-scale relaxation determines V by

$$\begin{aligned}0 &= \nabla_V E[P[V]] \\ &= P^T \nabla_v E[\boldsymbol{v}^0 + \boldsymbol{c}]\end{aligned} \tag{36}$$

(by the chain rule), which can be premultiplied by Q diag $(1/w_a)$ to obtain

$$Q \operatorname{diag}(1/w_a) P^T \nabla_v E[\boldsymbol{v}^0 + \boldsymbol{c}] = 0. \tag{37}$$

That is,

$$\Pi \nabla_v E[\boldsymbol{v}^0 + \boldsymbol{c}] = 0. \tag{38}$$

We also know that $\nabla_v E[\boldsymbol{v}^*] = 0$, so

$$\Pi\left[\nabla_v E[\boldsymbol{v}^0 + \boldsymbol{c}] - \nabla_v E[\boldsymbol{v}^*]\right] = 0. \tag{39}$$

By the mean value theorem there is a $\hat{\boldsymbol{v}}$ in the generalized interval $[\boldsymbol{v}^*, \boldsymbol{v}^0 + \boldsymbol{c}]$ such that

$$\nabla_v E[\boldsymbol{v}^0 + \boldsymbol{c}] - \nabla_v E[\boldsymbol{v}^*] = J[\hat{\boldsymbol{v}}] \cdot (\boldsymbol{v}^0 + \boldsymbol{c} - \boldsymbol{v}^*) \tag{40}$$

where J is the Jacobian of E, generally nonsingular. Thus

$$\Pi J[\hat{\boldsymbol{v}}] \cdot \boldsymbol{\varepsilon}^1 = 0. \tag{41}$$

Adding any scalar multiple (λ) of this to (35), we find

$$\left(I - \Pi + \lambda \Pi J[\hat{\boldsymbol{v}}]\right) \cdot \boldsymbol{\varepsilon}^1 = (I - \Pi) \cdot \boldsymbol{\varepsilon}^0. \tag{42}$$

Consequently,

$$\forall\, \lambda \neq 0,\ \boldsymbol{\varepsilon}^1 = \left(I - \Pi + \lambda \Pi J[\hat{\boldsymbol{v}}]\right)^{-1} (I - \Pi) \cdot \boldsymbol{\varepsilon}^0. \tag{43}$$

This is the desired error expression. In the extreme case where there is one coarse-scale variable for each fine-scale variable (e.g., $B_{ia} = \delta_{ia}$), $\Pi = I$ and $\boldsymbol{\varepsilon}^1 = 0$, as it should be: complete coarse-scale relaxation is also complete fine-scale relaxation in this circumstance.

Equation (43) is an analytic expression for the "error propagator" for our multiscale algorithm. Such error propagator expressions, and approximations of them, are fundamental to most analyses of numerical methods. They often result in the development of improved prolongation and restriction operators: for example if in (43) Π is close to I or if the error ϵ can

be driven close to the null space of $I - \Pi$ by the action of the rest of the propagator during the previous iteration, then rapid convergence will result.

Appendix II
Clocked Objective Function Notation

In this appendix we briefly consider the question of implementing our multiscale neural nets as analog circuits or other special-purpose hardware. A slight change of notation can bring this question into the domain of a circuit-design method that uses rewrite rules by which one objective function can be algebraically transformed into another, more implementable one [2]. We will describe this alternate notation for the multiscale optimization method.

In our case we want to transform a generic neural net objective function $E[\boldsymbol{v}]$, given by (1), into a two-level optimization scheme using $E[\boldsymbol{v}]$ and $\hat{E}[V]$. The problem is that the result of this transformation is not a single objective function, but two objective functions (E and $\hat{E}$) and also the dynamic relationship between them. At some times we optimize E; at other times we optimize $\hat{E}$. We therefore allow the result of transforming an objective function to be a "clocked objective function," whose argument list and functional form depend on time through nonoverlapping clock functions $\psi_\alpha(t) = 0$ or 1 (with $\Sigma_\alpha \psi_\alpha(t) \leq 1$). Such clocked objective functions can be written as

$$E_{\text{clocked}}[x, t] = \sum_\alpha \psi_\alpha(t) E_\alpha[\mathfrak{X}_\alpha^{\text{free}} | \mathfrak{X}_\alpha^{\text{fixed}}] \tag{44}$$

where $\mathfrak{X}_\alpha^{\text{free}}$ and $\mathfrak{X}_\alpha^{\text{fixed}}$ are subsets of variables from the entire set $\{x_i\}$. During phase α (i.e., when $\psi_\alpha(t) = 1$) $E_{\text{clocked}} = E_\alpha[\mathfrak{X}_\alpha^{\text{free}} | \mathfrak{X}_\alpha^{\text{fixed}}]$ is to be minimized with respect to all variables in $\mathfrak{X}_\alpha^{\text{free}}$, while all variables in $\mathfrak{X}_\alpha^{\text{fixed}}$ are to be held fixed or "clamped."

