Optimization Dynamics for Partitioned Neural Networks

Dimitris I. Tsioutsias
Eric Mjolsness

Research Report YALEU/DCS/RR-970
May 1993

Optimization Dynamics for Partitioned Neural Networks

Dimitris I. Tsioutsias[*,‡] and Eric Mjolsness[†,‡]
Yale University, P.O. Box 2158 Yale Station, New Haven, CT 06520-2158

May 27, 1993

Abstract

Given a relaxation-based neural network and a desired *partition* of the neurons in the network into modules with relatively slow communication between modules, we investigate relaxation dynamics for the resulting partitioned neural network. In particular, we show how the slow inter-module communication channels can be modeled by means of certain transformations of the original objective function which introduce new state variables for the inter-module communication links. We report on a parallel implementation of the resulting relaxation dynamics, for a two-dimensional image segmentation network, using a network of workstations. Experiments demonstrate a functional and efficient parallelization of this neural network algorithm. We also discuss implications for analog hardware implementations of relaxation networks.

[*] *Department of Electrical Engineering,* tsioutsias@cs.yale.edu
[†] *Department of Computer Science,* mjolsness@cs.yale.edu
[‡] This work was supported by AFOSR grant AFOSR-90-0224 and DARPA/ONR grant N00014-92-J-4048.

1 Introduction

Many practical problems in computer vision, pattern recognition, robotics and other areas can be described by relatively simple *objective functions* in a form that permits the design of neural networks which constitute highly parallel optimization algorithms. After the work of Hopfield[7] and others,[8,9] it has become a common practice to derive neural networks as relaxation dynamics for such an objective function. Neural networks of this type show great promise owing to their ability to solve difficult optimization problems with highly parallel relaxation algorithms.

When we consider the implementation of such networks using parallel computers or special-purpose hardware such as VLSI chips, however, a number of new problems arise. Chief among them is that the neural network dynamics constitute a highly parallel algorithm, but only for an implementation medium consisting of abstract neurons with exactly the required connection topology. The network must in practice be *mapped* into some other, readily available hardware without losing much computational efficiency. Furthermore, the desired mapping may change substantially when we need to run the network on a larger parallel computer or a different chip implementation, so it would be valuable to automate the mapping process. Fortunately, we may study (and optimize) such mappings mathematically by making simple models of the actual implementation medium, and asking how abstract neural networks can be mapped efficiently into such models.

For example many parallel computers (including both multicomputers and networks of workstations) can be modeled as sets of "modules" (the separate processors and their local memory), within which a piece of a neural network can be simulated efficiently, connected by relatively slow communication channels that limit the rate at which neurons assigned to different modules can exchange information. A similar model may describe multiple-chip implementations of neural networks, in which affordable inter-chip wires are slower to transmit information than intra-chip wires, and perhaps even multiple-module implementations on a single chip. The entire multi-module picture may of course be generalized to a hierarchy of larger module types with slower interconnection technologies, though we will not discuss this generalization further.

The problem of mapping a relaxation network to an implementation medium described by such a model involves (*a*) assigning each neuron to some module, i.e. *partitioning* the neurons among the modules, and (*b*) introducing special dynamics for neurons connected across modules, or at least for the the inter-module connections themselves, so that the slow speed of communication between modules does not destroy the computational efficiency of the individual modules. Problem (*a*), partitioning the neurons, could perhaps be addressed by considering the sparse graph of network connections and partitioning that graph by means of relaxation networks with deterministic annealing dynamics;[14,21] much further work remains to be done to make such an approach practical. But in this paper we assume that an acceptable partition of neurons into modules has been found, and we address problem

(*b*): how to introduce dynamics for the slow inter-module connections, while still efficiently using the computational capacity and fast connections inside each module.

Our basic solution to this problem will take the form of an algebraic transformation of the objective function, along with suitable dynamics, which preserves the fixed points of the network. This method is developed in section 2.1. Section 2.2 describes the image segmentation network that we use as a test problem. Section 2.3 applies the algebraic transformation method to the image segmentation problem. Section 3.1 describes the methods and algorithms used to test our partitioned dynamics, and section 3.2 exhibits experimental results which confirm the efficient use of computational capacity for a physical computer which satisfies the module model, namely a network of workstations. Good speed-up and efficiency plots are shown. Section 3.2 also describes further experiments designed to probe the probable efficiency of a hypothetical multi-chip implementation in which inter-module wires have a relatively large RC time constant. Section 4 summarizes our results.

2 Theory

A standard form of neural network objective function is[7]

$$(1) \qquad E[\mathbf{v}] \;=\; -\frac{1}{6}\sum_{ijk} T_{ijk} v_i v_j v_k - \frac{1}{2}\sum_{ij} T_{ij} v_i v_j - \sum_i h_i v_i + \sum_i \phi_i(v_i).$$

Here v_i are the neural variables, T_{ij} are pairwise connections between them, T_{ijk} are optional triple connections, h_i are constant bias inputs to the neurons, and $\phi_i(v_i)$ is a potential function for neuron v_i related to its transfer function (or gain function) $g_i(u_i)$ by $\phi_i(v) = \int g_i^{-1}(v)dv$.[7] Many other objective functions may be put in the above form by suitable fixed-point-preserving algebraic tranformations.[12]

We assume a partition of the neural variables v_i into a set of blocks, describable by a sparse rectangular matrix $\{B_{ia}\}$ with 0/1 entries for which $\sum_a B_{ia} = 1$, $\forall i$. Our problem is to introduce neural optimization dynamics for (1) which allow a reduced rate of communication between all pairs of neurons v_i and v_j which happen to be in different blocks or "modules" according to B, but which are connected in the network topology (directly by a nonzero T_{ij}, or by means of some nonzero triple connection T_{ijk}). The method we use is to isolate all summands of (1) which cross module boundaries (these can only be $T_{ijk}v_iv_jv_k$ or $T_{ij}v_iv_j$ terms) and to replace them with new expressions which (*a*) leave the fixed points of E unchanged, and (*b*) introduce new state variables for each cross-module connection or "wire" which can be updated relatively slowly in our neural dynamics by comparison with all other variables. In this way, we model the limited communication capacity of an inter-module connection. Then it becomes purely an experimental question, taken up in section 3, to measure the computational efficiency of the resulting network dynamics.

