

Algebraic Transformations of Objective Functions

Eric Mjolsness and Charles Garrett
Department of Computer Science
Yale University

November 23, 1994

Abstract

Many neural networks can be derived as optimization dynamics for suitable objective functions. We show that such networks can be designed by repeated *transformations* of one objective into another with the same fixpoints. We exhibit a collection of algebraic transformations which reduce network cost and increase the set of objective functions that are neurally implementable. The transformations include simplification of products of expressions, functions of one or two expressions, and sparse matrix products (all of which may be interpreted as Legendre transformations); also the minimum and maximum of a set of expressions. These transformations introduce new interneurons which force the network to seek a saddle point rather than a minimum. Other transformations allow control of the network dynamics, by reconciling the Lagrangian formalism with the need for fixpoints. We apply the transformations to simplify a number of structured neural networks, beginning with the standard reduction of the winner-take-all network from $\mathcal{O}(N^2)$ connections to $\mathcal{O}(N)$. Also susceptible are inexact graph-matching, random dot matching, convolutions and coordinate transformations, and sorting. Simulations show that fixpoint-preserving transformations may be applied repeatedly and elaborately, and the example networks still robustly converge.

Keywords: Objective function, Structured neural network, Analog circuit, Transformation of objective, Fixpoint-preserving transformation, Lagrangian dynamics, Graph-matching neural net, Winner-take-all neural net.

1 Introduction

Objective functions have become important in the study of artificial neural networks, for their ability to concisely describe a network and the dynamics of its neurons and connections. For neurons with objective-function dynamics, the now standard procedure (Hopfield, 1984; Hopfield and Tank, 1985; Koch et al., 1986) is to formulate an objective function (called the “objective” in what follows) which expresses the goal of the desired computation, then to derive a local update rule (often a simple application of steepest descent) which will optimize the dynamic variables, in this case the artificial analog neurons. The update rule should ultimately converge to a fixpoint which minimizes the objective and should be interpretable as the dynamics of a circuit.

This procedure is direct but has drawbacks. For example it considers only the goal of the computation and not the cost for attaining the goal or the path taken in doing so. The resulting neural nets can be quite expensive in their number of connections, and for some objectives the associated local update rule has an algebraic form unsuitable for direct implementation as a neural network.

In this paper we show how to modify the standard procedure by interpolating an extra step: after the objective is formulated, it can be algebraically manipulated in a way which preserves its meaning but improves the resulting circuits, e.g. by decreasing some measure of their cost. Then the improved circuit is derived from the modified objective. Often it will be clear from the algebra alone that some savings in number of connections will occur, or that a previously non-implementable objective has been transformed to a form (e.g. single-neuron potentials plus polynomial interactions (Hopfield, 1984)) whose minimization may be directly implemented as an analog circuit in a given technology. And one interesting class of transformations can establish detailed control over the state-space trajectory followed during optimization.

We do not require that the algebraic manipulation be done automatically; we just ask whether and how it can be done. Our answer is in the form of a short, non-exhaustive list of very general transformations which can be performed on summands of an objective, provided they are of the requisite algebraic form, without altering the fixpoints of the resulting network. Many of these transformations introduce a relatively small number of new neurons which change the local minima of the original objective into saddle points of the new one, and whose dynamical behavior is

to seek saddle points. It also seems likely that many useful and general algebraic transformations await discovery.

Although we use the terminology appropriate to the optimization of objectives for dynamic neurons with fixed connections, much of the theory may apply also to “learning” considered as the optimization of dynamic connections in an unstructured net (e.g. (Rumelhart et al., 1986b)) or of some smaller set of parameters which indirectly determine the connections in a structured net (Mjolsness et al., 1989b).

In the remainder of this section, we will introduce the ideas by rederiving a well-known network simplification: that of the winner-take-all (WTA) network from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$ connections. Section 2 develops the theory of our algebraic transformations, including the reduction of: squares and products of expressions; a broad class of functions of one and two expressions; the minimum and maximum of a set of expressions; and certain matrix forms which retain their sparseness as they are reduced. Implications for circuit design are discussed, and a major unsolved problem related to the handling of sparse matrices is stated. Further algebraic transformations (section 2.7) allow control of the temporal aspects of the optimization process, by modifying the usual Lagrangian formalism (which uses variational calculus to derive time-reversible dynamics) to accommodate the need for fixpoints in neural network dynamics. All the fixpoint-preserving transformations are cataloged in section 2.8. In section 3, some of the transformations are exercised in design examples. Experimental results are available for a graph-matching network, a random-dot-matching network, and an approximate sorting network which involves a series of fixpoint-preserving transformations. For all these networks, approach to a fixpoint is guaranteed if the minimizing neurons operate at a much slower time scale than the maximizing neurons, but experimentally such convergence is also observed when the two time scales are close; the advantage of the latter mode of operation is that it requires much less time for a network to converge. Finally, a discussion follows in section 4.

1.1 Reversed Linear Neurons in the WTA Network

Consider the ordinary winner-take-all analog neural network. Following (Hopfield and Tank, 1985), such a network can be obtained from the objective

$$E_{\text{wta}}(\vec{v}) = \frac{c_1}{2} \left(\sum_i v_i - 1 \right)^2 + c_2 \sum_i h_i v_i + \sum_i \phi(v_i) \quad (c_1 > 0) \quad (1)$$

where (Hopfield, 1984; Grossberg, 1988)

$$\phi(v_i) = \int^v dx g^{-1}(x), \quad (2)$$

using steepest-descent dynamics

$$\dot{v}_i = -\partial E/\partial v_i \quad (3)$$

or Hopfield-style dynamics

$$\dot{u}_i = -\partial E/\partial v_i, \quad v_i = g(u_i). \quad (4)$$

The resulting connection matrix is

$$T_{ij} = -c_1,$$

which implies global connectivity among the neurons: if there are N neurons, there are N^2 connections. It is well known that the winner-take-all circuit requires only $\mathcal{O}(N)$ connections if one introduces a linear neuron σ whose value is always $\sum_i v_i$. It is not so well known that this can be done entirely within the objective function, as follows:

$$\hat{E}_{\text{wta}}(\vec{v}) = c_1 \left(\sum_i v_i - 1 \right) \sigma - \frac{c_1}{2} \sigma^2 + c_2 \sum_i h_i v_i + \sum_i \phi(v_i) \quad (5)$$

where the steepest-descent dynamics, for example, is modified to become

$$\begin{aligned} \dot{v}_i &= -r_i \partial \hat{E} / \partial v_i \\ &= -\partial \hat{E} / \partial v_i \quad (\text{specialize to } r_i = 1) \end{aligned} \quad (6)$$

$$\begin{aligned} \dot{\sigma} &= +r_\sigma \partial \hat{E} / \partial \sigma \quad (r_\sigma > 0) \\ &= c_1 r_\sigma (-\sigma + \sum_i v_i - 1) \\ &= 0 \text{ if } \sigma = \sum_i v_i - 1. \end{aligned} \quad (7)$$

But if $\sigma = \sum_i v_i - 1$ then one can calculate that $\partial E/\partial v_i = \partial \hat{E}/\partial v_i$; thus equations (1) and (5) have the same fixpoints. The connectivity implied by counting the monomials in equation (5) is $\mathcal{O}(N)$ connections, the minimum possible for this problem.

Note that the σ linear neuron actually behaves so as to *increase* the objective $E(V, \sigma)$, while the v_i neurons act to decrease it; σ may be called a *reversed* neuron. Reversed neurons introduce a new element of competition into a network; indeed,

two-person zero-sum games are usually modelled using objectives which one player increases and the other decreases (von Neumann and Morgenstern, 1953). So in this network, and in the others we will introduce, minimization is replaced with finding a saddle point and the problem becomes hyperbolic. This follows immediately from the sign of σ^2 in (5), which eliminates all local minima. Fortunately there are hyperbolic versions of such efficient optimization procedures as the conjugate gradient method; Luenberger (Luenberger, 1984) gives two examples.

For finite r_σ , σ is a delayed version of the sum $\sum_i v_i - 1$ and although the network dynamics are different from equation (3), the fixed point is the same. Alternatively, the rate parameter r_σ may be adjusted to make σ move at a different time scale from the rest of the neurons. As r_σ approaches infinity, σ becomes an *infinitely fast* neuron whose value is always

$$\sigma = \sum_i v_i - 1$$

and the other neurons see an effective objective

$$\hat{E}_{\text{wta}}(\vec{v}, \sigma(\vec{v})) = E_{\text{wta}}(\vec{v}) \quad (8)$$

so that their dynamics become identical to that of the original fully connected winner-take-all network, which is guaranteed to approach a fixpoint since $dE_{\text{wta}}/dt < 0$ and E_{wta} is bounded below. There are very efficient serial and parallel implementations for networks with $r_\sigma \rightarrow \infty$, which update the infinitely fast neuron whenever its ordinary neighbors change; this is a standard trick in neural network and Monte Carlo physics simulations. When it is applied to simplify the simulation of the WTA equations of motion, we arrive at the standard WTA trick.

Moody (Moody, 1989) independently discovered an objective function equivalent to (5) and first simulated its delayed Hopfield-style equations of motion (see (4)). But he remained unaware that the σ neuron acts to maximize rather than minimize the objective, driving the system to a saddle point. Platt and Barr (Platt and Barr, 1988; Platt and Barr, 1987) performed constrained optimization (including multiple WTA constraints) by using a subset of neurons that explicitly increase the objective, and hence a network that seeks saddle points rather than local minima. Indeed reversed neurons are a generalization of their analog Lagrange multiplier neurons, which were found earlier in a non-neural context by Arrow (Arrow et al., 1958).

Both reversed neurons and Lagrange multiplier neurons act to maximize an objective which other neurons act to minimize. The difference is that Lagrange multiplier neurons must appear linearly in the objective. General reversed neurons can have self-interactions; in particular, the potential of a linear reversed neuron like σ in the winner-take-all network is

$$-\frac{1}{2g_0}\sigma^2 = \int^\sigma dx g^{-1}(x) \quad (9)$$

where $g(x) = -g_0x$

i.e. linear reversed neurons have linear transfer functions with negative gain. Thus in circuit language they are just inverters, which happen to occur in a network with an objective function and to act so as to increase E . Lagrange multiplier neurons, on the other hand, have no potential term and are not inverters. It is also worth noting that Lagrange multiplier neurons work best in conjunction with an additional penalty term $h(\vec{v})^2/2$ (where $h(\vec{v}) = 0$ is the constraint) and the penalty term can be efficiently implemented using one reversed neuron per constraint, as we will see.

If in addition to being reversed, a neuron is also infinitely fast, then it may be necessary to restrict its connectivity in order to efficiently simulate or implement the network. One possible design rule is to entirely prohibit connections between infinitely fast neurons; this prevents one from having to solve a system of linear equations in order to update a set of infinitely fast neurons. We will *not* generally assume that reversed neurons are infinitely fast.

2 Theory

The reversed linear neuron is applicable in many circumstances beyond the winner-take-all network. We can begin to see its generality by considering objectives of the form

$$E(\vec{v}) = E_0(\vec{v}) + \frac{c}{2}X^2(\vec{v})$$

where E_0 and X are any algebraic expressions, and c is a constant of either sign. This may be transformed to

$$\hat{E}(\vec{v}, \sigma) = E_0 + cX\sigma - \frac{c}{2}\sigma^2$$

and if X is a polynomial, this represents a reduction in the number and order of the monomials that occur in E . The transformation technique used here is simple to state:

find squared expressions $(c/2)X^2$ as summands in the objective function, and replace them with $cX\sigma - (c/2)\sigma^2$. c must be a constant and σ is a new linear interneuron, reversed if c is positive. Most, but not all, of our reversed linear interneurons will be introduced this way. The transformation may be abbreviated as

$$\frac{1}{2}X^2 \rightarrow X\sigma - \frac{1}{2}\sigma^2. \quad (10)$$

We employ the steepest-ascent-descent dynamics

$$\begin{aligned} \dot{v}_i &= -\partial\hat{E}/\partial v_i \\ &= -\partial E_0/\partial v_i - c\sigma\partial X/\partial v_i, \\ \dot{\sigma} &= +(r_\sigma/c)\partial\hat{E}/\partial\sigma \\ &= r_\sigma(X - \sigma) \end{aligned}$$

which, at a fixpoint, has $X = \sigma$ and $\partial E_0/\partial v_i + cX\partial X/\partial v_i = \partial E/\partial v_i = 0$. Likewise a fixpoint of E can be extended, by setting $\sigma = X$, to a fixpoint of \hat{E} . So fixpoints are preserved by the transformation (10). The argument also works if some of the v_i are already reversed neurons, so that $\dot{v}_i = \pm\partial\hat{E}/\partial v_i$.