With this notation, the algebraic transformation which encodes our multiscale method can be written as follows (assuming $\hat{T}_{abc} = 0$ for simplicity):

$$E[\boldsymbol{v}] \equiv E[\boldsymbol{v} | T, h] \rightarrow E_{\text{clocked}} \tag{45}$$

where

$$\begin{aligned} E_{\text{clocked}} = {} & \psi_1(t)\, E[\boldsymbol{v}] \\ & + \psi_2(t) \left(\frac{1}{2} \Sigma_a V_a^2 + \frac{1}{2} \Sigma_i (v_i - v_i^0)^2 \right. \\ & \left. + \frac{1}{2} \Sigma_i \left(z_i + \frac{\partial E}{\partial v_i} \right)^2 \right) [z, \boldsymbol{v}^0, V | \boldsymbol{v}] \\ & + \psi_3(t) \left(\Sigma_{ab} \left(\tfrac{1}{2} \hat{T}_{ab}^2 - \hat{T}_{ab} \Sigma_{ij} B_{ia} B_{jb} z_i z_j T_{ij} \right) \right. \\ & \quad + \Sigma_a \left(\tfrac{1}{2} \hat{h}_a^2 - \hat{h}_a \Sigma_i B_{ia} h_i a_i \right. \\ & \quad \left. - \hat{h}_\alpha \Sigma_{ij} B_{ia} z_i v_j^0 (T_{ij} + T_{ji}) \right) \\ & \quad \left. + e_{\text{aggregate}}[\hat{\boldsymbol{a}}_\pm, \hat{\boldsymbol{b}}_\pm, \hat{c}_\pm] \right) [\hat{T}, \hat{\boldsymbol{a}}_\pm, \hat{\boldsymbol{b}}_\pm, \hat{c}_\pm | z] \\ & + \psi_4(t)\, \hat{E}[V | z, \boldsymbol{v}^0, \hat{T}, \hat{h}, \hat{\boldsymbol{a}}_\pm, \hat{\boldsymbol{b}}_\pm, \hat{c}_\pm] \\ & + \psi_5(t)\, \Sigma_i (v_i - v_i^0 + \Sigma_a B_{ia} z_i V_a)^2 [\boldsymbol{v} | z, \boldsymbol{v}^0, V] \end{aligned} \tag{46}$$

Phases 1 through 5 occur in cyclic order. Phase 1 performs fine-scale relaxation; phase 2 is the restriction step, incorporating (6) and the definition of z as the negative gradient; phase 3 creates the coarse-scale net by means of (7); phase 4 is the coarse-scale relaxation step; and phase 5 is the prolongation step corresponding to (5). Phase 3 also involves the analog computation of the parameters of $\hat{\phi}$, summarized by $e_{\text{aggregate}}[\hat{\boldsymbol{a}}_\pm, \hat{\boldsymbol{b}}_\pm, \hat{c}_\pm]$, by means of simple maximum-picking and analog arithmetic networks which may be designed using methods described in [2], for example. Phases 2, 3, and 5 are associated with simple quadratic objectives (except for $e_{\text{aggregate}}$), and we assume that the optimization dynamics can almost completely optimize these objectives in the time allowed by $\psi_\alpha(t)$. By contrast, phases 1 and 4 have nonquadratic objectives which are just partially relaxed during each cycle. The entire sequence could be done recursively for more than two levels of optimization.

Acknowledgment

The authors express their gratitude to C. Douglas and J. Platt for valuable discussions.