2.1 Border Neurons

As an example of a fixed-point-preserving transformation, consider the transformation of a summand of E which is a product, XY, of two expressions X and Y. X and Y could be a pair of connected neurons, or more generally they could be any two functions of N variables. In the latter case, such products are generally quite expensive to implement in neural networks and can increase the space and time costs of any algorithm. These considerations motivated[12] the fixed-point-preserving transformations

$$(2)\quad XY = \frac{1}{2}[(X+Y)^2 - X^2 - Y^2] \longrightarrow X(\sigma-\tau) + Y(\sigma-\omega) - \frac{1}{2}\sigma^2 + \frac{1}{2}\tau^2 + \frac{1}{2}\omega^2,$$

and

$$\begin{aligned} XY = \frac{1}{4}[(X+Y)^2 - (X-Y)^2] \quad &\longrightarrow \quad \frac{1}{2}(X+Y)\sigma - \frac{1}{2}(X-Y)\tau - \frac{1}{4}\sigma^2 + \frac{1}{4}\tau^2 \\ (3)\qquad\qquad &= \quad \frac{1}{2}X(\sigma-\tau) + \frac{1}{2}Y(\sigma+\tau) - \frac{1}{4}\sigma^2 + \frac{1}{4}\tau^2, \end{aligned}$$

that introduce new state variables σ, τ, and ω which are linear neurons, of which σ acts to *maximize* the objective rather than minimize it. The dynamics of the network is chosen to seek saddle points.

Our use of these transformations is especially simple. We take X and Y to be individual neurons, or products of two neurons, so that XY is a pairwise or triple interaction term which crosses module boundaries. After the transformation is applied, the newly introduced variables model the state of the inter-module connection which is to be updated slowly by comparison with intra-module connections. For this purpose we arbitrarily chose transformation (3) rather than (2); this choice has the advantage of introducing fewer new neurons, at the cost of introducing two slow neurons per wire rather than just one, since τ and ω of equation (2) exchange information entirely within one module or the other.

After applying tranfromation (3) to all relevant inter-module connections, the Hopfield-Grossberg descent dynamics of the network becomes a descent/ascent dynamics (v_i is the output of the regular neurons in the network, and u_i is the corresponding input)

$$(4)\qquad \dot{u}_i = -r_v\frac{\partial \hat{E}}{\partial v_i} = -r_v[\frac{1}{2}\frac{\partial X}{\partial v_i}(\sigma-\tau) + \frac{1}{2}\frac{\partial Y}{\partial v_i}(\sigma+\tau)],$$

and

$$\begin{aligned} \dot{\omega}_{ij} &= +r_\omega\frac{\partial \hat{E}}{\partial \omega_{ij}} \quad (r_\omega > 0) \\ (5)\qquad &= +r_\omega[\frac{1}{2}X\frac{\partial(\sigma_{ij}-\tau_{ij})}{\partial\omega_{ij}} + \frac{1}{2}Y\frac{\partial(\sigma_{ij}+\tau_{ij})}{\partial\omega_{ij}} - \frac{1}{4}\frac{\partial(\sigma_{ij}^2-\tau_{ij}^2)}{\partial\omega_{ij}}], \end{aligned}$$

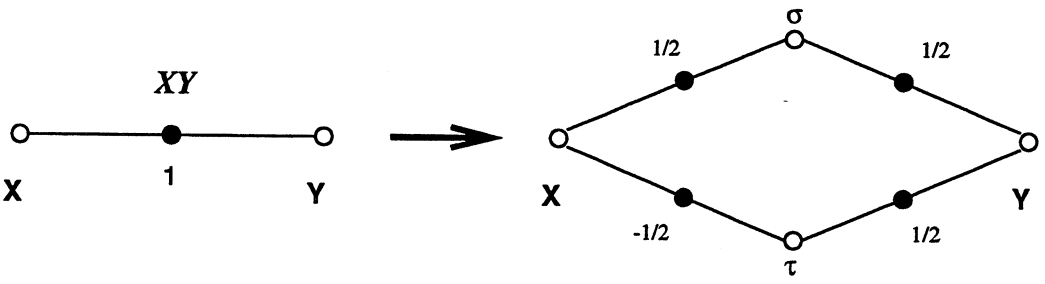


Figure 1: Modeling a slow connection. We replace the XY link with two slow linear neurons that reduce the required communication rate between neurons X and Y.

where ω_{ij} is either σ_{ij} or τ_{ij}, and $\hat{E} = E(\mathbf{v}, \omega)$ is the transformed objective function.

The change in network connectivity achieved by this fixed-point-preserving transformation is shown in Figure 1. Here, σ is a reversed linear neuron,[a] and τ is an ordinary linear neuron. We generally take $r_\sigma = r_\tau$, and a crucial parameter is the relative speed r_σ / r_v of the border neurons σ and τ compared to the original neurons v_i. If we assume that the border neurons are infinitely fast, then transformation (3) does not change the dynamics of the rest of the neurons in the network. In case the linear neurons are not infinitely fast, the basins of attraction of minima might change, but the network's fixed-points remain as before. In the limit in which the border neurons are much *slower* than the original neurons, we have a circuit model incorporating large RC time constants on inter-module wires and hence slow inter-module communications, as desired.

Several further points should be made about the use of transformation (3) or (2) for modeling inter-module connections. It is clear that if v_i is any neuron from block a of the neuron partition, and v_j is any neuron from block $b \neq a$ of the partition, then any connection $T_{ij} v_i v_j$ of E (where $T_{ij} \neq 0$) can be transformed as above at the cost of introducing linear border neurons such as σ_{ij}. But we will also have occasion to consider triple interactions $T_{ijk} v_i v_j v_k$. The simplest case, which holds for the test objective of section 2.2 below and for our computer experiments, obtains when v_k is a member of block a or of block b, that is, when $B_{ka} = 1$ or $B_{kb} = 1$. Then the same transformation rules can be used with e.g. $X = v_i v_k$ and $Y = v_j$. But if v_k belongs to partition $c \notin \{a, b\}$, then these transformations must be applied several times to separate v_i, v_j, v_k into different summands of a final objective function, interacting only with slow border neurons.

The second observation is that a parallel computer implementation becomes considerably easier, and even a VLSI circuit implementation may be easier to design, if the original neurons and the border neurons are updated in alternating phases. Each set of neurons is held

[a] Due to their linear transfer functions with negative gain, in CMOS terminology linear reversed neurons would simply be *inverters*.

fixed during the other set's update phase. Thus, an external clock signal or a synchronization protocol is used to alternate phases of internal relaxation and border relaxation. Using *clocked objective function* notation,[12,13] we have ascent/descent dynamics for an objective function which depends on the update phase:

$$\begin{aligned} E_{clocked}[\mathbf{v},\boldsymbol{\sigma},\boldsymbol{\tau}] &= \hat{E}[\mathbf{v},\bar{\boldsymbol{\sigma}},\bar{\boldsymbol{\tau}}] \oplus \hat{E}[\bar{\mathbf{v}},\boldsymbol{\sigma},\boldsymbol{\tau}] \\ &\equiv \psi_1(t)\hat{E}[\mathbf{v},\bar{\boldsymbol{\sigma}},\bar{\boldsymbol{\tau}}] + \psi_2(t)\hat{E}[\bar{\mathbf{v}},\boldsymbol{\sigma},\boldsymbol{\tau}]. \end{aligned} \tag{6}$$

where $\bar{x}$ is a clamped version of variable x, and $\psi_\alpha(t)$ are non-overlapping clock signals with $\psi_\alpha(t) \in \{0,1\}$.