As an example of the transformation (10), one can robustly implement a constraint $h(\vec{v}) = 0$ using both a penalty term $ch^2/2$ and a Lagrange multiplier neuron λ . The objective becomes

$$E_{\text{constraint}}(\vec{v}, \sigma, \lambda) = c\sigma h(\vec{v}) - c\sigma^2/2 + \lambda h(\vec{v}). \quad (11)$$

2.1 Products and Order Reduction

From the transformation (10) we may deduce two others, which are applicable whenever a summand of an objective is a product XY of expressions X and Y . Such products are common and can be expensive; for example if X and Y are each sums of N variables then expanding their product out into monomial interactions gives N^2 connections. But only $\mathcal{O}(N)$ connections are needed if one uses the transformation

$$\begin{aligned} XY &= \frac{1}{2}(X + Y)^2 - \frac{1}{2}X^2 - \frac{1}{2}Y^2 \\ &\rightarrow (X + Y)\sigma - X\tau - Y\omega - \frac{1}{2}\sigma^2 + \frac{1}{2}\tau^2 + \frac{1}{2}\omega^2 \\ &\rightarrow X(\sigma - \tau) + Y(\sigma - \omega) - \frac{1}{2}\sigma^2 + \frac{1}{2}\tau^2 + \frac{1}{2}\omega^2. \end{aligned} \quad (12)$$

Here σ is a reversed linear neuron, and τ and ω are ordinary linear neurons. If all three linear interneurons are infinitely fast, which is easy to simulate since they are

Figure 1: **Fig 1 goes about here.**

not directly connected, then the transformation does not change the dynamics of the rest of the variables in the network. Otherwise, the dynamics and the basins of attraction change, but the network fixed points remain the same.

This transformation may be simplified,¹ to

$$\begin{aligned}
 XY &= \frac{1}{4} [(X + Y)^2 - (X - Y)^2] \\
 &\rightarrow \frac{1}{2}(X + Y)\sigma - \frac{1}{2}(X - Y)\tau - \frac{1}{4}\sigma^2 + \frac{1}{4}\tau^2 \\
 &= \frac{1}{2}X(\sigma - \tau) + \frac{1}{2}Y(\sigma + \tau) - \frac{1}{4}\sigma^2 + \frac{1}{4}\tau^2.
 \end{aligned} \tag{13}$$

Compared to equation (12) this transformation results in the same number of monomial interactions and one less neuron, which may be useful on occasion.

Reversed neurons allow one to transform a high-order polynomial objective, monomial by monomial, into a third-order objective. Similar transformations on the neural networks or analog circuits are well known. But it is easier to do theoretical work with the objective, and by transforming the objective first, and then translating to a neural net, one can obtain novel third order neural nets.

We may expand a high-order polynomial objective into monomials, each of which corresponds to one connection or “synapse” of the associated neural network. We may reduce the order of an entire objective by reducing the order of each monomial. Consider, then, a single fourth-order monomial:

$$E_{\text{mono}}(x, y, z, w) = -Txyzw \tag{14}$$

which by (12) may be transformed to

$$\hat{E}_{\text{mono}}(x, y, z, w, \sigma, \tau, \omega) = T[xy(\tau - \sigma) + zw(\omega - \sigma) + \frac{1}{2}\sigma^2 - \frac{1}{2}\tau^2 - \frac{1}{2}\omega^2]. \tag{15}$$

Here σ , τ , and ω are linear neurons with gain $1/T$. The order-reducing transformation is illustrated in network language in Figure 1.

The same technique may be used to recursively transform a monomial of any order m to a sum of third order monomials, plus potentials for the new linear interneurons.

¹as pointed out to us by P. Anandan

The resulting number of new monomials and interneurons is $< \mathcal{O}(m(\log m)^2)$ if the reduction is done in a balanced way and if expressions like $X(\sigma - \tau)$ are not expanded out to $X\sigma - X\tau$ during the reduction.

Another order-reduction transformation, superior in some circumstances, will be developed in section 2.4.

2.2 Reducing $F(X)$

Often an objective function includes a fairly general nonlinear function F of an entire expression X . This may be much more difficult and costly to implement directly than either a low-order polynomial or a single-neuron potential function $\phi_i(v_i)$, because the $F(X)$ nonlinearity involves an interaction of many variables. But for some functions F , such an algebraic form is still neurally implementable by means of transformations:

$$\int^X f(u)du \rightarrow X\sigma - \int^\sigma f^{-1}(u)du \quad (f \text{ invertable}) \quad (16)$$

Note that X no longer appears inside the function $F = f f$. The validity of this transformation (equation 16) may be proven by noting that the optimal value of σ is $f(X)$, and then either integrating by parts the expression

$$\int^{f(X)} f^{-1}(u)du = \int^X v f'(v)dv,$$

or else differentiating both pre- and post-transformation objectives with respect to X .

In this way one can treat functions $\exp X$, $|X| \log |X|$, $\log |X|$, and $|X|^p$ of arbitrary algebraic expressions X :

$$\begin{aligned} e^X &\rightarrow (X + 1)\sigma - \sigma \log \sigma \\ |X| \log |X| &\rightarrow |X|(\sigma + 1) - e^\sigma \\ \log |X| &\rightarrow X\sigma - \log |\sigma| \\ \frac{1}{1+p}|X|^{1+p} &\rightarrow X\sigma - \frac{1}{1+1/p}|\sigma|^{1+1/p} \quad (p \neq -1, 0). \end{aligned} \quad (17)$$

Thus, neural nets may be constructed from some highly nonpolynomial objectives.

The interneurons may still be reversed, but are no longer linear, in this kind of transformation. The potential $\phi(\sigma)$ permits a possible efficiency technique. The dynamics of equation (4) is expected to be more efficient than equation (3), since it may be viewed as a quasi-Newton method which takes into account the potential but not the interaction part of a neural net objective (as shown by J. Utans (Utans et al., 1989)). A related update scheme for the σ reversed interneuron is

$$\begin{aligned} \sigma &= f(s), & \dot{s} &= r_\sigma \partial E / \partial \sigma = r_\sigma (X - s) \\ v_i &= g_i(u_i), & \dot{u}_i &= -\partial E(\vec{v}, \sigma) / \partial v_i \end{aligned} \quad (18)$$

which is an alternative to direct steepest-ascent-descent. This dynamics has the distinct advantage of a simple interpretation in terms of analog electrical circuits (Hopfield, 1984). For example, $F(X) = X(\log X - 1)$ requires a special neuron whose transfer function is logarithmic. This can be provided, approximately and within a mildly restricted domain of the input values, in analog VLSI (Sivilotti et al., 1987). Similarly $F(X) = \log X$ would require a transfer function $f(s) = 1/s$, and an exponential transfer function would lead to $F(X) = \exp X$. It might be possible to characterize a particular technology by a list of the basic forms of objectives it makes available, with their respective costs and restrictions, and to compile general networks into the desired forms by using a catalog of algebraic transformations.

Equation (16) generally may be used to transform a term $F(X)$ in an objective by transferring the nonlinearity due to F from $F(X)$ to the single-neuron potential $\phi(\sigma) = \int^\sigma f^{-1}(u) du$. By transferring the nonlinearity from an interaction term (i.e. a summand of the objective which involves several dynamical variables) to a single-neuron potential term, one can not only decrease the cost of implementing a network which uses gradient methods for optimization, but one can transform unimplementable objectives into implementable ones. For example, one might regard the class of multi-variable polynomials as the “implementable” interactions in a certain technology (Rumelhart et al., 1986a). (In this case the word “interaction” is usually reserved for a multivariable monomial, out of which polynomials are built by addition of objectives.) Then one might use equation (16) to reduce other, far more general interaction objectives to the implementable form.

Of course the required potential $\phi(\sigma)$ may itself be “unimplementable”, but approximating its gradient with a small circuit is likely to be far more tractable than approximating $\nabla F(X)$ because ϕ , unlike F , is a function of just one dynamic variable.

An approximation of $\phi'(\sigma)$ might be formulated as

$$\phi(\sigma) \rightarrow \hat{\phi}(\sigma) = \sum_{\alpha} c_{\alpha} \hat{\phi}_{\alpha}(\sigma)$$

where each $\hat{\phi}_{\alpha}(\sigma)$ is regarded as an implementable self-interaction and c_{α} are adjustable coefficients.

2.3 Reducing $\mathbf{F}(\mathbf{X})$: Examples

We exhibit two examples of the objective function transformation of equation (16), with dynamics (18). First consider the toy problem of optimizing

$$E(x) = e^x + e^{-x}.$$

One of the exponentials may be taken to be the potential of a neuron whose transfer function is exponential; such a transfer function is implementable in many technologies and, like the logarithmic transfer function, might be part of a standard component library for analog neural nets. Adding the other exponential would however require a special modification to this transfer function, and their sum might not be in the standard component library. So let us move the second exponential nonlinearity to another neuron whose transfer function is logarithmic:

$$\hat{E}(x, \sigma) = e^x - x\sigma - \sigma \log \sigma + \sigma$$

whence the dynamics (18) become

$$\begin{aligned} \dot{\sigma} &= r_{\sigma}(-x - s) \\ \dot{x} &= r_x(\sigma - u). \end{aligned} \tag{19}$$

The evolution of this two-neuron network is shown in Figure 2, along with a contour map of the saddle-shaped objective \hat{E} . Note that for quick descent, $r_x \approx r_{\sigma}$ is preferred. Despite the saddle point, and despite the potential numerical sensitivity of exponential and logarithmic transfer functions, the network functions well.

A second example is the linear programming network of (Tank and Hopfield, 1986) which can also be interpreted as an application of transformation (16) with $r_{\sigma} \rightarrow \infty$

Figure 2: **Fig 2 goes about here.**

in the dynamics of equation (18). The linear programming problem is to minimize $\vec{A} \cdot \vec{v}$ subject to a set of constraints $\vec{D}_j \cdot \vec{v} \geq B_j$. Their objective is

$$E[\vec{v}] = \sum_i A_i v_i + \sum_j F\left(\sum_i D_{ji} v_i - B_j\right) + \sum_i v_i^2 / g_0 \quad (\text{large } g_0),$$

where $dF(x)/dx = f(x) = \max(0, -x)$ penalizes violations of the inequality constraints and proved to be electronically implementable. Transforming according to (16) we get

$$\hat{E}[\vec{v}] = \sum_i A_i v_i + \sum_j \sigma_j \left(\sum_i D_{ji} v_i - B_j \right) - \sum_j \int^{\sigma_j} f^{-1}(s) ds + \sum_i v_i^2 / g_0$$

and equations of motion

$$\begin{aligned} \sigma_j &= f(s_j), & \dot{s}_j &= r_\sigma \left(\sum_i D_{ji} v_i - B_j - s_j \right) \\ v_i &= g_0 u_i, & \dot{u}_i &= r_v \left(-u_i - A_i - \sum_j D_{ji} \sigma_j \right). \end{aligned} \quad (20)$$

This network approaches a saddle point rather than a minimum, but the $r_\sigma \rightarrow \infty$ version,

$$\dot{u}_i / r_v = -u_i - A_i - \sum_j D_{ji} f\left(\sum_i D_{ji} v_i - B_j\right)$$

is exactly the network dynamics of equation (17) of (Tank and Hopfield, 1986).

2.4 Interacting Expressions: Reducing $G(X, Y)$

Until now we have attempted to reduce all interactions to the forms xy and xyz , but those may not be the only cost-effective few-variable interactions allowed by a given physical technology. If others are allowed, then there are one-step transformations for a class of functions of two arbitrary expressions $G(X, Y)$. In this way the set of objectives with known low-cost neural implementations can be expanded to include common algebraic forms such as X/Y and $XF(Y)$.