References

[1] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proc. Nat. Acad. Sci. U.S.*, vol. 81, pp. 3088–3092, May 1984.
[2] E. Mjolsness and C. Garrett, "Algebraic transformations of objective functions," *Neural Networks*, vol. 3, pp. 651–669, 1990.
[3] W. Hackbusch, "On the multi-grid method applied to difference equations," *Computing*, vol. 20, pp. 291–306, 1978.
[4] W. L. Miranker, *Numerical Methods for Stiff Equations*. Dordrecht: D. Reidel, 1981.
[5] D. Terzopoulos, "Regularization of inverse visual problems involving discontinuities," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-8, p. 419, July 1986.
[6] D. Terzopoulos, "Multiresolution computation of visible-surface representations," PhD thesis, MIT, 1984.
[7] F. Chatelin and W. L. Miranker, "Acceleration by aggregation of successive approximation methods," *Linear Algebra and its Applications*, vol. 43, pp. 17–47, 1982.
[8] J. J. Hopfield, "The effectiveness of analogue 'neural network' hardware," *Network*, vol. 1, pp. 27–40, Jan. 1990.
[9] A. Brandt, "Multi-level adaptive solutions to boundary value problems," *Math. Comp.*, vol. 31, pp. 333–390, 1977.
[10] C. C. Douglas and J. Douglas, Jr., "Abstract multilevel convergence theory requires almost no assumptions," Tech. Rep. RC15853, IBM Yorktown Heights, 1990.
[11] Y. G. Leclerc, "Constructing simple stable descriptions for image partitioning," *Int. J. Computer Vision*, vol. 3, pp. 73–102, 1989.
[12] A. Witkin, D. Terzopoulos, and M. Kass, "Signal matching through scale space," *Int. J. Computer Vision*, vol. 1, pp. 133–144, 1987.
[13] M. R. Garey and D. S. Johnson, *Computers and Intractability*. New York: Freeman, 1979, p. 209, graph partitioning problem in A2.2.
[14] C. Peterson and B. Soderberg, "A new method for mapping optimization problems onto neural networks," *Int. J. Neural Syst.*, vol. 1, no. 3, 1989.
[15] D. E. V. D. Bout, "Graph partitioning using annealed neural networks," *IEEE Trans. Neural Networks*, vol. 1, June 1990.
[16] J. J. Hopfield and D. W. Tank, "Collective computation with continuous variables," in *Disordered Systems and Biological Organization*. New York: Springer-Verlag, 1986, pp. 155–170.
[17] C. von der Malsburg and E. Bienenstock, "Statistical coding and short-term synaptic plasticity: A scheme for knowledge representation in the brain," in *Disordered Systems and Biological Organization*. New York: Springer-Verlag, 1986, pp. 247–252.

[18] J. J. Hopfield and D. W. Tank, "'Neural' computation of decisions in optimization problems," *Biol. Cybern.*, vol. 52, pp. 141–152, 1985.
[19] C. Koch, J. Marroquin, and A. Yuille, "Analog 'neuronal' networks in early vision," *Proc. Nat. Acad. Sci. U.S.*, vol. 83, June 1986.
[20] J. R. Anderson and C. Peterson, "Applicability of mean field theory neural network methods to the graph partitioning problem," Tech. Rep. ACA-ST-064-88, Microelectronics and Computer Technology Corp., February 1988.
[21] D. G. Luenberger, *Linear and Nonlinear Programming*. Reading, MA: Addison-Wesley, 1984, ch. 7.

*

Eric Mjolsness (M'86) received the A.B. degree in physics and mathematics from Washington University in St. Louis in 1980 and the Ph.D. degree in physics and computer science from the California Institute for Technology in 1985.

In 1985 he joined the faculty of the Yale Computer Science Department, where he is currently an Associate Professor. His research interests include neural networks, high-level vision, parallel computation, and biological modeling.

Charles D. Garrett received the B.A. degree in physics and philosophy from Yale University in 1988. At present he is a Computer Programmer for the Yale Computer Science Department.

*

Willard L. Miranker received the B.A. (1952), M.S. (1954), and Ph.D. (1956) degrees in mathematics from New York University.

He has been a research staff member of the IBM T. J. Watson Research Center, Yorktown Heights, NY, since 1958. He is an Adjunct Professor of Computer Science at Yale University. He is the author of numerous publications in applied mathematics and computational mathematics. He has also written books on differential equations and on computer arithmetic. His current research interests are in neural nets, real and artificial, and other nonstandard models of computation.

Dr. Miranker is a member of SIAM and of the American Mathematical Society. He is a Fellow of the AAAS and a recipient of a Humboldt Society award.