With this algorithmic simplification, which we use in our computer experiments, the dynamics (5) of the border neurons becomes a simple exponential convergence to the current optimal value by means of a passive RC circuit equation:

$$\begin{aligned} \dot{\sigma}_{ij} &= +r_\sigma \frac{\partial \hat{E}}{\partial \sigma_{ij}} \quad (r_\sigma > 0) \quad \textit{and} \quad \frac{\partial \hat{E}}{\partial \sigma_{ij}} \stackrel{\text{def}}{=} \frac{1}{2}(\sigma^*_{ij} - \sigma_{ij}) \quad \Longrightarrow \\ \sigma_{ij} &= \sigma^*_{ij}(1 - e^{-t/\tau_\sigma}), \end{aligned} \tag{7}$$

where σ^*_{ij} is the current optimal value for the neuron σ_{ij}, being equal to $X \pm Y$ for simulation of the σ_{ij} or τ_{ij} border neurons, respectively. The constant $\tau_\sigma = 2r_\sigma^{-1}$ (in *sec*) corresponds to the RC time constant of the equivalent analog circuit for charging a capacitor, C, through a resistor, R. Our parallel computer implementation uses the analytic form (7) to further decrease the cost of simulating the border neurons.

2.2 Test Problem: 2-D Image Estimation and Segmentation

The problems of image segmentation and estimation have been treated from a *maximum a posteriori* (*MAP*) viewpoint by the use of Gibbs distributions that are defined on the image intensities and their discontinuities. By developing a probabilistic (Bayesian) framework to model dense fields, it can be shown[17] that posterior estimates obtained with Markov Random Fields are equivalent to estimates obtained from regularization theory[20] and similar energy minimization methods. The regularized equations can be derived from a model in which piecewise-smooth images are somehow corrupted by noise (e.g. white Gaussian noise), resulting in a degraded model. The degradation is represented in the energy function by a term measuring closeness to the data, and the piecewise-smooth assumption by terms penalising discontinuities, multiple edges, broken contours, etc.[b] Such smoothness functionals, imposed as weak contraints, may include the membrane and the thin plate models.[19] The regularized equations are then implemented on a discrete mesh using finite element analysis and solved by some iterative relaxation method.

[b] The stabilizers that enforce smoothness on the model are thus interpreted as probabilistic prior models.

The form of one objective function for the image segmentation problem is[16]

$$
\begin{aligned}
(8)\, E(\mathbf{f},\mathbf{v},\mathbf{h}) \;=\; & \alpha \sum_{ij}^{N} \frac{(f(i,j)-g(i,j))^2}{2\sigma_d^2} \\
& + \frac{b}{2}\sum_{ij}^{N} \left[\lambda f_h^2(i,j)(1-v(i,j)) + v(i,j) + \epsilon_1 v(i,j)v(i,j+1) - \epsilon_2 v(i,j)v(i+1,j)\right] \\
& + \frac{b}{2}\sum_{ij}^{N} \left[\lambda f_v^2(i,j)(1-h(i,j)) + h(i,j) + \epsilon_1 h(i,j)h(i+1,j) - \epsilon_2 h(i,j)h(i,j+1)\right] \\
& + \frac{1}{\beta}\sum_{ij}^{N} \Big[\; u_v(i,j)v(i,j) - \log[1+\exp(u_v(i,j))] \;+ \\
& \qquad + \; u_h(i,j)h(i,j) - \log[1+\exp(u_h(i,j))] \;\Big] .
\end{aligned}
$$

Here, $\mathbf{f}$ is the piecewise-smooth surface to be fit to the image data; $\mathbf{g}$ is the fixed set of image intensities to be used as data points; and $\mathbf{v}$ and $\mathbf{h}$ are the discontinuity fields to be used to model the region boundaries in the image, also known as *line processes.* The latter are binary variables that serve to indicate a discontinuity in the intensity field, i.e., a place where the gradient of the intensity has a non-zero value. α and b are constants carefully tuned for different images or sets of images, λ is a smoothing parameter, ϵ_1 and ϵ_2 control edge linking and thinning respectively, $\mathbf{u}$ is a field related to $\mathbf{v}$ and $\mathbf{h}$, and β is the mean field annealing parameter that acts as the inverse temperature and guides the continuation procedure.

The process $\mathbf{u}$ is updated as[16]

$$
(9)\quad u_v(i,j) \;=\; \frac{\beta b}{2}\left[\lambda f_h^2(i,j) - 1 - \epsilon_1(v(i,j+1)+v(i,j-1)) + \epsilon_2(v(i+1,j)+v(i-1,j))\right] ,
$$

$$
(10)\quad u_h(i,j) \;=\; \frac{\beta b}{2}\left[\lambda f_v^2(i,j) - 1 - \epsilon_1(h(i+1,j)+h(i-1,j)) + \epsilon_2(h(i,j+1)+h(i,j-1))\right] ,
$$

and the horizontal and vertical discontinuity fields as

$$
(11)\qquad h(i,j) = \frac{1}{1+\exp(-u_h(i,j))}, \qquad v(i,j) = \frac{1}{1+\exp(-u_v(i,j))}.
$$

In the expressions above, f_h and f_v correspond to the horizontal and vertical intensity gradients, i.e., $f_h \stackrel{\text{def}}{=} (f(i,j+1) - f(i,j))$ and $f_v \stackrel{\text{def}}{=} (f(i+1,j) - f(i,j))$. (9), (10) and (11) update $\mathbf{u}, \mathbf{v}$ and $\mathbf{h}$ given $\mathbf{f}$, so $\mathbf{f}$ is the only remaining quantity to be optimized.