From equation (16) we may derive a generalization to functions of two expressions, $G(X, Y)$:

$$\begin{aligned}
 \int^X du \left[\int^Y dv g(u, v) \right]^{-1} (u) & \xrightarrow{\text{[note function inverse]}} X\sigma - \int^\sigma du \int^Y dv g(u, v) \quad \text{[by (16)]} \\
 & = X\sigma - \int^Y dv \int^\sigma du g(u, v) \\
 & \rightarrow X\sigma - Y\tau + \int^\tau dv \left[\int^\sigma du g(u, v) \right]^{-1} (v) \quad \text{[by (16)]}.
 \end{aligned}$$

Thus,

$$\int^X du \left[\int^Y dv g(u, v) \right]^{-1} (u) \rightarrow X\sigma - Y\tau + \int^\tau dv \left[\int^\sigma du g(u, v) \right]^{-1} (v) \quad (21)$$

Of course the inverse functions must exist for this transformation to be valid, and this restricts G .

Taking

$$g(u, v) = \frac{1}{2\sqrt{uv}}$$

we can derive the transformation $-Y/X \rightarrow X\sigma - Y\tau - \sigma/\tau$. Rescaling Y and σ by -1 , then switching X for Y and σ for τ , this is equivalent to

$$X/Y \rightarrow X\sigma - Y\tau + \tau/\sigma \quad (22)$$

which is linear in τ but effectively nonlinear due to the optimization of σ .

From equation 21, Appendix A derives two transformations for the special form $YF(X)$:

$$YF(X) \rightarrow -X\sigma + Y\tau + \sigma F^{-1}(\tau) \quad (23)$$

(which implies (22)) and

$$Y \int^X f(u) du \rightarrow XY\sigma - Y \int^\sigma du f^{-1}(u) \quad (24)$$

assuming $f = F'$ is invertable and $f^{-1} = (F')^{-1}$ is differentiable.

Monomial order reduction can sometimes be accomplished more cheaply using $x \log y$ interactions than third-order ones. If v_i are all restricted to be positive, then

$$\begin{aligned} \prod_{i=1}^m v_i &= \exp \sum \log v_i \\ &\rightarrow \sigma (\sum_i \log v_i + 1) - \sigma \log \sigma \\ &= \sum_i \sigma \log v_i + \sigma (1 - \log \sigma). \end{aligned} \quad (25)$$

This objective has $\mathcal{O}(m)$ interactions of the new type. The fixpoint value of σ is $\prod_i v_i$ at which point the steepest-descent input to v_i is $-\sigma/v_i = -\prod_{j \neq i} v_j$.

A product of expressions could be further reduced using equation (23) and $x e^y$ interactions:

$$\prod_{\alpha=1}^m |X_\alpha| \rightarrow \sum_{\alpha} (\sigma \tau_\alpha - |X_\alpha| \omega_\alpha + \omega_\alpha e^{\tau_\alpha}) + \sigma (1 - \log \sigma). \quad (26)$$

It has been pointed out to us (Simic, 1989) that the transformations for $F(X)$ and $G(X, Y)$, and hence all the transformations discussed so far, can be interpreted as Legendre transformations (Courant and Hilbert, 1962).

2.5 Min and Max

The minimum or maximum of a set of expressions $\{X_\alpha\}$ can be implemented using a winner-take-all network in which each expression is represented by a neuron σ_α which competes with the others and is constrained to lie between 0 and 1. Indeed, $\sum_{\alpha} X_\alpha \sigma_\alpha$ attains the value $\min_{\alpha} X_\alpha$ when the correct representative wins, and also provides inhibition to the representatives in proportion to the values of the expressions they represent, so that the correct representative will win. The potential $\phi(\sigma)$ that occurs in the WTA network must have only one minimum, so that there is no hysteresis in the circuit, and must closely approximate a square well i.e. must have high gain. Under these circumstances, we can transform

$$E = \min_{\alpha} X_\alpha \rightarrow \sum_{\alpha} X_\alpha \sigma_\alpha + C(\sum_{\alpha} \sigma_\alpha - 1)\lambda - \frac{C}{2}\lambda^2 + \sum_{\alpha} \phi(\sigma_\alpha) \quad (27)$$

(where C and the gain of ϕ are sufficiently large) and at any fixpoint of $\vec{\sigma}$ the derivatives of E with respect to all other dynamical variables will be preserved. So, fixpoints will be preserved.

Likewise

$$\max_{\alpha} X_{\alpha} \rightarrow \sum_{\alpha} X_{\alpha} \sigma_{\alpha} - C(\sum_{\alpha} \sigma_{\alpha} - 1)\lambda + \frac{C}{2}\lambda^2 - \sum_{\alpha} \phi(\sigma_{\alpha}). \quad (28)$$

An alternate transformation for max proceeds through the identity

$$\max_{\alpha} X_{\alpha} = \lim_{p \rightarrow \infty} \left(\sum_{\alpha} X_{\alpha}^p \right)^{1/p} \quad (X_{\alpha} > 0). \quad (29)$$

These transformations have application in the design of some standard “content addressable memories” for which the ideal objective, which must be translated to a form polynomial in its interactions, may be taken as

$$E_{\text{CAM}}(\vec{v}) = - \max_{\text{memories } m} \vec{v} \cdot \vec{v}^m - \epsilon \vec{v} \cdot \vec{h}_{\text{input}} + \sum_i \phi_{\pm 1}(v_i)$$

with $-1 \leq v_i \leq 1$. Using the transformation (28) yields a CAM with one “grandmother neuron” (representative neuron) per memory, and a WTA net among them (closely related to a network described in (Moody, 1989)):

$$\hat{E}_{\text{CAM}}(\vec{v}, \vec{\sigma}) = - \sum_{m,i} v_i^m v_i \sigma_m - \epsilon \vec{v} \cdot \vec{h}_{\text{input}} + C(\sum_m \sigma_m - 1)\lambda - C\lambda^2/2 + \sum_m \phi_{0/1}(\sigma_m) + \sum_i \phi_{\pm 1}(v_i).$$

Another efficient CAM design (Gindi et al., 1987; Moody, 1989) may be derived by applying transformation (29) to the max expression:

$$\max_m |\vec{v} \cdot \vec{v}^m| \approx \left(\sum_m |\vec{v} \cdot \vec{v}^m|^p \right)^{1/p} \quad (p \text{ large})$$

which we replace by a monotonic function thereof, $(1/p) \sum_m |\vec{v} \cdot \vec{v}^m|^p$, possibly adjusting ϵ and ϕ to compensate. (At this point one could take $p = 2$ to get a quadratic expression, and regenerate the content addressable memory of (Hopfield, 1984).) Then using (17d) we find an implementable neural net objective for large p :

$$\hat{E}_{\text{CAM}}(\vec{v}, \vec{\sigma}) = - \sum_{m,i} v_i^m v_i \sigma_m - \epsilon \vec{v} \cdot \vec{h}_{\text{input}} + \frac{(p-1)}{p} \sum_m |\sigma_m|^{p/(p-1)} + \sum_i \phi_{\pm 1}(v_i).$$

2.6 Matrices, Graphs, and Pointers

One can apply order reduction to objectives containing polynomials of matrices. For dense $N \times N$ matrices, a typical term like $\text{tr} \prod_{l=1}^L A^{(l)}$ contains N^L scalar monomial interactions, but this can be reduced to $\mathcal{O}(N^3 L (\log L)^2)$. (Here $\text{tr} A \equiv$ trace of $A \equiv \sum_i A_{ii}$.) To show this we need only establish the matrix analog of transformation (12), which upon iteration can reduce an L -th order matrix monomial to $\mathcal{O}(L (\log L)^2)$ third-order matrix monomials like ABC . Each of these involves N^3 scalar monomial interactions.

Using equation (12) (one could also use (13)), one can reduce $\text{tr} XY$, where X and Y are now matrix-valued expressions. This form has exactly the same generality as $\text{tr} XY^T$ (where $Y^T \equiv$ transpose of Y).

$$\begin{aligned}
 \text{tr} XY^T &= \sum_{ij} X_{ij} Y_{ij} \\
 &\rightarrow \sum_{ij} \left(X_{ij} (\sigma_{ij} - \tau_{ij}) + Y_{ij} (\sigma_{ij} - \omega_{ij}) - \frac{1}{2} \sigma_{ij}^2 + \frac{1}{2} \tau_{ij}^2 + \frac{1}{2} \omega_{ij}^2 \right) \\
 &= \text{tr} X (\sigma - \tau)^T + \text{tr} Y (\sigma - \omega)^T - \frac{1}{2} \text{tr} \sigma \sigma^T + \frac{1}{2} \text{tr} \tau \tau^T + \frac{1}{2} \text{tr} \omega \omega^T
 \end{aligned} \tag{30}$$

This transformation preserves the sparseness of X and Y in the following sense: if $X_{ij} = Y_{ij} = 0$ at a fixed point, then $\sigma_{ij} = \tau_{ij} = \omega_{ij} = 0$ and the contribution of these neurons to the gradient of the objective is also zero.

A major problem in neural network research (c.f. (Feldman, 1982)) is to reduce the cost of networks which manipulate graphs. Usually (Hopfield and Tank, 1985; Hopfield and Tank, 1986; Mjolsness et al., 1989a) objectives for such problems involve dense matrices of neurons representing all the possible links in a graph. But the graphs that arise in computer science and in computer programming usually have a relatively small number of links per node, and are therefore representable by sparse matrices. (If a sparse matrix's entries are all zero or one, it is equivalent to a set of "pointers" in many current computer languages. Pointers are used ubiquitously, wherever some fluidity in data representation is required.) Since we have just shown how to reduce a wide class of matrix objective functions to summands of the form $\text{tr} ABC$ while retaining sparseness, it becomes important to reduce this form further by exploiting the sparseness of the matrices involved:

$$\text{tr} ABC \rightarrow ?$$

where A , B , and C are sparse-matrix-valued dynamical variables. We do not yet

know how to do this correctly.

One approach to this problem is through the use of codes, like the binary code, which can concisely name the pair of nodes connected by each nonzero matrix element. Zero matrix elements are not explicitly encoded and this is the advantage of the method. A disadvantage of such codes, for objective functions, is that the configuration space is altered in such a way that new local minima may be introduced, though the old ones will be preserved. A code which allows order reduction to proceed most advantageously is used in the sorting networks of section 3.4.

Another approach is used by Fox and Furmanky (Fox and Furmanky, 1988). Their load-balancing network involves binary encoding, but the network evolution is divided into a number of phases in which different classes of neurons are allowed to vary while most neurons are held constant. The connections are different from one phase to the next, and do not recur, so that the network is not directly implemented in terms of a circuit but rather requires “virtual” neurons and connections. A virtual neural network can be provided by suitable software on general-purpose computers, or perhaps by further objective function transformations leading to a real (and efficient) neural circuit; the latter alternative has not been achieved.

2.7 Control of Neural Dynamics

So far we have exclusively considered steepest-ascent-descent dynamics such as equations (3), (4), or (18), which allow little control over the temporal behavior of a network. Often one must design a network with nontrivial temporal behaviors such as running longer in exchange for less circuitry, or focussing attention on one part of a problem at a time. We discuss two algebraic transformations which can be used to introduce detailed control of the dynamics with which an objective is extremized.

One transformation, developed by one of the authors in collaboration with W. Miranker (Mjolsness and Miranker, 1990), replaces an objective E with an associated Lagrangian functional to be extremized in a novel way:

$$E[\vec{v}] \rightarrow L[\dot{\vec{v}}|\vec{q}] = \int dt \left(K[\dot{\vec{v}}, \vec{v}|\vec{q}] + \frac{dE}{dt} \right), \quad \delta L / \delta \dot{v}_i(t) = 0. \quad (31)$$

Here \vec{q} is a set of control parameters, and K is a cost-of-movement term independent of the problem and of E . The integrand $\mathcal{L} = K + dE/dt$ is called the “Lagrangian density”. Ordinarily in physics, Lagrangian dynamics have a conserved total energy

which prohibits convergence to fixed points. Here the main difference is the unusual functional derivative with respect to \dot{v} rather than v . This is a “greedy” functional derivative, in which the trajectory is optimized from beginning to end by repeatedly choosing an extremal value of $\vec{v}(t)$ without considering its effect on any subsequent portion of the path:

$$\frac{\delta}{\delta v_i(t)} \int_{-\infty}^t dt' \mathcal{L}[\dot{\vec{v}}, \vec{v}] \approx \delta(0) \frac{\partial \mathcal{L}[\dot{\vec{v}}, \vec{v}]}{\partial \dot{v}_i(t)} = \delta(0) \frac{\delta}{\delta \dot{v}_i(t)} \int_{-\infty}^{\infty} dt' \mathcal{L}[\dot{\vec{v}}, \vec{v}] \propto \frac{\delta L}{\delta \dot{v}_i(t)}. \quad (32)$$

Since

$$\frac{\delta L}{\delta \dot{v}_i} = \frac{\partial K}{\partial \dot{v}_i} + \frac{\partial E}{\partial v_i}, \quad (33)$$

transformation (31) preserves fixpoints if $\partial K / \partial \dot{v}_i = 0 \Leftrightarrow \dot{\vec{v}} = 0$.

