An optimizing network architecture that works

Anand Rangarajan
Steven Gold
Eric Mjolsness

Research Report YALEU/DCS/RR-1036
May 1994

An optimizing network architecture that works[1]

Anand Rangarajan,[2] Steven Gold[2] and Eric Mjolsness[2]
Department of Computer Science
Yale University
51 Prospect Street
New Haven, CT 06520-8285

Abstract

We have achieved considerable success with an optimizing network architecture for problems in vision, learning, pattern recognition and combinatorial optimization. This architecture is constructed by bringing together a number of different techniques including (i) deterministic annealing, (ii) self-amplification (iii) algebraic transformations, (iv) clocked objectives and (v) projection methods.

1 Introduction

Optimizing networks have been an important part of neural computation since the seminal work of Hopfield and Tank [4]. The attractive features of these networks—intrinsic parallelism, continuous descent inside a hypercube, ease in programming and mapping onto analog VLSI—raised tremendous hopes of finding good solutions to many "hard" combinatorial optimization problems. The results (for both speed and accuracy) have been mixed. This can be attributed to a number of factors, viz., slow convergence of gradient descent algorithms, inadequate problem mappings, poor constraint satisfaction etc.

In contrast, we have achieved considerable success with an optimizing network architecture for problems in vision, pattern recognition, unsupervised learning and combinatorial optimization. These problems are 2-D and 3-D pose estimation with

[1]Keywords: Algorithms and Architectures; Oral presentation preferred.
[2]E-mail address of authors: lastname-firstname@cs.yale.edu

unknown correspondence (in vision), weighted graph matching (in pattern recognition), clustering with point and graph matching distance measures (in unsupervised learning) and the traveling salesman problem (in combinatorial optimization). We have designed a network architecture that emphasizes speed and accuracy by using a combination of optimization techniques including deterministic annealing, self-amplification, algebraic transformations, Expectation-Maximization (EM) like clocked objectives and projection methods.

2 Deriving the network architecture

We now describe the techniques used in deriving the network architecture. This derivation is carried out with inexact graph matching as an example. The same overall network architecture is subsequently used in all applications.

The weighted, inexact graph matching problem can be stated as follows: Given the adjacency matrices of the two graphs $G(V, E)$ and $g(v, e)$,

$$\min_M E_{\text{gm}}(M) = \sum_{a=1}^{M} \sum_{i=1}^{N} \left(\sum_{b=1}^{M} G_{ab} M_{bi} - \sum_{j=1}^{N} M_{aj} g_{ji} \right)^2 - \alpha \sum_{a=1}^{M} \sum_{i=1}^{N} M_{ai}, \; \alpha > 0, \quad (1)$$

$$\text{subject to} \sum_{a=1}^{M} M_{ai} \le 1, \; \forall i, \; \sum_{i=1}^{N} M_{ai} \le 1, \; \forall a, \; M_{ai} \in \{0, 1\}. \quad (2)$$

When the constraints are equalities rather than inequalities and weights in $\{0, 1\}$, this reduces to the graph isomorphism problem.

2.1 Deterministic annealing

The integral constraints on M are relaxed via deterministic annealing. It can be shown [11] that deterministic annealing corresponds to adding a certain barrier function to the original objective (1). The barrier is indexed by a control parameter β which is identified with the inverse temperature in an annealing process. The annealing is completely deterministic.

$$\begin{aligned} E_{\text{da}}(M, \mu, \nu) &= \frac{1}{\beta} \sum_{a=1}^{M+1} \sum_{i=1}^{N+1} M_{ai} (\log(M_{ai}) - 1) \quad (3) \\ &+ \sum_{a=1}^{M} \mu_a \left(\sum_{i=1}^{N+1} M_{ai} - 1 \right) + \sum_{i=1}^{N} \nu_i \left(\sum_{a=1}^{M+1} M_{ai} - 1 \right). \end{aligned}$$

The inequality constraints are enforced via Lagrange parameters μ and ν and slack variables. The slacks are incorporated in M as an extra row and column which do not necessarily sum to zero or one. The barrier function keeps M non-negative. An annealing schedule is prescribed for β (low to high values). This is similar to the gradual change of a barrier function control parameter. Deterministic annealing ensures gradual progress towards a vertex of the hypercube.