The energy function (8) consists of disparate 'forces' that have to be balanced at a fixed point: the mean squared difference between $\mathbf{f}$ and $\mathbf{g}$ (the α-term in (8)), and the calculation of the number and nature of the line processes quantified by the $\mathbf{v}$ and $\mathbf{h}$ fields (the b-terms). The last summand in (8) is imposed by the mean field annealing (MFA) equations; the weak

membrane stabilizer that was formulated with digital line processes $s \in \{0,1\}$, was changed with a monotonic transformation to use analog discontinuities $s \in [0,1]$. The $(1/\beta)$-term can be thought of as a barrier function that restricts the values of the line processes to be in the interval $(0,1)$ by raising infinite barriers at 0 and 1. A special point has to be made here: the function we implement is not quite a region segmentation function, at least by its classical meaning,[10] that is, having piecewise-flat, closed regions in the original intensity image. To make it such, we should set the λ-parameter in (8) to high values.[16]

Such energy functions are generally non-convex, and deterministic minimization methods easily get stuck in local minima. Stochastic methods, on the other hand, are burdened by their computational requirements. A method that enables energy minimization and which combines the uphill movements of a stochastic algorithm with the speed of a deterministic approach is mean field annealing (*MFA*), a deterministic continuation technique. *MFA*, like simulated annealing, has its origins in statistical mechanics but has some additional advantages: (*i*) the *MFA* equations are isomorphic to *RC*-equations for a circuit of analog amplifiers, and (*ii*) the *MFA* equations of motion are identical to those obtained when mapping optimization problems to neural networks. Furthermore, MFA as a feedback algorithm is inherently parallel and can be implemented on massively-parallel localy-connected computer architectures. For our purposes, *MFA* amounts to repeated deterministic optimization with slowly decreasing temperature. MFA has been used for many other applications.[7,14,15,22]

2.3 Transformation of Image Segmentation Objective

Objective Function: For the energy function (8), the only inter-module communication needed is due to the horizontal and vertical intensity gradients f_h, f_v, which appear in the objective as squared quantities leading to

$$
\begin{aligned}
\frac{1}{2}f_h^2 &= \frac{1}{2}(f_{i,j+1} - f_{i,j})^2 \\
&= \sum_{i,j}^{N} f_{i,j}^2 - \underbrace{f_{i,j}f_{i,j+1}} \\
&= \underbrace{\sum_{i,j}^{N} f_{i,j}^2}_{\text{(quadratic potential)}} - \underbrace{\{[\frac{1}{2}f_{i,j}(\sigma_{H,ij} - \tau_{H,ij}) + \frac{1}{2}f_{i,j+1}(\sigma_{H,ij} + \tau_{H,ij})]}_{\text{(interaction terms)}} - \underbrace{[\frac{1}{4}\sigma_{H,ij}^2 - \frac{1}{4}\tau_{H,ij}^2]}_{\text{(neuron potentials)}}\}.
\end{aligned} \tag{12}
$$

Here, σ_H, τ_H are the neurons introduced for the horizontal discontinuity field. Similarly, for the vertical intensity gradient

$$
\begin{aligned}
\frac{1}{2}f_v^2 &= \frac{1}{2}(f_{i+1,j} - f_{i,j})^2 \\
&= \frac{1}{2}f_{i+1,j}^2 + \frac{1}{2}f_{i,j}^2 - [\frac{1}{2}f_{i,j}(\sigma_{V,ij} - \tau_{V,ij}) + \frac{1}{2}f_{i+1,j}(\sigma_{V,ij} + \tau_{V,ij}) - \frac{1}{4}\sigma_{V,ij}^2 + \frac{1}{4}\tau_{V,ij}^2].
\end{aligned} \tag{13}
$$

Gradients of $\hat{E}$: The gradient vectors are computed as usual, except for the terms involving the new linear neurons which become (c.f. (13), (14))

$$\begin{aligned} \frac{\partial}{\partial f_{i,j}}[\frac{1}{2}(f_{i,j+1}-f_{i,j})^2] &\longrightarrow f_{i,j}-\frac{1}{2}(\sigma_{H,ij}-\tau_{H,ij}) \\ \frac{\partial}{\partial f_{i,j+1}}[\frac{1}{2}(f_{i,j+1}-f_{i,j})^2] &\longrightarrow f_{i,j+1}-\frac{1}{2}(\sigma_{H,ij}+\tau_{H,ij}) \end{aligned} \tag{14}$$

and similarly for the vertical gradients

$$\begin{aligned} \frac{\partial}{\partial f_{i,j}}[\frac{1}{2}(f_{i+1,j}-f_{i,j})^2] &\longrightarrow f_{i,j}-\frac{1}{2}(\sigma_{V,ij}-\tau_{V,ij}) \\ \frac{\partial}{\partial f_{i+1,j}}[\frac{1}{2}(f_{i+1,j}-f_{i,j})^2] &\longrightarrow f_{i+1,j}-\frac{1}{2}(\sigma_{V,ij}+\tau_{V,ij}) \end{aligned} \tag{15}$$

Linear Neuron Dynamics: The ascent/descent dynamics given by (4), (5) are applied to the objective function (8). The only contribution comes from the intensity gradients f_h and f_v, giving for the border neuron $\sigma_{(H/V,ij)}$ (which is a *reversed linear* neuron)

$$\begin{aligned} \dot{\sigma}_{(H/V,ij)} &= +r_\sigma \frac{\partial \hat{E}}{\partial \sigma_{(H/V,ij)}} = +r_\sigma \frac{\partial}{\partial \sigma_{(H/V,ij)}}[\frac{1}{2}f_{h/v}^2] \\ &= \frac{+r_\sigma}{2}[f_{(i,j)}+f_{(i,j+1/i+1,j)}-\sigma_{(H/V,ij)}] \quad (r_\sigma>0). \end{aligned} \tag{16}$$

and for the $\tau_{(H/V,ij)}$ (which is a *regular linear* neuron)

$$\begin{aligned} \dot{\tau}_{(H/V,ij)} &= +r_\tau \frac{\partial \hat{E}}{\partial \tau_{(H/V,ij)}} \\ &= \frac{+r_\tau}{2}[f_{h/v}+\tau_{(H/V,ij)}] \quad (r_\tau>0). \end{aligned} \tag{17}$$

3 Experiments

3.1 Methods

To test our methods experimentally, a network of IBM RS/6000 workstations connected by an ethernet was used. The cluster was comprised of six 560 and ten 340 models. The code was implemented in C++ and C-Linda. Linda adds shared-memory operations to many languages and computing platforms.[4]

The energy function $E(\mathbf{f},\mathbf{v},\mathbf{h})$ in (8) is non-convex in its parameters $\mathbf{f},\mathbf{v}$ and $\mathbf{h}$. To optimize such functions a continuation approach similar to the Graduated Non Convexity (GNC) was

used.[3] The function's optimum is tracked as the minimum of a sequence of functions that asymptotically converge to the original objective when the continuation control parameter, β, approaches its limit values. At every stage, the current optimal point is used as a starting position for the next iteration of the algorithm. Direct methods such as Gaussian elimination or triangular decomposition are usually computationally restrictive; relaxation techniques on the other hand are massively parallel. The Conjugate Gradient (CG) relaxation method[5] with the Polak-Ribière step-size γ_k^{PR} was chosen to run on the intensities.[c] The objective function (8) is quadratic in the intensities $\mathbf{f}$, but non-quadratic in the line processes. Thus, the Iterated Conditinal Modes (ICM) algorithm[2] was run on the latter. ICM is a coordinate-wise descent method that minimizes (8) with respect to h_{ij} or v_{ij} while keeping all other line processes fixed, until a full sweep of the lattice is performed.