For example, with a suitable K one may recover and improve upon steepest-ascent-descent dynamics:

$$\begin{aligned} E[\vec{v}] &\rightarrow L[\dot{\vec{v}}|r, \vec{s}] = \int dt \left(\sum_i s_i \phi_{\pm 1}(\dot{v}_i/r) + \sum_i (\partial E / \partial v_i) \dot{v}_i \right), \\ 0 &= \delta L / \delta \dot{v}_i(t) = s_i \phi'_{\pm 1}(\dot{v}_i/r) / r + dE / dv_i, \text{ i.e.} \\ \dot{v}_i &= r g_{\pm 1} \left(-s_i r dE / dv_i \right) \end{aligned} \quad (34)$$

where the transfer function $-1 \leq g_{\pm 1}(x) \leq 1$ reflects a velocity constraint $-r \leq \dot{v}_i \leq r$, and as usual $g = (\phi')^{-1}$. The constants $s_i = 1/s_i = \pm 1$ are used to determine whether a neuron attempts to minimize or maximize E and L . If all $s_i = 1$ then $dE/dt \leq 0$ and equation (34) is a descent dynamics.

Another transformation (proposed and subjected to preliminary experiments in (Mjolsness, 1987)) can be used to construct a new objective for the control parameters, q , through their effect on the trajectory $v(t)$:

$$\begin{aligned} E[\vec{v}] \rightarrow \hat{E}[\vec{q}] &= \sum_i (\partial E / \partial v_i) s_i \dot{v}_i + \hat{E}_{\text{cost}}[\vec{q}] \\ &= dE/dt + \hat{E}_{\text{cost}}[\vec{q}], \quad \text{if all } s_i = 1. \end{aligned} \quad (35)$$

In (Mjolsness, 1987) the $s_i = 1$ version of transformation (35) (but not (34)) was used to introduce a computational “attention mechanism” for neural nets as follows. Suppose we can only afford to simulate R out of $N \gg R$ neurons at a time in a large net. The R neurons can be chosen dynamically via control parameters $q_i = r_i \in [0, 1]$

in equations (34), with $r_i \approx 0$ for all but R active neurons. For high-gain $g_{\pm 1}$ we have $g_{\pm 1}(x) \approx \text{sgn}(x)$ and

$$\frac{dE}{dt} = \sum_i \frac{dE}{dv_i} \dot{v}_i = \sum_i r_i \frac{dE}{dv_i} g_{\pm 1} \left(-r_i \frac{dE}{dv_i} \right) \approx - \sum_i r_i \left| \frac{dE}{dv_i} \right|.$$

(Whether the gain is high or not, $g_{\pm 1}$ is an odd function so $dE/dt \leq 0$ and any dynamics for r yields a descent algorithm for E .) The objective for \vec{r} is

$$\hat{E}[\vec{r}] \approx - \sum_i r_i \left| \frac{dE}{dv_i} \right| + \frac{A}{2} \left(\sum_i r_i - R \right)^2 + \sum_i \phi_{0/1}(r_i)$$

which describes an R -winner version of a WTA network that determines which R neurons should be active and which $N - R$ should be frozen.

An especially simple and cheap dynamical system for r is to keep all r_i constant most of the time, but every so often to interrupt the simulation of $\delta L / \delta \dot{v} = 0$ and completely relax $\hat{E}[\vec{r}]$. This amounts to re-sorting the neurons v_i according to their gradient magnitudes $|\partial E / \partial v_i|$, and selecting only the first R neurons to be active ($r_i \approx 1$). When simulating a sparsely connected neural net on other underlying hardware, this algorithm can be implemented very cheaply since most v gradients are unchanged between phases of r relaxation, and therefore the new sorted order is just a minor refinement of the previous one. The result is necessarily a descent algorithm for $E[\vec{v}]$, with only R neurons active at any time.

2.8 List of Transformations

Let X and Y be any algebraic expressions, containing any number of variables. We list the following fixpoint-preserving transformations of objectives, or summands thereof:

1.1	$\frac{1}{2}X^2 \rightarrow X\sigma - \frac{1}{2}\sigma^2$
1.2	$XY \rightarrow X(\sigma - \tau) + Y(\sigma - \omega) - \frac{1}{2}\sigma^2 + \frac{1}{2}\tau^2 + \frac{1}{2}\omega^2$
1.3	$XY \rightarrow \frac{1}{2}X(\sigma - \tau) + \frac{1}{2}Y(\sigma + \tau) - \frac{1}{4}\sigma^2 + \frac{1}{4}\tau^2$
2.1	$\int^X f(u)du \rightarrow X\sigma - \int^\sigma f^{-1}(u)du \quad (f \text{ an invertable function})$
2.2	$e^X \rightarrow (X + 1)\sigma - \sigma \log \sigma$
2.3	$ X \log X \rightarrow X (\sigma + 1) - e^\sigma$
2.4	$\log X \rightarrow X\sigma - \log \sigma $
2.5	$\frac{1}{1+p} X ^{1+p} \rightarrow X\sigma - \frac{1}{1+1/p} \sigma ^{1+1/p} \quad (p \neq -1, 0).$
3.1	$\int^X du \left[\int^Y dv g(u, v) \right]^{-1}(u) \rightarrow X\sigma - Y\tau + \int^\tau dv \left[\int^\sigma dug(u, v) \right]^{-1}(v)$ (If function inverses exist.)
3.2	$YF(X) \rightarrow -X\sigma + Y\tau + \sigma F^{-1}(\tau) \quad (\text{If } ((F')^{-1})' \text{ exists.})$
3.3	$Y \int^X f(u)du \rightarrow XY\sigma - Y \int^\sigma du f^{-1}(u) \quad (\text{If } (f^{-1})' \text{ exists.})$
3.4	$X/Y \rightarrow X\sigma - Y\tau + \tau/\sigma$
4.1	$\min_\alpha X_\alpha \rightarrow \sum_\alpha X_\alpha \sigma_\alpha + C(\sum_\alpha \sigma_\alpha - 1)\lambda - \frac{C}{2}\lambda^2 + \sum_\alpha \phi(\sigma_\alpha)$ (Large C , and high-gain hysteresis-free barrier function ϕ which confines σ_α to $(0, 1)$.)
4.2	$\max_\alpha X_\alpha \rightarrow \sum_\alpha X_\alpha \sigma_\alpha - C(\sum_\alpha \sigma_\alpha - 1)\lambda + \frac{C}{2}\lambda^2 - \sum_\alpha \phi(\sigma_\alpha)$ (Same conditions.)
4.3	$\prod_\alpha X_\alpha \rightarrow \sum_\alpha (\sigma\tau_\alpha - X_\alpha \omega_\alpha + \omega_\alpha e^{\tau_\alpha}) + \sigma(1 - \log \sigma).$
5.1	$\text{tr } XY^T \rightarrow \text{tr } X(\sigma - \tau)^T + \text{tr } Y(\sigma - \omega)^T$ $-\frac{1}{2}\text{tr } \sigma\sigma^T + \frac{1}{2}\text{tr } \tau\tau^T + \frac{1}{2}\text{tr } \omega\omega^T$ (All matrices, possibly sparse.)
6.1	$E[\vec{v}] \rightarrow L[\dot{\vec{v}} \vec{q}] = \int dt \left(K[\dot{\vec{v}} \vec{v}, \vec{q}] + \frac{dE}{dt} \right), \quad \delta L/\delta \dot{\vec{v}}(t) = 0$ ($\partial K/\partial \vec{v} = 0 \Leftrightarrow \dot{\vec{v}} = 0$)
6.2	$E[v] \rightarrow \hat{E}[q] = \sum_i (\partial E/\partial v_i) s_i \dot{v}_i + \hat{E}_{\text{cost}}[q] \quad (s_i = \pm 1)$

The variables σ , τ , ω , and λ are assumed not to occur elsewhere in the original objective. Note that each transformation may have restrictions on its applicability, in addition to the particular form it matches.

We will report experiments only with transformations 1.1, 1.2, 2.2, and 5.1 on this list. Experiments with transformations 6.1 and 6.2 will be reported in a later paper (Mjolsness and Miranker, 1990). The rest are still theoretical.

These transformations may be iterated, at the expense of creating interactions between the added variables. They can be used to reduce the nonlinearity of the interactions in a neural network, transferring such nonlinearity to single-neuron potentials or distributing it among several simpler interactions.

3 Design Examples

3.1 Convolutions and Coordinate Transformations

Discrete convolutions

$$O_i = \sum_j K_{i-j} I_j$$

(where index subtraction is defined appropriately) and linear coordinate transformations

$$x'_i = \sum_j A_{ij} x_j + b_i$$

can both be expressed as sums of squared penalty terms in an objective:

$$E_{\text{conv}} = \frac{c}{2} \sum_i (O_i - \sum_j K_{i-j} I_j)^2 \quad (37)$$

or

$$E_{\text{coord}} = \frac{c}{2} \sum_i (x'_i - \sum_j A_{ij} x_j + b_i)^2, \quad (38)$$

with $c > 0$. (Equation (38) subsumes equation (37).) Alternatively the convolution or coordinate change could be turned into a hard constraint by using Lagrange multiplier neurons, but those procedures still work best when a penalty term exactly like equation (37) or equation (38) is added to the objective (Platt and Barr, 1988; Luenberger, 1984). As they stand, these objectives expand into very expensive networks due to the spurious squaring of the matrix. That is because convolution kernels,

which are usually constant but sparse, have their fanout squared; and coordinate transformations, which are usually dense but variable, have an excessive number of new (high-order) interactions created when A is squared.

Of course, equation (38) is of the type which we know how to transform using reversed linear neurons. We obtain the modified objective

$$\hat{E}_{\text{coord}} = c \left[\sum_i (x'_i - \sum_j A_{ij} x_j - b_i) \sigma_i - \frac{1}{2} \sum_i \sigma_i^2 \right] \quad (39)$$

which doesn't square A . If A is constant then there is no order reduction, since both E_{coord} and \hat{E}_{coord} are second order, but there are fewer connections unless A is also dense.

An alternative objective, not using reversed neurons, is also available for convolutions and coordinate system transformations. The objective

$$\hat{E}_{\text{coord}} = \frac{c}{2} \sum_{ij} A_{ij} (x'_i - b_i - x_j)^2 \quad (40)$$

is minimal with respect to x' when

$$x'_i = \sum_j A_{ij} x_j / \sum_j A_{ij} + b_i. \quad (41)$$

This type of dynamic normalization may be desirable, or if A is constant and already normalized then it does not hurt. Equation (40) also preserves any sparseness of A , and does not square the matrix.

3.2 Random Dot Matching

Here the problem begins with two inputs: a planar pattern of n dots specified by their independent random positions \vec{x}_i within a unit square, and a pattern of $m \ll n$ dots specified by their positions \vec{y}_a . The \vec{y}_a are generated by randomly selecting m of the \vec{x} dots, independently perturbing their positions by random displacements of 1/10 the size of the square, and globally shifting all m dots by a translation vector $\vec{\Delta}$. The problem is to reconstruct $\vec{\Delta}$ from $\{\vec{x}_i\}$ and $\{\vec{y}_a\}$, by consistently matching the dots. Since the match is parameterized by a few geometric parameters, this is really an image "registration" problem.