2.2 Self-amplification

Deterministic annealing by itself cannot guarantee that the network will converge to a valid solution. However, self-amplification [10] in conjunction with annealing will converge for the constraints in (2) [11]. A popular choice for self-amplification is [7, 8, 1]

$$E_{\text{sa}}(M) = -\frac{\gamma}{2}\sum_{a=1}^{M+1}\sum_{i=1}^{N+1} M_{ai}^2, \; \gamma > 0. \tag{4}$$

Self-amplification in conjunction with annealing ensures that a vertex of the hypercube is found.

2.3 Algebraic transformations

An algebraic transformation [6] transforms a minimization problem into a saddle-point problem. The advantage of this operation is two-fold: (i) it cuts network costs in terms of connections and (ii) the transformed objectives are easier to extremize. Consider the following transformations:

$$\frac{X^2}{2} \rightarrow \lambda X - \frac{\lambda^2}{2} \tag{5}$$

$$X\log(X) - X \rightarrow XU - \exp(U) \tag{6}$$

where λ and U are reversed "neurons" [6]. Using these transformations, we transform the graph matching objective to

$$\begin{aligned} E_{\text{GM}}(M, U, \lambda, \mu, \nu) = & \sum_{a=1}^{M}\sum_{i=1}^{N}\left[\lambda_{ai}\left(\sum_{b=1}^{M} G_{ab}M_{bi} - \sum_{j=1}^{N} M_{aj}g_{ji}\right) - \frac{1}{2}\lambda_{ai}^2\right] \\ & -\alpha\sum_{a=1}^{M}\sum_{i=1}^{N} M_{ai} + \frac{1}{\beta}\sum_{a=1}^{M+1}\sum_{i=1}^{N+1}\left[U_{ai}M_{ai} - \exp(U_{ai}) - \frac{\gamma}{2}M_{ai}^2\right] \\ & + \sum_{a=1}^{M}\mu_a\left(\sum_{i=1}^{N+1} M_{ai} - 1\right) + \sum_{i=1}^{N}\nu_i\left(\sum_{a=1}^{M+1} M_{ai} - 1\right). \end{aligned} \tag{7}$$

The objective becomes linear in M following the application of the transformations in (5) and (6) to the graph matching and barrier function terms respectively. This has come at the expense of finding a saddle-point (minimization w.r.t M and maximization w.r.t λ, U, μ and ν). With the application of the algebraic transformations, the relaxation is partitioned into four phases; λ, U, M and (μ, ν).

2.4 Clocked objectives

After expressing the constraints and performing the algebraic transformations, the control structure for performing the network dynamics is specified by a clocked objective function [2]:

$$E_{\oplus} = E_{\text{GM}}\left\langle \lambda^*, U^*, M^*, \left\langle (\mu, M)^*, (\nu, M)^* \right\rangle_{\oplus} \right\rangle_{\oplus} \tag{8}$$

where $E_{\mathrm{GM}}\langle\cdot\rangle_{\oplus}$ is a clocked phase (iterated when necessary) and $(\cdot)^*$ indicates an analytic solution within a phase.

With this notation, the clocked objective states that closed-form solutions of λ, U and M are used by the network.

$$\begin{aligned} \lambda^*_{ai} &= \sum_b G_{ab}M_{bi} - \sum_j M_{aj}g_{ji}, && (9) \\ U^*_{ai} &= -\beta\left(\sum_b G_{ba}\lambda_{bi} - \sum_j \lambda_{aj}g_{ij} - \gamma M_{ai}\right), && (10) \\ M^*_{ai} &= \exp\left(U_{ai} - \beta\mu_a - \beta\nu_i\right). && (11) \end{aligned}$$

The network dynamics for the graph matching problem proceeds by coordinate-wise extremization on λ, U, M and further relaxation on the constraints (μ and ν). Clocked objectives allow analytic solutions within each phase, highly reminiscent of the EM algorithm [5].