This iterated scheme is[16]

1. Set $\beta = \beta_0$.
2. Run the CG on the intensities.
3. Update the line processes by ICM after each CG step.
4. Return to step 2 till convergence.
5. Increase the control parameter, e.g. $\beta = 2^k\beta_0,\ k = 0, 1, 2, ...$
6. Return to step 2 until the limit value of $\beta = \beta^*$ has been reached.

A parallel implementation was obtained by partitioning the problem variables into rectangular blocks. Syncronization was achieved as in many Linda programs by using the master-worker paradigm; the supervising process is responsible for scheduling slave processes and for their synchronization. Each worker simultanously performs the optimization algorithm just described on a different part of the original network, and when it is ready to send and receive data, signals the master; when such information is available from all other workers, the master enables the workers to send and receive. This scheme helps emulate the *clocked objective* (6). Figure 2 shows the clocked network implementation: processors $P1 - P4$ are assigned parts of the relaxation net and are allowed to optimize in parallel. Then the master gathers the neuron values from the module boundaries and updates the linear neurons in the border stripes $M1, M2$. It then transmits these values back to $P1 - P4$ and a new cycle of the optmization procedure begins.

The performance of this method was quantified both by the total wall-clock time, T_{tot}, that the network took to converge to its fixed-points, and by the absolute error in final energy, ΔE, between the parallel and the corresponding serial runs. Four algorithmic parameters emerged as critical: the linear neuron update parameter, r_σ, the number of partial relaxation steps before communicating data, s, the network size, N, and the number of procesors, p.

---

[c] $\gamma_k^{PR} = \frac{<g_k, g_k - g_{k-1}>}{||g_{k-1}||^2}$, where g_k, g_{k-1} are the current and previous gradients of the objective.

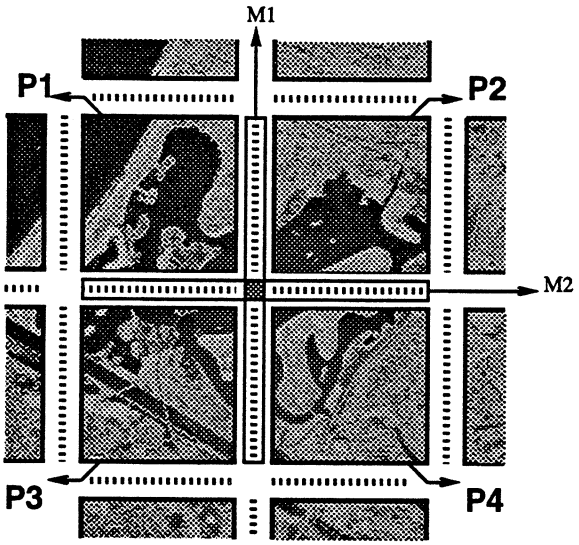


Figure 2: Part of the relaxation net. By an orthogonal partitioning into blocks, each processor P_i has to communicate at most with 4 others to send and receive neuron values.

3.2 Results

3.2.1 Optimal parameter selection.

Several images of increasing size were tried, namely 64×64, 128×128, 256×256 and 512×512 pixels. The net was partitioned from 1 up to 10 blocks, and both the ten-workstation ($340's$) and the six-workstation ($560's$) networks were used. Due to differences in peak performances between the two types of workstations, CPUs of the same type were used to avoid any machine-dependent imbalances in performance. Thus, two networks were set up, each comprised of workstations of the same type, and identical simulations were tried on both; the results gathered were similar and, hence, only those from the ten-workstation network are presented in the sequel.

Various different values for parameters s and r_σ were tried, and the performance criteria T_{tot} and ΔE were measured. The results are plotted in Figure 3. The simulations demonstrate that there exist optimal values for s and r_σ, and a careful choice results in reduced convergence times and lower errors in energy. The measured T_{tot} as a function of s and r_σ is roughly a valley open at one end. Upper and lower bounds for s exist: T_{tot} is high both for very frequent value communication (e.g. $s \leq 5$) and for very rare communication (e.g. $s \geq 100$); moreover, $\Delta E(s, r_\sigma)$ is high for rare communication (e.g. $s \geq 100$), whereas trying to keep ΔE low with small s (e.g. $s \leq 5$) results in a relatively high T_{tot}. [Note that $s \geq 5$ indicates that the high cost of communication cannot simply be ignored by updating border neurons as frequently as intra-module neurons, e.g. with $s = 1$.] Figure 3 also shows that the acceptable range of r_σ is roughly $r_\sigma \geq 0.1$, and has no upper bound.