A simple objective for this task is

$$E_{\text{dots}}[\vec{\Delta}] = - \sum_i \sum_a \exp \left(- |\vec{x}_i - \vec{y}_a - \vec{\Delta}|^2 / 2K^2 \right) \quad (42)$$

where the search width K is to be set initially to the width of the square (1.0) and gradually decreased down to the size of the geometric noise (0.1) as the optimization proceeds, in order to find better local minima. This objective, and the gradual change in K , is quite similar to that of the “elastic net” approach to the Traveling Salesman Problem (Durbin and Willshaw, 1987). Now E_{dots} may be transformed to remove the exponential from the interactions:

$$\hat{E}_{\text{dots}}[\vec{\Delta}, \tau] = \frac{1}{2K^2} \sum_{ia} |\vec{x}_i - \vec{y}_a - \vec{\Delta}|^2 \tau_{ia} + \sum_{ia} \tau_{ia} (\log \tau_{ia} - 1) \quad (43)$$

with dynamics

$$\begin{aligned} \dot{\vec{\Delta}} &= (1/K^2) \sum_{ia} \tau_{ia} (\vec{x}_i - \vec{y}_a - \vec{\Delta}) \\ \tau_{ia} &= \exp \omega_{ia} \\ \dot{\omega}_{ia} &= -\omega_{ia} - (1/2K^2) \sum_{ia} |\vec{x}_i - \vec{y}_a - \vec{\Delta}|^2 \end{aligned} \quad (44)$$

We experiment with $n = 30$ and $m = 6$. In Figure 3 we have shown a contour map of E_{dots} , which is to be minimized, along with the projection of a $K = .2$ trajectory onto the $\vec{\Delta}$ plane. The initial condition came from partially relaxing the net at $K = .5$ first, where the objective is unimodal. The $K = .2$ problem is somewhat more difficult than incrementally updating a solution in response to a small change in K , but the network found the right answer. For $n = 30$ and $m = 6$, some random patterns of dots would require several large- K minima to be tracked to small K for correct operation, but this defect of the original objective (42) did not arise for the case shown here (or 13 out of 15 other cases we examined) and is not relevant to the validity of the transformation.

The main numerical drawback of the 180 exponential-taking neurons in this net is that small time steps may be required. In using the Runge-Kutta method for solving equations (44) the stepsize had to be $\Delta t = .0003$ near the starting point, even though eventually it could be increased to .003.

Figure 3: **Fig 3 goes about here.**

3.3 Graph Matching and Quadratic Match

Consider the following objective for inexact graph-matching (Hopfield and Tank, 1986),

c.f. (von der Malsburg and Bienenstock, 1986):

$$\begin{aligned}
 E_{\text{graph}} = & -c_1 \sum_{\alpha\beta ij} G_{\alpha\beta} g_{ij} M_{\alpha i} M_{\beta j} \\
 & + c_2 \sum_{\alpha} (\sum_i M_{\alpha i} - 1)^2 + c_2 \sum_i (\sum_{\alpha} M_{\alpha i} - 1)^2 \\
 & + c_3 \sum_{\alpha i} M_{\alpha i} (1 - M_{\alpha i}) \\
 & + \sum_{\alpha i} \int^{M_{\alpha i}} dx g^{-1}(x)
 \end{aligned} \tag{45}$$

where G and g are connection matrices (each entry is zero or one) for two graphs, and $M_{\alpha i}$ is one when node α of G maps to node i of g , and zero otherwise.

The problem may be generalized slightly to “quadratic matching” by replacing the $GgMM$ term with

$$\sum_{\alpha\beta ij} G_{\alpha\beta} g_{ij} A_{\alpha i} B_{\beta j} \tag{46}$$

and altering the other constraints to reflect the fact that $A \neq B$. What we have to say about graph matching will apply equally well to this generalization.

The $GgMM$ term is superficially the expensive one since it involves four sums. If each graph is constant, there are $\mathcal{O}(N)$ nodes in each graph, and both graphs have fanout f , then the number of monomial synapses is $\mathcal{O}(N^2 f^2)$. We can reduce this to $\mathcal{O}(N^2 f)$. Also if one of G or g is variable, with N nodes in the variable graph and m in the constant graph, as in the “Frameville” networks of (Mjolsness et al., 1989a), and both graphs are represented densely, then the number of synapses is reduced from $\mathcal{O}(N^2 m f)$ to $\mathcal{O}(N^2 m + N m f)$. The reduction uses linear interneurons, both reversed

Figure 4: **Fig 4 goes about here.**

and normal:

$$\begin{aligned}
E_1 &= -c_1 \sum_{\alpha\beta ij} G_{\alpha\beta} g_{ij} M_{\alpha i} M_{\beta j} \\
&= -\frac{c_1}{2} \sum_{\beta i} \left[(\sum_{\alpha} G_{\alpha\beta} M_{\alpha i} + \sum_j g_{ij} M_{\beta j})^2 \right. \\
&\quad \left. - (\sum_{\alpha} G_{\alpha\beta} M_{\alpha i})^2 - (\sum_j g_{ij} M_{\beta j})^2 \right] \\
\hat{E}_1 &= -c_1 \left[\sum_{\alpha\beta i} G_{\alpha\beta} M_{\alpha i} \sigma_{\beta i} + \sum_{\beta ij} g_{ij} M_{\beta j} \sigma_{\beta i} \right. \\
&\quad \left. - \sum_{\alpha\beta i} G_{\alpha\beta} M_{\alpha i} \tau_{\beta i} - \sum_{\beta ij} g_{ij} M_{\beta j} \omega_{\beta i} \right. \\
&\quad \left. - \frac{1}{2} \sigma_{\beta i}^2 + \frac{1}{2} \tau_{\beta i}^2 + \frac{1}{2} \omega_{\beta i}^2 \right]
\end{aligned} \tag{47}$$

E_1 and \hat{E}_1 are illustrated in Figure 4.

The reduced graph-matching network works in simulation, for five out of six small ($N = 10$ nodes) hand-designed graphs with low fanout (from 1.8 to 2.5). The sixth case is not solved by the original, untransformed network either. The parameters we used were

$$\begin{array}{llll}
N = 10 & c_1 = 1.0 & c_2 = 1.0 & c_3 = 0.5 \\
g_0(M) = 20 & g_0(\sigma) = 1.0 & g_0(WTA) = 10.0 & r_{\sigma} = 1 \\
\Delta t = .004 & \text{sweeps} = 1\ 000 & &
\end{array}$$

and for the original network:

$$\begin{array}{llll}
N = 10 & c_1 = 1.0 & c_2 = 1.0 & c_3 = 0.5 \\
g_0(M) = 20 & g_0(\sigma) = 1.0 & g_0(WTA) = 10.0 & \Delta t = .004 . \\
\text{sweeps} = 1\ 000 & & &
\end{array}$$

Here $g_0(M)$ is the gain $g'(0)$ of the transfer function $g(M)$ for M , and similarly $g_0(\sigma)$ is the gain for the linear neurons that were introduced through transforming E_1 , namely σ, τ , and ω . Also $g_0(WTA)$ is the gain for the infinitely fast linear neurons which were used in both reduced and control experiments to implement the WTA constraints; this parameter effectively multiplies c_1 . *sweeps* is the number of iterations of the forward Euler method used in simulating the continuous update equations. Each iteration advanced the time coordinate by Δt .

There is only a little parameter-tuning involved here, concentrated on r_σ , Δt , and *sweeps*. The product $\Delta t \times \max(r_\sigma, r_M = 1)$ should be held fixed to maintain constant resolution in the discrete simulation of continuous update equations. But holding Δt and the other parameters fixed at the quoted values, the rate parameter r_σ can be varied from unity to 100 without altering the network convergence time, measured in sweeps, by more than 30% or so; network performance remains the same in that the same 5 out of 6 graphs are correctly matched. This would suggest, and other experiments confirm, that for low r there is some room for increasing Δt and decreasing sweeps ($r = 10$, $\Delta t = .016$, and *sweeps* = 300 respectively); this saves time whether time is measured in simulation sweeps or in circuit time constants.

3.4 Sorting

Sorting may be described in a manner very similar to graph matching. One requires a permutation matrix M which sorts the inputs into increasing order, so all terms in the objective remain the same except for $GgMM$. The objective becomes

$$\begin{aligned}
 E_{\text{sort}} = & -c_1 \sum_{ij} M_{ij} x_i y_j \\
 & + c_2 \sum_i (\sum_i M_{ij} - 1)^2 + c_2 \sum_j (\sum_j M_{ij} - 1)^2 \\
 & + c_3 \sum_{ij} M_{ij} (1 - M_{ij}) \\
 & + \sum_{ij} \int^{M_{ij}} dx g^{-1}(x).
 \end{aligned} \tag{48}$$

Here x_i are a set of input numbers to be sorted, and y_j are a constant set of numbers in increasing order, e.g. $y_j = j$. M will become that permutation matrix which maximizes the inner product of x and y , i.e. maps the largest x_i to the largest y_j , the next largest x_i to the next largest y_j , and so on.

After using the winner-take-all reduction on the row and column constraints of M , this network has $\mathcal{O}(N^2)$ connections and neurons. One cannot do better without reducing the number of match neurons M . But M is sparse at any acceptable answer, so it may be possible to keep it sparse throughout the time the network is running by using a different encoding of the matrix. For example, one might encode indices i , j , or both in a binary representation as would be done in an ordinary computer program for sorting, in which one commonly uses the binary representation of pointers to represent a sparse graph $M_{ij} \in \{0, 1\}$; alternatively if M is always a permutation

one can represent it by a one-dimensional array of binary addresses $j[i]$. The resulting objectives generally still have $\mathcal{O}(N^2)$ connections (monomial interactions), but a well-chosen matrix encoding, supplemented by suitable reversed neurons, can drastically reduce the number of connections.

In Appendix B it is shown that any permutation matrix M of size $L^2 \times L^2$ can be represented in the following form:

$$M_{i_1 i_2, j_1 j_2} = \sum_{k_1 k_2} A_{i_1 i_2, k_1}^{(1)} A_{i_2, k_1 k_2}^{(2)} \tilde{A}_{j_1 j_2, k_1}^{(1)} \tilde{A}_{j_2, k_1 k_2}^{(2)} \quad (49)$$

where $i \in \{1, \dots, N = L^2\}$, $i_k \in \{1, \dots, \sqrt{N} = L\}$, and $i = i_1 L + i_2 \equiv (i_1, i_2)$, and where two constraints apply to each nonsquare matrix:

$$\begin{aligned} \sum_{i_1} A_{i_1 i_2, j_1}^{(1)} = 1 \quad \Bigg| \quad \sum_{i_2} A_{i_2, j_1 j_2}^{(2)} = 1 \quad \Bigg\| \quad \sum_{i_1} \tilde{A}_{i_1 i_2, j_1}^{(1)} = 1 \quad \Bigg| \quad \sum_{i_2} \tilde{A}_{i_2, j_1 j_2}^{(2)} = 1 \\ \sum_{j_1} A_{i_1 i_2, j_1}^{(1)} = 1 \quad \Bigg| \quad \sum_{j_2} A_{i_2, j_1 j_2}^{(2)} = 1 \quad \Bigg\| \quad \sum_{j_1} \tilde{A}_{i_1 i_2, j_1}^{(1)} = 1 \quad \Bigg| \quad \sum_{j_2} \tilde{A}_{i_2, j_1 j_2}^{(2)} = 1 \end{aligned} \quad (50)$$

The matrix form (49) contains only $4N^{3/2}$ variables, and is our proposed encoding of M .

As explained in Appendix C, equation (49) is a coarse version of another expression for M which contains $\mathcal{O}(N \log N)$ variables. That expression codes index pairs (i, j) using the ‘‘Butterfly’’ connection topology that arises in the Fast Fourier Transform (FFT) and in many other parallel algorithms. The advantage of the Butterfly is that it allows one to make a gradual transition from one space (e.g. index i) to another (index j). There has been little success in transforming objectives based on the much less gradual binary or base- b encoding of i and M_{ij} .² An example similar to equation (49) is the base \sqrt{N} code obtained by listing all N links in the permutation matrix, indexed by k , and encoding their starting and ending locations as $i = (i_1, i_2)$ and $j = (j_1, j_2)$. Then

$$M_{i_1 i_2, j_1 j_2} = \sum_k A_{i_1, k}^{(1)} A_{i_2, k}^{(2)} \tilde{A}_{j_1, k}^{(1)} \tilde{A}_{j_2, k}^{(2)} \quad (51)$$

subject to obvious constraints on the A ’s.

²A partial exception is the load-balancing network of Fox and Furmanky (Fox and Furmanky, 1988), in which the crucial ‘‘histogram’’ may be understood as a set of reversed linear interneurons which simplify their load-balancing objective. But the result is a virtual neural net, not a statically connected circuit.

Since any permutation matrix can be expressed using equation (49), any (approximately quadratic) local minimum of $E_{\text{sort}}(M)$ (equation (48)), for which M is sufficiently close to being a permutation matrix, should be a local minimum of $E_{\text{sort}}(A^{(1)}, A^{(2)}, \tilde{A}^{(1)}, \tilde{A}^{(2)})$; but there may be local minima with respect to A and \tilde{A} which would be unstable in the larger M space. Thus, making the substitution (49) into equation (48) may expand the set of fixed points, or alter it entirely if the original objective is not yet tuned to produce permutation matrices. By contrast, the objective function transformations used heretofore have exactly preserved the set of fixed points. We will try it out anyway, in order to get a low-cost net, and we will observe whether and how well it sorts.