2.5 Projection methods

The clocked objective includes a phase where we have to solve for the Lagrange parameters μ and ν. However, it can be shown [11] that this can be replaced by a projection onto the constraints. Note that the constraints are not as complex as the general integer programming constraint [1]. Projection corresponds to iterative row and column normalization:

$$M_{ai} \leftarrow \frac{M_{ai}}{\sum_{a=1}^{M+1} M_{ai}},\ i \in \{1, N\}\ ;\ M_{ai} \leftarrow \frac{M_{ai}}{\sum_{i=1}^{N+1} M_{ai}},\ a \in \{1, M\}. \qquad (12)$$

The projection includes the slacks as well and is not restricted to a square matrix, unlike [11].

To summarize, deterministic annealing creates a sequence of objective functions which approaches the original objective function (as β is increased). Self-amplification in conjunction with annealing ensures that a vertex of the hypercube is reached. Algebraic transformations in conjunction with clocked objectives help partition the relaxation into separate phases within which analytic solutions can be found. Constraints are solved by projection and in the case of graph matching (and quadratic assignment in general), the iterative row-column projection method is used. With the clocked objectives, analytic solutions and the projection method in place, we have our network architecture.

3 Problem examples

We now apply the same methodology used in deriving the inexact graph matching network to several problems in vision, learning, pattern recognition and combinatorial optimization. None of the networks used penalty functions, gradient descent with line search parameters or general constraint solvers. In all experiments, Silicon Graphics workstations with R4000 and R4400 processors were used.

3.1 Point matching

The point matching problem arises in the field of computer vision as pose estimation with unknown correspondence [2]. The problem is:

$$\min_{M,A} E_{\text{pm}}(M,A) = \sum_{ai} M_{ai}\|x_i - Ay_a\|^2 + g(A) - \alpha \sum_{ai} M_{ai} \tag{13}$$

$$\text{subject to } \sum_a M_{ai} \leq 1, \text{ and, } \sum_i M_{ai} \leq 1$$

where x and y are two point sets (2-D or 3-D) and A is a set of analog variables (rotation, translation, scale, shear). Since the objective is linear in M, no algebraic transformations are necessary. With M held fixed it is possible to solve exactly for A. When A is held fixed, we project on the constraints. In our experiments, A is an affine transformation which is decomposed into rotation, translation, scale and vertical and oblique shear. Each point matching instance takes between 15 to 50 seconds depending on speed/accuracy tradeoffs.

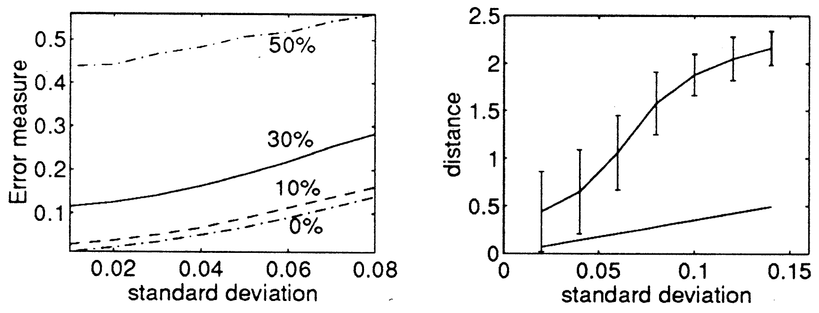

Figure 1: Left: (a) Affine point matching of 50 point 2-D point sets with 0, 10, 30 and 50% deleted points and 10% spurious points. Right: (b) Clustering with an affine point matching distance.

To test affine point matching, 2-D point sets with 50 points were generated by sampling from a uniform distribution in the unit square. The point sets were transformed via an affine transformation followed by additive Gaussian noise with varying standard deviations. Then a fraction of the points are deleted and another fraction added. We generated 500 test instances at each standard deviation. The results are shown in Figure 1(a). The four curves show the performance for 0, 10, 30 and 50% deleted and 10% spurious points. The error is computed for all affine transformation parameters and is the ratio of the absolute error to the interval in which the parameters lie. Note that there is virtually no difference between 0 and 10% deleted points.