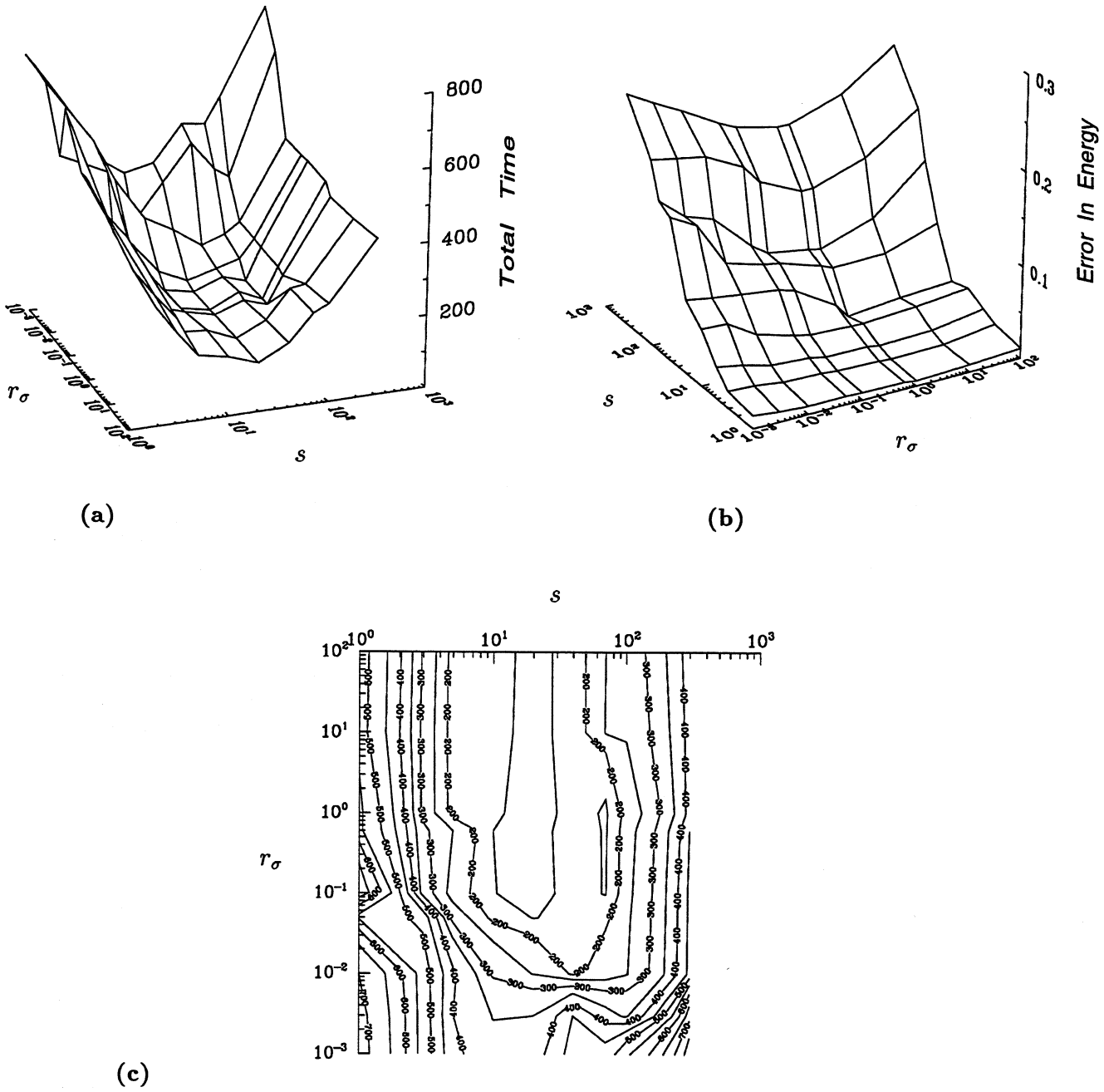


Figure 3: (a) Total convergence time as a function of s, r_σ for the 64×64 image. (b) Error in final energy as a function of (s, r_σ) for the same image. (c) shows a contour plot of surface-(a); the axes for s and r_σ are drawn in a *log*-scale to incorporate the wide range of values tested. The network was partitioned in 4 blocks, each of which was assigned to a different CPU.

A specific range of r_σ values resulted in the best simulation results (Figure 3). Good performance was found for $r_\sigma \in [0.6, 10.0]$ and $s \in [10, 40]$. Optimal results were obtained for $r_\sigma \in [0.8, 5.0]$ and $s \in [10, 20]$. An important point is that these parameter regions remained roughly unchanged for the different image sizes. In all cases the convergence-time penalty for an excessively small value of r_σ (too slow a communication channel) was large, whereas the convergence time penalty for a very large r_σ was mild, at least if the number of steps s was also adjusted correctly.

3.2.2 Tuning r_σ and analog circuits.

In section 2.1 it was argued that large RC time constants on inter-module wires can represent slow inter-module communication in a distributed or parallel computer which implements the network. The observation is stronger for an actual analog circuit implementation, in which the RC time constants for wires between modeules may literally be relatively large owing to the power dissipation cost of a relatively long wire (higher for smaller RC), and to other engineering factors that prohibit $r_\sigma \to \infty$ such as finite source impedances, signal termination at the output, reflections, etc. In our simulations, linear-neurons dynamics (7) with only partial convergence (to 80% of each neuron's current optimal value) were adequate in practice.[d] Furthermore, the large values of s mean that even $r_\sigma = 1.0$ in our units corresponds to a relatively low inter-module communication rate, since s intra-module CG steps were performed for every partial RC relaxation step.

In an analog circuit implementation additional costs would enter to discourage the use of very large r_σ: the fixed hardware cost and recurring power costs of long, fast, dissipative wires could dominate. So it is interesting to observe that very large values of r_σ were not required (nor even advantageous) to obtain good convergence times in Figure 3, and that intermediate values of r_σ were used to obtain the speed-up and efficiency results below.

3.2.2 Efficient parallelization.

In general, the efficiency of a parallel system decreases as the number of processors, p, increases, provided the problem size N is held fixed, since the communication overhead $T_o(N, p)$ increases with p.[e] We now report our observations of parallel efficiency. A binary partitioning was chosen for the domain decomposition: the domain was partitioned into two parts recursively so that the number of neurons in each part was identical. This technique ensures a good load balance: it optimizes the computational load and allows a trade-off between locality of computation and maximization of the ratio of computation to communication (or equivalently, that area to perimeter ratio should be as large as possible).[1]

[d] The inter-module wires emulated by border neurons are equivalent to low-pass filters when brought to the frequency domain, for which a *breakpoint* occurs at $f_{3dB} = 1/2\pi RC$.

[e] Speedup S is defined as the ratio of the time needed by the best serial algorithm to converge, to the time needed by a parallel algorithm ($S = T_s/T_p$). (Numerical) efficiency, E, is defined as the ratio of S to the number of processors ($E = S/p$).