The problem now is to reduce the number of monomial interactions to $\mathcal{O}(N^{3/2})$. This is easy for quadratic penalty terms corresponding to the constraints (50), which consist of $8N$ winner-take-all constraints each involving $N^{1/2}$ variables. The remaining interaction $-c \sum Mxy$ can be reduced in two stages: substitute $M = A\tilde{A}^T$ with no reduction in connection costs; then substitute the $\mathcal{O}(N^{3/2})$ forms for A and \tilde{A} .

Thus we may replace $E_1 = -c_1 \sum_{ij} M_{ij}x_iy_j$ with

$$\begin{aligned} E_1(A, \tilde{A}) &= -c_1 \sum_{ijk} A_{ik}\tilde{A}_{jk}x_iy_j \\ &= -\frac{c_1}{2} \left[\sum_k (\sum_i A_{ik}x_i + \sum_j \tilde{A}_{jk}y_j)^2 \right. \\ &\quad \left. - \sum_k (\sum_i A_{ik}x_i)^2 - \sum_k (\sum_j \tilde{A}_{jk}y_j)^2 \right] \end{aligned} \quad (52)$$

$$\begin{aligned} \hat{E}_1(A, \tilde{A}, a, b, \tilde{b}) &= -c_1 \left[\sum_k (a_k - b_k) \sum_i A_{ik}x_i + \sum_k (a_k - \tilde{b}_k) \sum_j \tilde{A}_{jk}y_j \right. \\ &\quad \left. - \frac{1}{2} \sum_k a_k^2 + \frac{1}{2} \sum_k b_k^2 + \frac{1}{2} \sum_k \tilde{b}_k^2 \right] \end{aligned}$$

which may be interpreted as four interacting sorting problems, with linear interneurons a interpolating between x and y and with reversed neurons b and \tilde{b} cancelling echos as in equation (15). So far there is no reduction in number of neurons or connections.

If we substitute the special forms for A and \tilde{A} , we find

$$\begin{aligned}
\hat{E}_1 &= -c_1 \left[\sum_{i_1 i_2} \sum_{k_1 k_2} x_i A_{i k_1}^{(1)} A_{i_2 k}^{(2)} (a_k - b_k) + \sum_{j_1 j_2} \sum_{k_1 k_2} y_j \tilde{A}_{j k_1}^{(1)} \tilde{A}_{j_2 k}^{(1)} (a_k - \tilde{b}_k) \right. \\
&\quad \left. - \frac{1}{2} \sum_k a_k^2 + \frac{1}{2} \sum_k b_k^2 + \frac{1}{2} \sum_k \tilde{b}_k^2 \right] \\
&= -c_1 \left[\frac{1}{2} \sum_{i_2 k_1} [(\sum_{i_1} A_{i_1 i_2, k_1}^{(1)} x_i + \sum_{k_2} A_{i_2, k_1 k_2}^{(2)} (a_k - b_k))^2 \right. \\
&\quad \left. - (\sum_{i_1} A_{i_1 i_2, k_1}^{(1)} x_i)^2 - (\sum_{k_2} A_{i_2, k_1 k_2}^{(2)} (a_k - b_k))^2] \right. \\
&\quad \left. + \frac{1}{2} \sum_{j_2 k_1} [(\sum_{j_1} \tilde{A}_{j_1 j_2, k_1}^{(1)} y_j + \sum_{k_2} \tilde{A}_{j_2, k_1 k_2}^{(2)} (a_k - \tilde{b}_k))^2 \right. \\
&\quad \left. - (\sum_{j_1} \tilde{A}_{j_1 j_2, k_1}^{(1)} y_j)^2 - (\sum_{k_2} \tilde{A}_{j_2, k_1 k_2}^{(2)} (a_k - \tilde{b}_k))^2] \right. \\
&\quad \left. - \frac{1}{2} \sum_k a_k^2 + \frac{1}{2} \sum_k b_k^2 + \frac{1}{2} \sum_k \tilde{b}_k^2 \right]
\end{aligned} \tag{53}$$

and finally

$$\begin{aligned}
\hat{\hat{E}}_1 &= -c_1 \left[\sum_{i k_1} A_{i i_2, k_1}^{(1)} x_i (\sigma_{i_2 k_1} - \tau_{i_2 k_1}) + \sum_{i_2 k} A_{i_2, k_1 k_2}^{(2)} (a_k - b_k) (\sigma_{i_2 k_1} - \omega_{i_2 k_1}) \right. \\
&\quad \left. - \frac{1}{2} \sum_{i_2 k_1} \sigma_{i_2 k_1}^2 + \frac{1}{2} \sum_{i_2 k_1} \tau_{i_2 k_1}^2 + \frac{1}{2} \sum_{i_2 k_1} \omega_{i_2 k_1}^2 \right. \\
&\quad \left. + \sum_{j k_1} \tilde{A}_{j_1 j_2, k_1}^{(1)} y_j (\tilde{\sigma}_{j_2 k_1} - \tilde{\tau}_{j_2 k_1}) + \sum_{j_2 k} \tilde{A}_{j_2, k_1 k_2}^{(2)} (a_k - \tilde{b}_k) (\tilde{\sigma}_{j_2 k_1} - \tilde{\omega}_{j_2 k_1}) \right. \\
&\quad \left. - \frac{1}{2} \sum_{j_2 k_1} \tilde{\sigma}_{j_2 k_1}^2 + \frac{1}{2} \sum_{j_2 k_1} \tilde{\tau}_{j_2 k_1}^2 + \frac{1}{2} \sum_{j_2 k_1} \tilde{\omega}_{j_2 k_1}^2 \right. \\
&\quad \left. - \frac{1}{2} \sum_k a_k^2 + \frac{1}{2} \sum_k b_k^2 + \frac{1}{2} \sum_k \tilde{b}_k^2 \right]
\end{aligned} \tag{54}$$

which has $\mathcal{O}(N^{3/2})$ neurons and connections.

In Appendix C we show how to extend this result to a series of successively cheaper approximations of the original sorting network, down to $\mathcal{O}(N \log N)$ neurons and connections.

3.5 Sorting: Experiments

The $\mathcal{O}(N^{3/2})$ sorting network only sorts in an approximate way. The reversed neurons work correctly at finite r , which is nontrivial since they are connected to each other, but the encoding scheme is prone to trapping by local minima. We used the following

parameter values for the $\mathcal{O}(N^{3/2})$ sorting network:

$$\begin{array}{cccc}
 N = 16 & c_1 = 0.6 & c_2 = 6.0 & c_3 = 0 \\
 g_0(A) = 20 & r_A = 1 & r_\sigma = 3 & \Delta t = .01 \\
 sweeps = 20\,000 & & & \\
 \hline
 N = 25 & c_1 = .44 & c_2 = 6.0 & c_3 = 0 \\
 g_0(A) = 20 & r_A = 1 & r_\sigma = 3 & \Delta t = .01 \\
 sweeps = 20\,000 & & &
 \end{array}$$

and for the $\mathcal{O}(N^2)$ network:

$$\begin{array}{cccc}
 N = 16 & c_1 = 0.6 & c_2 = 6.0 & c_3 = 0 \\
 g_0(A) = 20 & \Delta t = .01 & sweeps = 5\,000 & \\
 \hline
 N = 25 & c_1 = .44 & c_2 = 6.0 & c_3 = 0 \\
 g_0(A) = 20 & \Delta t = .01 & sweeps = 5\,000 &
 \end{array}$$

As in the graph-matching example, most of the parameter-tuning was concentrated on r_σ , Δt , and *sweeps*. Here, r_σ is the rate parameter appearing in the update equation (7), and it applies to neurons $a, b, \tilde{b}, \sigma, \tilde{\sigma}, \tau, \tilde{\tau}, \omega, \tilde{\omega}$. Likewise r_A applies to the update equations for A and \tilde{A} . As in equation (48), c_1 multiplies the strength of the permuted inner product of \vec{x} and \vec{y} in the objective. Also c_2 is the strength of the syntax constraints, and c_3 is the strength of a term penalizing intermediate neuron values. $g_0(A)$ is the gain $g'(0)$ of the transfer function $g(A)$ for A and \tilde{A} , which obeyed steepest-descent dynamics. The constant y values are $y_1 = -(N - 1)/2$, $y_2 = -(N - 3)/2, \dots, y_{N-1} = (N - 3)/2, y_N = (N - 1)/2$. *sweeps* is the number of iterations of the forward Euler method used in simulating the continuous update equations.

For input size $N = 16$ we find an average placement error of 1.4 out of 16 possible output places. Eight would be random. For $N = 25$, which is the first size (with integral \sqrt{N}) for which there are fewer neurons in the asymptotically smaller network, the average placement error is 1.7 out of 25 places. The errors can be characterized by a histogram showing the frequency with which placement errors of different sizes occur. (The size of a placement error is the difference, in the permuted output vector, between the desired and actual positions of an element.) Histograms for $N = 16$ and $N = 25$ are presented in Figure 5 and they show that small mistakes are far more likely than large ones, and that the frequency falls off as roughly the

Figure 5: **Fig 5 goes about here.**

-2.1 (respectively -2.0) power of the size of the placement error, with correlation $r = .90$ (.93). In addition, 17.7% (respectively 21%) of the experimental runs failed to meet our convergence criterion, which was that exactly one element in each row and column of the computed matrix M must have a value greater than .5.

Iterating the sort can improve the score marginally, but not to the perfect sorts achievable with the $\mathcal{O}(N^2)$ network at these sizes.

4 Discussion and Conclusion

Although much research now focuses on expressing new computational problems using objective functions and then deriving neural networks which solve them, we would like to suggest that when such efforts are successful there may be an additional advantage to be obtained. If the solution can be regarded as a novel algebraic transformation of an unremarkable objective, then the transformation may also be immediately applicable to other objectives and problems. This provides a kind of *reuse* of neural net design effort which is fundamentally more flexible than the reuse of modules or components as practiced in electronic design, (Heinbuch, 1988), and in neural net learning, e.g. (Mjolsness et al., 1989b). The transformational approach has also proved useful in VLSI design, e.g. transforming a program into a microprocessor (Martin et al., 1989).

We have shown that there are algebraic transformations of objective functions which not only preserve the set of fixpoints of the resulting analog neural network, but alter the number and nature of interactions requiring physical connections between different neurons. Some of these transformations require the network to find a saddle point rather than a minimum of the objective, but that can be arranged. Others provide control over the dynamics by which an objective is extremized. A set of such transformations was derived, together with their conditions of validity.

Several design examples were given, along with experimental results to show convergence with reasonable parameter values. Reduced-cost designs were presented for convolution and linear coordinate transformation. A reduced network for sorting

converged to approximately correct answers despite the use of an illegitimate transformation which introduced spurious fixpoints. This design also involved legitimate but repeated, interacting product-reduction transformations. Transformed networks for quadratic matching and for registration of random dot patterns were simulated without difficulty.

Acknowledgements

We wish to acknowledge discussions with P. Anandan, Christian Darken, Wojtek Furmansky, Gene Gindi, Drew McDermott, Willard Miranker, John Platt, Petar Simic, and Joachim Utans.

This work was supported in part by AFOSR grant AFOSR-88-0240.

Appendices

A Reducing $YF(X)$

The problem is to use equation 21 to reduce expressions of the form $YF(X)$. We will derive two versions: the first, from stopping after the first step in the derivation of (21); the second, from carrying the transformation all the way through.

Now

$$\begin{aligned}
 Y \int^X du f(u) &= \int^X du \left[\int^Y dv g(u, v) \right]^{-1} (u) \\
 \Rightarrow Y f(X) &= \left[\int^Y dv g(u, v) \right]^{-1} (X) \\
 \Rightarrow \int^Y dv g(X, v) &= f^{-1}(X/Y) \\
 \Rightarrow g(u, v) &= (d/dv)f^{-1}(u/v) = -(u/v^2)(f^{-1})'(u/v).
 \end{aligned} \tag{55}$$

To use the first step in the derivation of equation (21), we must evaluate

$$\int^\sigma du \int^Y dv g(u, v) = \int^\sigma du f^{-1}(u/v) = Y \int^{\sigma/Y} du f^{-1}(u)$$

so

$$Y \int^X f(u) du \rightarrow XY\sigma - Y \int^\sigma du f^{-1}(u) \tag{56}$$

since σ may be rescaled by Y . This shows that both sides of the transformation (16) may be multiplied by any expression Y .