3.2 Clustering with smart distances

We now embed the fast 2-D affine point matching method in a clustering objective. The goal is to classify shapes that are distorted by an affine transformation and have missing and extra features with respect to each other. We have described this in an earlier work as clustering with a domain-specific distance [2] where the aim was to use the natural distortion measure (point matching in this case) rather than a standardized, invariant distance measure. The domain specific clustering problem is formulated as

$$\min_{M,y,m,A} E_{\text{cl}}(M,y,m,A) = \sum_{ai} M_{ai} D_{ai}(x,y,m,A) \tag{14}$$

$$\text{subject to } \sum_{a} M_{ai} = 1, \text{ and constraints on } m, A$$

where $D_{ai}(x,y,m,A)$ is the point matching distance measure of (13). Here y are the point set cluster centers and M_{ai} the cluster membership matrix [2]. The resulting network is similar but more elaborate than the one reported in [2]. The distance measure allows for shear and scaling distortions and missing and extra features which were absent in [2]. The network consists of an outer phase (clustering) and an inner phase (point matching). Both phases are extremized with EM like dynamics. The objective has nearly a million variables and takes about 4 hours to optimize.

We generated 8 prototype point sets with 20 points each and a total of 256 point sets from all prototypes using the entire range of transformations (affine, permutation, missing and extra features, additive noise). We report the results in Figure 1(b). At each standard deviation, 10 trials were run. The results are good at low standard deviations and the bias in the results gets larger as the standard deviation is increased, as expected.

3.3 Graph matching

We have already examined the graph matching problem and constructed its optimizing network. Undirected 50 node graphs were generated with weights uniformly distributed in [0, 1]. The distorted graph was generated by randomly permuting the nodes and adding uniform noise to the links. Then, a fraction of the nodes were deleted. We generated 100 test instances at each standard deviation. Figure 2(a) summarizes our inexact, weighted graph matching results. The three curves show the performance for 2, 10 and 20% deleted nodes As expected, the performance degrades as more nodes are deleted and more noise is added. At low numbers of deleted nodes and moderate standard deviations, the network performs well. Figure 2(b) depicts exact graph matching for 100 and 50 nodes with 200 and 1000 test instances respectively at each standard deviation. The 50 node and 100 node exact graph matching optimization take about 10 seconds and 80 seconds respectively.

Elsewhere, we have reported closely related (more accurate but slower) Lagrangian relaxation networks for 100 node graph isomorphism and matching [8] and these compare favorably with other neural net graph matching networks in the literature [9].

3.4 TSP

We formulate the TSP problem in a conventional manner [4, 7].

$$\min_{M} E_{\text{tsp}}(M) = \sum_{aij} M_{ai} M_{(a\oplus 1)j} d_{ij} \tag{15}$$

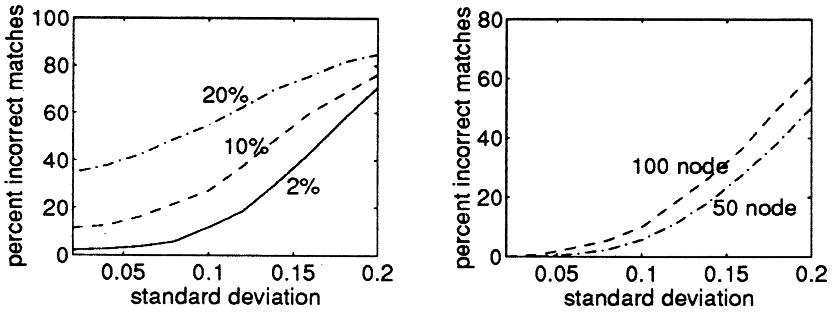

Figure 2: Left: (a) Inexact 50 node graph matching with 2%, 10% and 20% deleted nodes. Right: (b) Exact 50 and 100 node graph matching

$$\text{subject to } \sum_a M_{ai} = 1, \forall\, i, \ \sum_i M_{ai} = 1, \ \forall\, a, \ \text{and } M_{ai} \in \{0,1\} \qquad (16)$$

where d_{ij} is the city–city distance. Poor constraint satisfaction of (16) is one of the reasons for the poor performance of the original Hopfield network. This was partially corrected in [7] by keeping one of the constraints (row or column) always satisfied. By using the row-column projection method, we ensure that the constraints are always satisfied via a separate phase of optimization. The algebraic transformation in (6) is used for the barrier function and the resulting network is very similar to the one reported in [7] with the crucial difference being the row-column projection method for the constraints. A typical 100 node TSP run takes about 3 minutes.