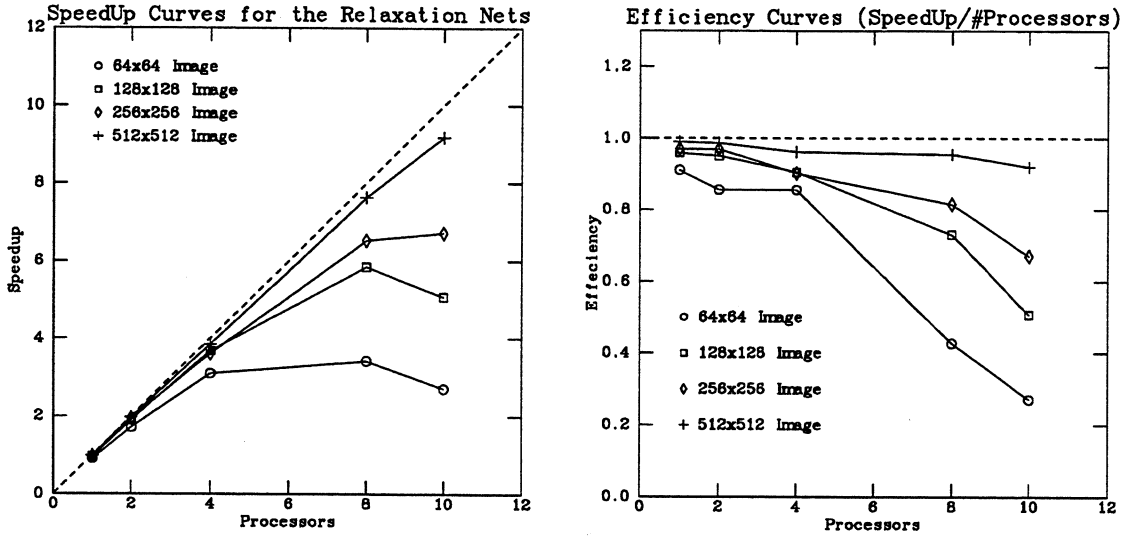


Figure 4: Speed-up and Efficiency curves for various image sizes, N. All simulations were performed with $r_\sigma = 1.0$ and $s = 20$. Notice the improvement in efficiency with increased values of N, and the good absolute efficiency for image sizes generally required in practice.

Our measured speed-up and efficiency curves for various image sizes and numbers of processors are presented in Figure 4. For each run the same values for s and r_σ were used, namely $s = 20$ and $r_\sigma = 1.0$. It is clear that for increased number of processors and larger image sizes, a substantial speed-up is obtained that is close to the optimal linear speed-up. Furthermore, the measured efficiency is above 0.9 for the largest image and above 0.5 for all but the smallest image, at all processor numbers.

These results are also somewhat burdened by a relatively slow ethernet communication and other NFS network delays. Future simulations with a high-speed fiber-optic network could improve the parallel convergence times for the smaller image sizes. But the larger image sizes are the ones relevant for most practical applications of computer vision.

Figure 5 presents a segmentation ($\mathbf{f}$ field) and the line processes (fields $\mathbf{v}$, $\mathbf{h}$) for the 128×128 image and Figures 6, 7 the original and a segmented instance for the 512×512 image, respectively.

4 Discussion and Conclusion

We have presented a general method for adapting an optimization-based neural network to an implementation medium consisting of a set of modules with relatively high cost or low capacity for each inter-module communication link. Examples of such implementation media include multiple special-purpose VLSI chips, large parallel computers, and the

workstation networks used in our experiments. The method consists of applying algebraic transformations which preserve the fixed-points of the objective function and which introduce new neurons to model the state of the inter-module connections.

We applied the method in a favorable case, that of a nonconvex two-dimensional image segmentation problem and its neural network, and observed good parallel speed-up and computational efficiency on a network of workstations. The best parameter settings allowed slow inter-module communication and did not change significantly with image size; they are expected to be compatible with implementation by modular analog circuits as well as by parallel or distributed computers.

Aknowledgements

We wish to acknowledge Anand Rangarajan for interesting discussions and for providing us with the San Diego Mission Bay aerial photo.

References

1. G.Y. Ananth, and V.Kumar, "Scalability Analysis of Partitioning Strategies for Finite Element Graphs", *Dept. of Comp.Sc., Univ.of Minnesota, Technical Report-Preprint* (April 1992).

2. J. Besag, "On the statistical analysis of dirty pictures", *Journal of the Royal Statistical Society, B*, **48(3)**, pp.259-302, (1986).

3. A. Blake, and A. Zisserman, *Visual Reconstruction*, MIT Press, (1987).

4. N. Carriero, and D. Gelernter, "How to write Parallel Programs: A Guide to the Perplexed", *ACM Computing Surveys*, **21(3)**, (Sep. 1989).

5. R. Fletcher, *Practical Methods of Optimization*, John-Wiley & Sons, (1987).

6. D. Geiger, and A. Yuille, "A Common Framework for Image Segmentation", *International Journal of Computer Vision*, **6(3)**, pp.227-243, (1991).

7. J. Hopfield, "Neurons with graded response have collective computational properties like those of a two-state neuron", *Proc. of the National Academy of Sciences USA*, **81**, pp.3088-3092, (1984).

8. J. Hopfield, and D.W. Tank, "Neural computation of decisions in optimization problems", *Biol. Cybernet.*, **52**, pp.141-152, (1985).

9. C. Koch, J. Marroquin, and A. Yuille, "Analog 'neuronal' networks in early vision", *Proc. of the National Academy of Sciences USA*, **83**, pp.4263-4267, (1986).

10. Y.G. Leclerc, "Constructing Simple Stable Descriptions for Image Partitioning", *International Journal of Computer Vision*, **3(1)**, pp.73-102, (1989).

11. C. Mead, *Analog VLSI and Neural Systems*, Addison-Wesley, (1989).

12. E. Mjolsness, and C. Garrett, "Algebraic Transformations of Objective Functions", *Neural Networks*, **3**, pp.651-669, (1990).

13. E. Mjolsness, and W.L. Miranker, "Greedy Lagrangians for Neural Networks: Three Levels of Optimization in Relaxation Dynamics", *Research Report YALEU/DCS/TR-945, Version 1*, (1993).

14. C. Peterson, and B. Söderberg, "A New Method for Mapping Optimization Problems onto Neural Networks", *International Journal of Neural Systems*, **1(1)**, (1989).

15. C. Peterson, "Mean Field Theory Neural Networks for Feature Recognition, Content Addrssable Memory and Optimization", *Connection Science*, **3(1)**, (1991).

16. A. Rangarajan, and R. Chellappa, "A Continuation Method for Image Estimation and Segmentation", *Center for Automation Research, Un. of Maryland, College Park, CS-TR-276 9*, (October 1991).

17. M.A. Sivilotti, M.A. Mahowald, and C.A. Mead, "Real-time visual computation using analog CMOS processing arrays", in *Advanced Research in VLSI: Proc. of the 1987 Stanford Conference*, Cambridge, MA: MIT Press.

18. R. Szeliski, Bayesian Modeling of Uncertainty in Low-Level Vision, *Kluwer Academic Publishers*, (1989).

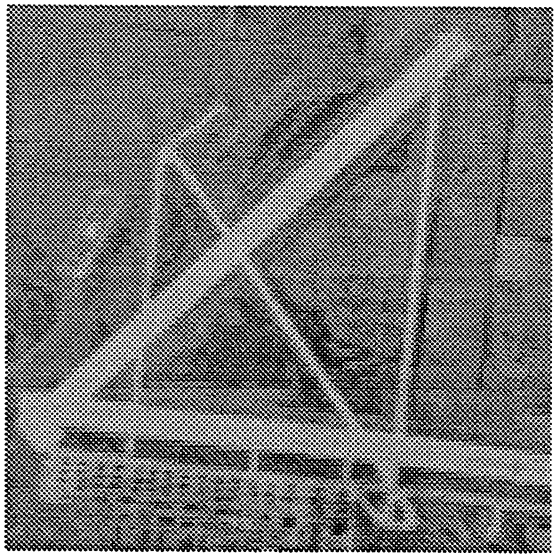
19. D. Terzopoulos, "Regularization of inverse visual problems involving discontinuities", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **8(4)**, pp.413-424, (1986).