To carry through transformation (21) we must calculate

$$\begin{aligned}
 \int^\tau dv \left[\int^\sigma du g(u, v) \right]^{-1} (v) &= \int^\tau dv \left[\int^\sigma du (-u/v^2) (f^{-1})'(u/v) \right]^{-1} (v) \\
 &= \int^\tau dv \left[- \int^{\sigma/v} du u (f^{-1})'(u) \right]^{-1} (v) \\
 &= \int^\tau dv \left[- \int^{f^{-1}(\sigma/v)} dz f(z) \right]^{-1} (v) \\
 &= \int^\tau dv \left[-F(f^{-1}(\sigma/v)) \right]^{-1} (v) \\
 &= \int^\tau dv \left[\sigma/f(F^{-1}(-v)) \right] = -\sigma \int^{F^{-1}(-\tau)} dw \\
 &= -\sigma F^{-1}(-\tau)
 \end{aligned} \tag{57}$$

from which we deduce that

$$YF(X) \rightarrow -X\sigma + Y\tau + \sigma F^{-1}(\tau). \tag{58}$$

(τ and σ have been rescaled by -1). The algebra can be checked by optimizing with respect to σ . Note that the derivations of (56) and (58) assume that $f = F'$ is invertible and that $f^{-1} = (F')^{-1}$ is differentiable (i.e. $((F')^{-1})'$ exists).

B Butterfly Networks

B.1 Back-to-back Butterflies

By a recursive induction argument (Benes, 1965), any permutation matrix of size $N = 2^n$ (n an integer) can be expressed by setting switches in two back-to-back butterfly networks independently, as shown in Figure 6. If we label the corresponding connection matrices A and \tilde{A} , then the entire network represents a permutation matrix M in the form of a matrix product $M_{ij} = (A\tilde{A}^T)_{ij} = \sum_k A_{ik}\tilde{A}_{jk}$, with A and \tilde{A} being of the special “butterfly” form. Butterfly networks are best analyzed by introducing binary notation for all indices, e.g.

$$\begin{aligned} i &\rightarrow (p_1, \dots, p_n) \sim p_1 \dots p_n, \\ j &\rightarrow (q_1, \dots, q_n) \sim q_1 \dots q_n. \end{aligned} \quad (59)$$

In this notation, the outer column of switches in the A butterfly has the form

$$B_{p_1 \dots p_n, q_1}^{(1)} \in [0, 1] \quad (60)$$

with constraints

$$\sum_{p_1} B_{p_1 \dots p_n, q_1}^{(1)} = 1 \text{ and } \sum_{q_1} B_{p_1 \dots p_n, q_1}^{(1)} = 1 \quad (61)$$

as illustrated by the “butterfly” (\bowtie) highlighted in Figure 6. Likewise the l 'th column of switches has the form

$$B_{p_l \dots p_n, q_1 \dots q_l}^{(l)} \in [0, 1] \text{ with } \sum_{p_l} B_{p_l \dots p_n, q_1 \dots q_l}^{(l)} = 1 = \sum_{q_l} B_{p_l \dots p_n, q_1 \dots q_l}^{(l)}, \quad (62)$$

for $1 \leq l \leq n$. With constraints, each of the $\log N$ layers contains $N/2$ bits, which is one reason that \tilde{A} is also needed to specify an entire permutation matrix.

The entire permutation matrix is obtained by finding all the possible paths through the network from i to j , of which there is only one since each stage irrevocably decides one bit of j . This is equivalent to a conjunction of switch settings:

$$A_{ij} = \prod_{l=1}^n B_{p_l \dots p_n, q_1 \dots q_l}^{(l)}. \quad (63)$$

It is easy to check that this is a permutation matrix, using the constraints on $A^{(l)}$. (In what follows the terms of a product \prod do not commute because they contain

Figure 6: **Fig 6 goes about here.**

summations \sum that extend over subsequent terms in the product).

$$\begin{aligned}
 \sum_j A_{ij} &= \sum_{q_1 \dots q_n} A_{i, q_1 \dots q_n} \\
 &= \prod_{l=1}^n \left(\sum_{q_l} B_{p_l \dots p_n, q_1 \dots q_l}^{(l)} \right) \\
 &= \left[\prod_{l=1}^{n-1} \left(\sum_{q_l} B_{p_l \dots p_n, q_1 \dots q_l}^{(l)} \right) \right] \left(\sum_{q_n} B_{p_n, q_1 \dots q_n}^{(n)} = 1 \right) \\
 &= \prod_{l=1}^{n-1} \left(\sum_{q_l} B_{p_l \dots p_n, q_1 \dots q_l}^{(l)} \right) \\
 &= 1, \text{ by induction on } n.
 \end{aligned} \tag{64}$$

Likewise

$$\begin{aligned}
 \sum_i A_{ij} &= \sum_{p_1 \dots p_n} A_{i, p_1 \dots p_n} \\
 &= \prod_{l=n}^1 \left(\sum_{p_l} B_{p_l \dots p_n, q_1 \dots q_l}^{(l)} \right) \\
 &= \left[\prod_{l=n}^2 \left(\sum_{p_l} B_{p_l \dots p_n, q_1 \dots q_l}^{(l)} \right) \right] \left(\sum_{p_1} B_{p_1 \dots p_n, q_1}^{(1)} = 1 \right) \\
 &= \prod_{l=n}^2 \left(\sum_{p_l} B_{p_l \dots p_n, q_1 \dots q_l}^{(l)} \right) \\
 &= 1, \text{ by induction.}
 \end{aligned} \tag{65}$$

B.2 Coarse Butterflies

A less restrictive form for A may be derived as follows. Let k be roughly $n/2$. Then

$$i_1 \equiv p_1 \dots p_k, \quad i_2 \equiv p_{k+1} \dots p_n, \quad j_1 \equiv q_1 \dots q_k, \quad j_2 \equiv q_{k+1} \dots q_n;$$

from equation (63),

$$\begin{aligned}
 A_{ij} &= \prod_{l=1}^n B_{p_l \dots p_n, q_1 \dots q_l}^{(l)} \\
 &= \left(\prod_{l=1}^k B_{p_l \dots p_n, q_1 \dots q_l}^{(l)} \right) \left(\prod_{l=k+1}^n B_{p_l \dots p_n, q_1 \dots q_l}^{(l)} \right) \\
 &= \left(\prod_{l=1}^k B_{p_l \dots p_n, q_1 \dots q_k}^{(l)} \right) \left(\prod_{l=k+1}^n B_{p_{k+1} \dots p_n, q_1 \dots q_n}^{(l)} \right) \\
 &= A_{i_1 i_2, j_1}^{(1)} A_{i_2, j_1 j_2}^{(2)}
 \end{aligned} \tag{66}$$

and as in equation (64) one can derive the constraints

$$\begin{aligned}
 \sum_{j_1} A_{i_1 i_2, j_1}^{(1)} &= \prod_{l=1}^k \left(\sum_{q_l} B_{p_l \dots p_n, q_1 \dots q_l}^{(l)} \right) \\
 &= \left[\prod_{l=1}^{k-1} \left(\sum_{q_l} B_{p_l \dots p_n, q_1 \dots q_l}^{(l)} \right) \right] \left(\sum_{q_k} B_{p_k \dots p_n, q_1 \dots q_k}^{(k)} = 1 \right) \\
 &= \prod_{l=1}^{k-1} \left(\sum_{q_l} B_{p_l \dots p_n, q_1 \dots q_l}^{(l)} \right) \\
 &= 1, \text{ by induction on } k.
 \end{aligned} \tag{67}$$

Likewise

$$\begin{aligned}
 \sum_{i_1} A_{i_1 i_2, j_1}^{(1)} &= \prod_{l=k}^1 \left(\sum_{p_l} B_{p_l \dots p_n, q_1 \dots q_l}^{(l)} \right) \\
 &= \left[\prod_{l=k}^2 \left(\sum_{p_l} B_{p_l \dots p_n, q_1 \dots q_l}^{(l)} \right) \right] \left(\sum_{p_1} B_{p_1 \dots p_n, q_1}^{(1)} = 1 \right) \\
 &= \prod_{l=k}^2 \left(\sum_{p_l} B_{p_l \dots p_n, q_1 \dots q_l}^{(l)} \right) \\
 &= 1, \text{ by induction.}
 \end{aligned} \tag{68}$$

The constraints on $A^{(2)}$ and \tilde{A} are similar. Thus the constraints on $B^{(l)}$ imply the less restrictive constraints on $A^{(1)}$ and $A^{(2)}$, which we then adopt as the only constraints operating on A .

This completes the proof that any permutation matrix M can be represented by $\mathcal{O}(N^{3/2})$ variables in the form of equation (49) with constraints as in equation (50).

C Full Butterfly Neural Nets

The arguments of Appendix B can be generalized to yield a series of reduced objectives interpolating between $\mathcal{O}(N^{3/2})$ and $\mathcal{O}(N \log N)$ variables, each having about the same number of connections as variables. In Appendix B the idea was to express each index i in base $\sqrt{N} = 2^{n/2=k}$, by dividing the indices $p_1 \dots p_n$ of i 's binary expansion in two groups. We may instead divide the n binary indices into m groups of size k , with $n = km$, deriving the base 2^k expansion $i = i_1 \dots i_m$. (We have dealt with the special cases $k = n$ and $k \approx n/2$.) Then

$$\begin{aligned}
 A_{ij} &= \prod_{l=1}^n B_{p_l \dots p_n, q_1 \dots q_l}^{(l)} \\
 &= \prod_{l=1}^m \left(\prod_{r=1}^k B_{p_{l,r} \dots p_n, q_1 \dots q_{l,r}}^{(l,r)} \right) \\
 &= \prod_{l=1}^m A_{i_l \dots i_m, j_1 \dots j_l}^{(l)}
 \end{aligned} \tag{69}$$

as in equation (66). The constraints on $A^{(l)}$ are, as usual,

$$\sum_{i_l} A_{i_l \dots i_m, j_1 \dots j_l}^{(l)} = 1 = \sum_{j_l} A_{i_l \dots i_m, j_1 \dots j_l}^{(l)} \tag{70}$$

which are generally less restrictive than the original constraints on the butterfly switches B .

From equation (52), our problem is to reduce

$$E_1(A) = -c_1 \sum_{ij} A_{ij} x_i e_j \quad (71)$$

(where e_j is any expression) upon substituting equation (69).