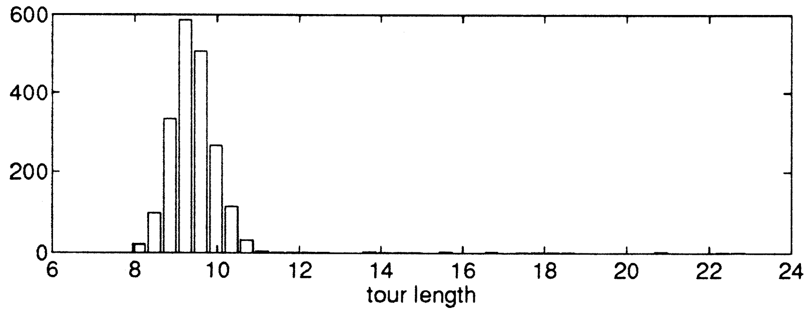

Figure 3: Histogram plot of tour lengths in the 100 city Euclidean TSP problem

We ran 2000 100-city TSPs with points uniformly generated in the 2-D unit square. The asymptotic expected length of an optimal tour for cities distributed in the unit square is given by $L(n) = K\sqrt{n}$ where n is the number of cities and $0.765 \leq K \leq 0.765 + \frac{4}{n}$ [3]. This gives us the interval $[7.65, 8.05]$ for the 100 city TSP. A histogram of the tour lengths is displayed in Figure 3. From the histogram, we observe that

98% of the tour lengths fall in the interval [8, 11]. No heuristics were used in either pre- or post-processing. These results are preliminary.

4 Conclusions

We have constructed an optimizing network architecture that performs well on a variety of problems in vision, learning, pattern recognition and combinatorial optimization. The resulting networks are fast and have been used in large scale (million variable) nonlinear optimization problems. This work renews the promise of optimizing networks.

Acknowledgements

This work is supported by AFOSR grant F49620-92-J-0465, ONR/ARPA grant N00014-92-J-4048 and the Yale Center for Theoretical and Applied Neuroscience. We acknowledge Chien-Ping Lu, Suguna Pappu, Manisha Ranade and Alan Yuille.

References

[1] A. H. Gee and R. W. Prager. Polyhedral combinatorics and neural networks. *Neural Computation*, 6(1):161–180, Jan. 1994.

[2] S. Gold, E. Mjolsness, and A. Rangarajan. Clustering with a domain specific distance measure. In J. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in NIPS 6*. Morgan Kaufmann, San Francisco, CA, 1994.

[3] B. L. Golden and W. R. Stewart. Empirical analysis of heuristics. In E. L. Lawler *et al.* editors, *The Traveling Salesman problem: A guided tour of combinatorial optimization*, pages 207–249. John Wiley and Sons, New York, 1985.

[4] J. J. Hopfield and D. Tank. "Neural" computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.

[5] M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6(2):181–214, March 1994.

[6] E. Mjolsness and C. Garrett. Algebraic transformations of objective functions. *Neural Networks*, 3:651–669, 1990.

[7] C. Peterson and B. Söderberg. A new method for mapping optimization problems onto neural networks. *Intl. Journal of Neural Systems*, 1(1):3–22, 1989.

[8] A. Rangarajan and E. Mjolsness. A Lagrangian relaxation network for graph matching. In *Proc. ICNN '94*. IEEE Press, 1994.

[9] P. D. Simić. Constrained nets for graph matching and other quadratic assignment problems. *Neural Computation*, 3:268–281, 1991.

[10] C. von der Malsburg. Network self-organization. In S. F. Zornetzer *et al.*, editors, *An Introduction to Neural and Electronic Networks*, pages 421–432. Acad. Press, San Diego, CA, 1990.

[11] A. L. Yuille and J. J. Kosowsky. Statistical physics algorithms that converge. *Neural Computation*, 6(3):341–356, May 1994.