20. A.N. Tikhonov, and V.Y. Arsenin, *Solutions of Ill-Posed Problems*, V.H.Winston, Washington D.C., (1977).

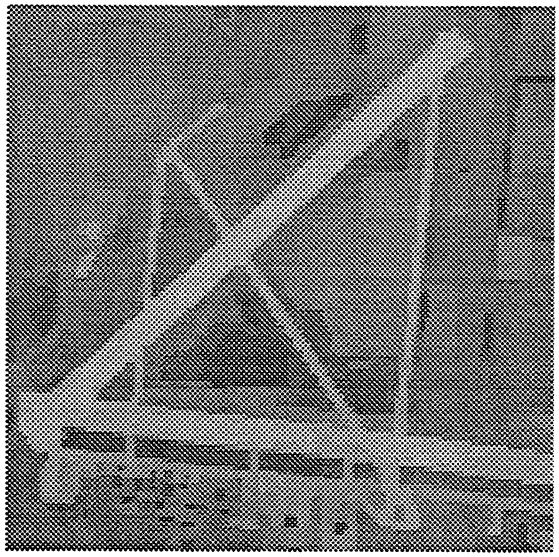
21. D.E. Van den Bout, and T.K. Miller,III, "Graph Partitioning Using Annealed Networks", *IEEE Trans. on Neural Networks*, **1(2)**, (1990).

22. A. Yuille, "Energy Functions for Early Vision and Analog Networks", *Biol. Cybernet.*, **61**, pp.115-123, (1989).

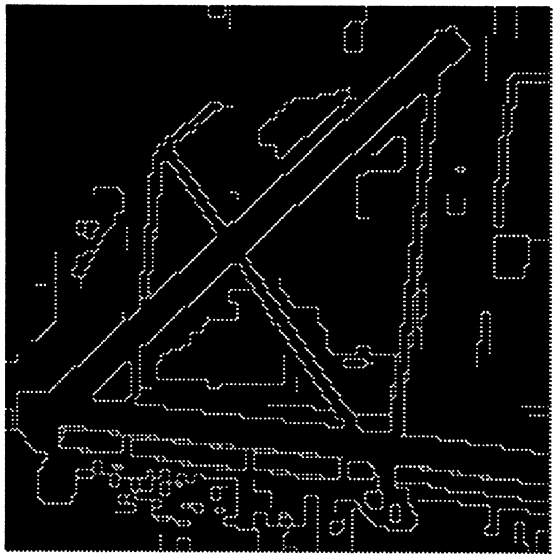
23. A. Yuille, "Generalized Deformable Models, Statistical Physics, and Matching Problems", *Neural Computation*, **2**, pp.1-24, (1990).



(a)



(b)



(c)

Figure 5: (a) Original 128×128 airport aerial photo, (b) Segmentated image, (c) Line processes of the segmented image. 4 processors were used, with $\lambda = 0.022$ and $\sigma_d = 11.2$.

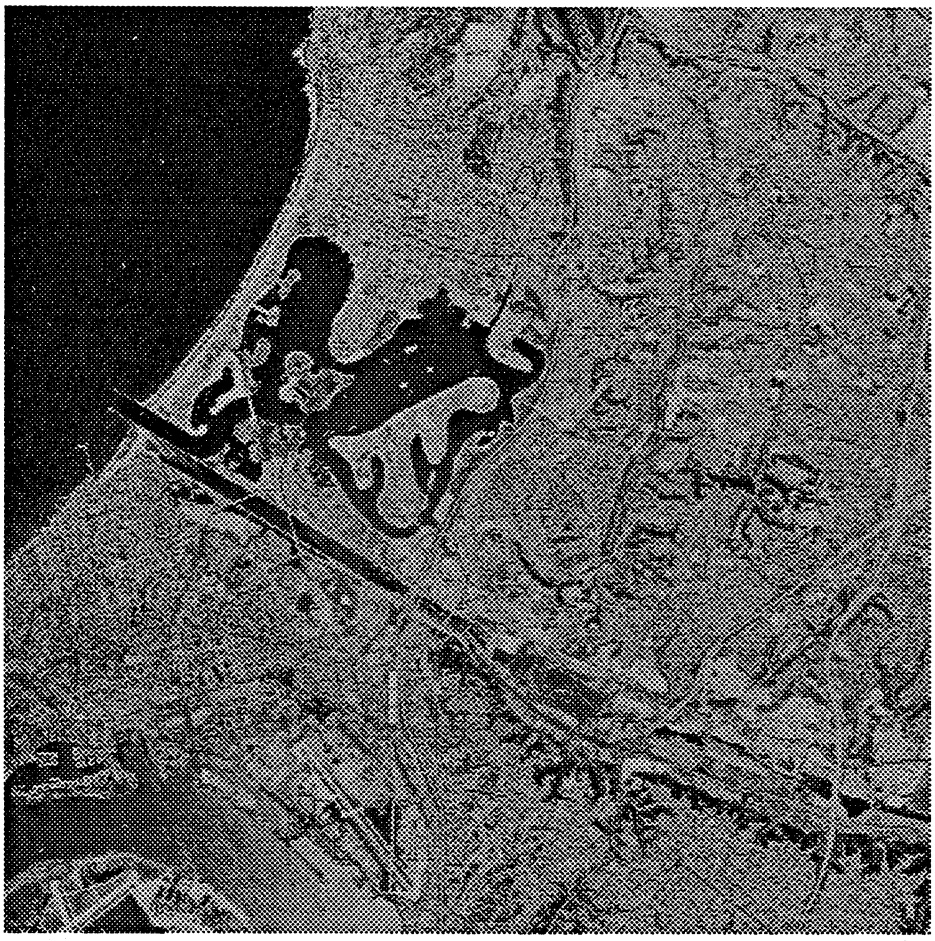


Figure 6: Original 512×512 San Diego Mission Bay aerial photo. Notice the great amount of information contained.

Figure 7: Segmented image. 8 processors were used, with $\lambda = 0.024$ and $\sigma_d = 19.5$.