If we take $e_j = e_{j_1 \dots j_m}^{(m)}$, $A_{ij} = C_{i_1 \dots i_m, j_1 \dots j_m}^{(m)}$, and $E_1(A) = E_1^{(m)}(C^{(m)})$, then we may use induction on a to reduce

$$E_1^{(a)}(C^{(a)}) = -c_1 \sum_{\substack{i_1 \dots i_m \\ j_1 \dots j_a}} C_{i_1 \dots i_m, j_1 \dots j_a}^{(a)} x_{i_1 \dots i_m} e_{i_{a+1} \dots i_m, j_1 \dots j_a}^{(a)} \quad (a \geq 2), \quad (72)$$

where

$$C_{i_1 \dots i_m, j_1 \dots j_a}^{(a)} \equiv \prod_{l=1}^a A_{i_1 \dots i_m, j_1 \dots j_l}^{(l)}. \quad (73)$$

Then

$$C_{i_1 \dots i_m, j_1 \dots j_a}^{(a)} = C_{i_1 \dots i_m, j_1 \dots j_{a-1}}^{(a-1)} A_{i_a \dots i_m, j_1 \dots j_a}^{(a)}$$

whence

$$\begin{aligned} E_1^{(a-1)} &= -\frac{c_1}{2} \sum_{\substack{i_a \dots i_m \\ j_1 \dots j_{a-1}}} \left[\left(\sum_{i_1 \dots i_{a-1}} C_{i_1 \dots i_m, j_1 \dots j_{a-1}}^{(a-1)} x_{i_1 \dots i_m} \right. \right. \\ &\quad \left. \left. + \sum_{j_a} A_{i_a \dots i_m, j_1 \dots j_a}^{(a)} e_{i_{a+1} \dots i_m, j_1 \dots j_a}^{(a)} \right)^2 \right. \\ &\quad \left. - \left(\sum_{i_1 \dots i_{a-1}} C_{i_1 \dots i_m, j_1 \dots j_{a-1}}^{(a-1)} x_{i_1 \dots i_m} \right)^2 - \left(\sum_{j_a} A_{i_a \dots i_m, j_1 \dots j_a}^{(a)} e_{i_{a+1} \dots i_m, j_1 \dots j_a}^{(a)} \right)^2 \right] \\ &\rightarrow -c_1 \left[\sum_{i_1 \dots i_m} \sum_{j_1 \dots j_{a-1}} C_{i_1 \dots i_m, j_1 \dots j_{a-1}}^{(a-1)} x_{i_1 \dots i_m} \left(\sigma_{i_a \dots i_m, j_1 \dots j_{a-1}}^{(a-1)} - \tau_{i_a \dots i_m, j_1 \dots j_{a-1}}^{(a-1)} \right) \right. \\ &\quad \left. \sum_{i_a \dots i_m} \sum_{j_1 \dots j_a} A_{i_a \dots i_m, j_1 \dots j_a}^{(a)} e_{i_{a+1} \dots i_m, j_1 \dots j_a}^{(a)} \left(\sigma_{i_a \dots i_m, j_1 \dots j_{a-1}}^{(a-1)} - \omega_{i_a \dots i_m, j_1 \dots j_{a-1}}^{(a-1)} \right) \right. \\ &\quad \left. + \frac{1}{2} \sum_{i_a \dots i_m} \sum_{j_1 \dots j_{a-1}} \left[- \left(\sigma_{i_a \dots i_m, j_1 \dots j_{a-1}}^{(a-1)} \right)^2 + \left(\tau_{i_a \dots i_m, j_1 \dots j_{a-1}}^{(a-1)} \right)^2 + \left(\omega_{i_a \dots i_m, j_1 \dots j_{a-1}}^{(a-1)} \right)^2 \right] \right]. \quad (74) \end{aligned}$$

Note that, upon identifying

$$\sigma^{(a-1)} - \tau^{(a-1)} = \epsilon^{(a-1)},$$

we have an induction step $a \rightarrow a - 1, \forall a \geq 2$, which decreases the number of neurons and connections in the network. By induction on a , one may reduce equation (52) to:

$$\begin{aligned}
 E_{\text{sort}} = & -c_1 \left[\sum_{i_1 \dots i_m, j_1} A_{i_1 \dots i_m, j_1}^{(1)} x_{i_1 \dots i_m} \left(\sigma_{i_2 \dots i_m, j_1}^{(1)} - \tau_{i_2 \dots i_m, j_1}^{(1)} \right) \right. \\
 & + \sum_{a=2}^{m-1} \sum_{\substack{i_a \dots i_m \\ j_1 \dots j_a}} A_{i_a \dots i_m, j_1 \dots j_a}^{(a)} \left(\sigma_{i_a \dots i_m, j_1 \dots j_a}^{(a-1)} - \omega_{i_a \dots i_m, j_1 \dots j_{a-1}}^{(a-1)} \right) \\
 & \quad \times \left(\sigma_{i_{a+1} \dots i_m, j_1 \dots j_a}^{(a)} - \tau_{i_{a+1} \dots i_m, j_1 \dots j_{a-1}}^{(a)} \right) \\
 & + \sum_{i_m, j_1 \dots j_m} A_{i_m, j_1 \dots j_m}^{(m)} \left(\sigma_{i_m, j_1 \dots j_{m-1}}^{(m-1)} - \omega_{i_m, j_1 \dots j_{m-1}}^{(m-1)} \right) (a_{j_1 \dots j_m} - b_{j_1 \dots j_m}) \\
 & + \frac{1}{2} \sum_{a=1}^{m-1} \sum_{\substack{i_{a+1} \dots i_m \\ j_1 \dots j_a}} \left[- \left(\sigma_{i_{a+1} \dots i_m, j_1 \dots j_a}^{(a)} \right)^2 \right. \\
 & \quad \left. + \left(\tau_{i_{a+1} \dots i_m, j_1 \dots j_a}^{(a)} \right)^2 + \left(\omega_{i_{a+1} \dots i_m, j_1 \dots j_a}^{(a)} \right)^2 \right] \\
 & + \left\{ \begin{array}{l} x \rightarrow y \quad \sigma \rightarrow \tilde{\sigma} \\ \text{same, with } B \rightarrow \tilde{B} \quad \tau \rightarrow \tilde{\tau} \\ b \rightarrow \tilde{b} \quad \omega \rightarrow \tilde{\omega} \\ a \rightarrow a \end{array} \right\} \\
 & \left. + \frac{1}{2} \sum_{i_1 \dots i_m} \left[-a_{i_1 \dots i_m}^2 + b_{i_1 \dots i_m}^2 + \tilde{b}_{i_1 \dots i_m}^2 \right] \right]. \tag{75}
 \end{aligned}$$

The number of variables and connections for this objective is $\mathcal{O}(mN^{1+1/m})$, $1 \leq m \leq n$, which takes on values $N^2, 2N^{3/2}, 3N^{4/3}, \dots, N \log N$.

References

- Arrow, K. J., Hurwicz, L., and Uzawa, H., editors (1958). *Studies in Linear and Nonlinear Programming*. Stanford University Press.
- Benes, V. E. (1965). *Mathematical Theory of Connecting Networks and Telephone Traffic*. Academic Press. p. 113.
- Courant, R. and Hilbert, D. (1962). *Methods of Mathematical Physics*, volume II, pages 32–39. John Wiley and Sons.
- Durbin, R. and Willshaw, D. J. (1987). An analogue approach to the travelling salesman problem using an elastic net method. *Nature*, 326:689–691.
- Feldman, J. A. (1982). Dynamic connections in neural networks. *Biological Cybernetics*, 46:27–39.
- Fox, G. C. and Furmanský, W. (1988). Load balancing loosely synchronous problems with a neural network. Technical Report *C³P363B*, California Institute of Technology.
- Gindi, G., Gmitro, A., and Parthasarathy, K. (1987). Winner-take-all networks and associative memory: Analysis and optical realization. In *Proc. of First International Conference on Neural Networks*, volume vol. III, pages 607–614. IEEE.
- Grossberg, S. (1988). Nonlinear neural networks: Principles, mechanisms, and architectures. *Neural Networks*, 1:17–61.
- Heinbuch, D. V., editor (1988). *CMOS3 Cell Library*. Addison-Wesley.
- Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences USA*, vol. 81:3088–3092.
- Hopfield, J. J. and Tank, D. W. (1985). ‘Neural’ computation of decisions in optimization problems. *Biological Cybernetics*, vol. 52:141–152.
- Hopfield, J. J. and Tank, D. W. (1986). Collective computation with continuous variables. In *Disordered Systems and Biological Organization*, pages 155–170. Springer-Verlag.

- Koch, C., Marroquin, J., and Yuille, A. (1986). Analog “neuronal” networks in early vision. *Proceedings of the National Academy of Sciences USA*, 83.
- Luenberger, D. G. (1984). *Linear and Nonlinear Programming*. Addison-Wesley.
- Martin, A. J., Burns, S. M., Lee, T. K., Borkovic, D., and Hazewindus, P. J. (1989). The design of an asynchronous microprocessor. In *Proc. Decennial Caltech Conference on VLSI*, pages 351–373. MIT press.
- Mjolsness, E. (1987). Control of attention in neural networks. In *Proc. of First International Conference on Neural Networks*, volume vol. II, pages 567–574. IEEE.
- Mjolsness, E., Gindi, G., and Anandan, P. (1989a). Optimization in model matching and perceptual organization. *Neural Computation*, 1.
- Mjolsness, E. and Miranker, W. (1990). manuscript in preparation.
- Mjolsness, E., Sharp, D. H., and Alpert, B. K. (1989b). Scaling, machine learning, and genetic neural nets. *Advances in Applied Mathematics*, 10:137–163.
- Moody, J. (1989). Optimal architectures and objective functions for associative memory. Technical Report TR668, Yale University Department of Computer Science.
- Platt, J. C. and Barr, A. H. (1987). Constrained differential optimization. In Anderson, D. Z., editor, *Neural Information Processing Systems*. American Institute of Physics.
- Platt, J. C. and Barr, A. H. (1988). Constraint methods for flexible models. *Computer Graphics*, 22(4). Proceedings of SIGGRAPH '88.
- Rumelhart, D. E., Hinton, G. E., and McClelland, J. L. (1986a). A general framework for parallel distributed processing. In *Parallel Distributed Processing*, pages 73–74. MIT Press.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986b). Learning internal representations by error propagation. In *Parallel Distributed Processing*, pages 318–362. MIT Press.
- Simic, P. (1989). Personal communication.

- Sivilotti, M. A., Mahowald, M. A., and Mead, C. A. (1987). Real-time visual computations using analog CMOS processing arrays. In *Advanced Research in VLSI: Proceedings of the 1987 Stanford Conference*. MIT Press.
- Tank, D. W. and Hopfield, J. J. (1986). Simple ‘neural’ optimization networks: An a/d converter, signal decision circuit, and a linear programming circuit. *IEEE Transactions on Circuits and Systems*, CAS-33.
- Utans, J., Gindi, G., Mjolsness, E., and Anandan, P. (1989). Neural networks for object recognition within compositional hierarchies: Initial experiments. Technical Report Center for Systems Science Report No. 8903, Yale University Department of Electrical Engineering.
- von der Malsburg, C. and Bienenstock, E. (1986). Statistical coding and short-term synaptic plasticity: A scheme for knowledge representation in the brain. In *Disordered Systems and Biological Organization*, pages 247–252. Springer-Verlag.
- von Neumann, J. and Morgenstern, O. (1953). *Theory of Games and Economic Behavior*. Princeton University Press.

Figure Captions

Figure 1: **Order reduction.** Arbitrary fourth to third order reduction, using linear interneurons. Open circles: original neurons. Open squares: ordinary linear interneurons. Closed squares: reversed linear interneurons. Dots: connections, with strengths as indicated. Equilibrium values of the interneurons are indicated. After the transformation, neuron w receives input $xyz + z^2w$ from the left and a compensating input of $-z^2w$ from the right.

Figure 2: **Exponential and logarithmic neurons.** The minimum of $e^x + e^{-x}$ occurs at the saddle point of $e^x - x\sigma - \sigma \log \sigma + \sigma$, whose contours are plotted here. Also various two-neuron trajectories to the saddle point are shown, in which x or σ moves more slowly than the fastest implementable time scale, assumed to be $r = 1$. (a) $r_x = 1, r_\sigma = .1$. (b) $r_x = 1, r_\sigma = .3$. (c) $r_x = 1, r_\sigma = 1$. (d) $r_x = .3, r_\sigma = 1$. (e) $r_x = .1, r_\sigma = 1$. Dots occur every 10 time constants, so (c) gives quickest convergence.

Figure 3: **Random dot matching network.** Trajectory of network evolution equations (44) projected to the $\vec{\Delta}$ plane, superposed on a contour plot of E_{dots} . K was $.2$. The starting point was obtained by a partial relaxation of the same net for $K = .5$, for which the objective has a single local minimum, starting from $\vec{\Delta} = (1, 1)$ and $\vec{\omega} = \vec{0}$. (Further relaxation at $K = .5$ would result in an initial $\vec{\Delta}$ even closer to the $K = .2$ minimum.) The correct answer is $\vec{\Delta} = \vec{0}$.

Figure 4: **Graph matching networks.** E_1 is a sum over the indices α, β, i , and j , which are connected by neurons (line segments) in the shape of a “rectangle”. This objective can be transformed into \hat{E} , which is a sum of triangles, while preserving fixpoints. The triangles are obtained from the rectangle by introducing linear interneurons along a diagonal, as shown. Only three indices are summed over, resulting in a less costly network.

Figure 5: **Histogram of placement errors.** Sorting network with butterfly encoding, $\mathcal{O}(N^{3/2})$ connections. (a) Size $N = 16$. Average and standard deviation (upper or lower half of an error bar) for 62 runs. (b) Size $N = 25$. Average and

standard deviation for 39 runs.

Figure 6: **Butterfly switching networks.** Any permutation of $N = 2^n$ elements can be represented by appropriately setting the switches in a pair of back-to-back butterfly switching networks, as can be shown recursively by induction on n . Highlighted: one 2×2 permutation matrix or “butterfly”, and one path through the the entire switching network.