An Improved Connectionist Activation Function for Energy Minimization

Gadi Pinkas
Dept. of Computer Science
Washington University
St. Louis, MO 63131

Rina Dechter
Information and Computer Science
University of California, Irvine, CA 92717

Technical Report 92–62

March, 1992

Appeared in *Proceedings of AAAI-92.*

Supported by NSF grant R-9008012 and by the Center for Intelligent Computer Systems at Washington University.

Partially supported by NSF grant IRI-9157636, the Air Force Office of Scientific Research, AFOSR 900136, GE Corporate R&D and by Toshiba of America.

An Improved Connectionist Activation Function for Energy Minimization

Gadi Pinkas*
Dept. of Computer Science
Washington University
St. Louis, MO 63131

Rina Dechter†
Dept. of Computer
and Information Science
University of California, Irvine

Abstract

Symmetric networks that are based on energy minimization, such as Boltzmann machines or Hopfield nets, are used extensively for optimization, constraint satisfaction, and approximation of NP-hard problems. Nevertheless, finding a global minimum for the energy function is not guaranteed, and even a local minimum may take an exponential number of steps. We propose an improvement to the standard activation function used for such networks. The improved algorithm guarantees that a global minimum is found in linear time for tree-like subnetworks. The algorithm is uniform and *does not* assume that the network is a tree. It performs no worse than the standard algorithms for any network topology. In the case where there are trees growing from a cyclic subnetwork, the new algorithm performs better than the standard algorithms by avoiding local minima along the trees and by optimizing the free energy of these trees in linear time. The algorithm is self-stabilizing for trees (cycle-free undirected graphs) and remains correct under various scheduling demons. However, no uniform protocol exists to optimize trees under a pure distributed demon and no such protocol exists for cyclic networks under central demon.

Introduction

Symmetric networks, such as Hopfield nets, and Boltzmann machines, mean-field and harmony networks are widely used for optimization, constraint satisfaction, and approximation of NP-hard problems [Hopfield 82], [Hopfield 84], [Hinton, Sejnowski 86], [Peterson, Hartman 89], [Smolensky 86, page 259].

These models are characterized by a symmetric matrix of weights and a quadratic energy function that should be minimized. Usually, each unit computes the gradient of the energy function and updates its own activation value so that the free energy decreases gradually. Convergence to a *local* minimum is guaranteed, although in the worst case it is exponential in the number of units [Kasif et al. 89], [Papadimitriou et al. 90].

In many cases, the problem at hand is formulated as an energy minimization problem and the best solutions (sometimes the *only* solutions) are the global minima [Hopfield, Tank 85],[Ballard et al. 86], [Pinkas 91]. The desired connectionist algorithm is, therefore, one that reduces the impact of shallow local minima and improves the chances of finding a global minimum. Models such as Boltzmann machines and Harmony nets use simulated annealing to escape from local minima. These models asymptotically converge to a global minimum, meaning that if the annealing schedule is slow enough, a global minimum is found. Nevertheless, such a schedule is hard to find and therefore, practically, finding a global minimum for such networks is not guaranteed even in exponential time (note that the problem is NP-hard).

In this paper, we look at the *topology* of symmetric neural networks. We present an algorithm that optimizes tree-like subnetworks[1] in linear time. It is based on a dynamic programming algorithm presented in [Dechter et al. 90]. Our adaptation is connectionist in style; that is, the algorithm can be stated as a simple, uniform activation function [Rumelhart et al. 86], [Feldman, Ballard 82]. It does not assume the desired topology (tree) and performs no worse than the standard algorithms for all topologies. In fact, it may be integrated with many of the standard algorithms in such a way that if the network happens to have tree-like subnetworks, the new algorithm out-performs the standard algorithms.

The paper is organized as follows: Section 2 dis-

*Supported by NSF grant R-9008012 and by the Center for Intelligent Computer Systems at Washington University.

†Partially supported by NSF grant IRI-9157636 and by the Air Force Office of Scientific Research, AFOSR 900136.

[1]The network is characterized by an undirected graph without cycles; i.e., only one path exists between any two nodes. The terms cycle-free or unrooted tree are synonymous in this context.

cusses connectionist energy minimization. Section 3 presents the new algorithm and gives an example where it out-performs the standard algorithms. Section 4 discusses convergence under various scheduling demons and self-stabilization. Section 5 summarizes.

Connectionist energy minimization

Suppose a quadratic energy function of the form

$$E(X_1, ..., X_n) = -\sum_{i<j}^{n} w_{i,j} X_i X_j + \sum_{i}^{n} -\theta_i X_i.$$

Each of the variables X_i may have a value of zero or one (called the activation value), and the task is to find a zero/one assignment to the variables $X_1, ..., X_n$ that minimizes the energy function. To avoid confusion with signs, we will consider the equivalent problem of maximizing the goodness function:

$$G(X_1, ..., X_n) = -E(X_1, ..., X_n) = \sum_{i<j} w_{i,j} X_i X_j + \sum_{i} \theta_i X_i \quad (1)$$

In connectionist approaches, we look at the network that is generated by assigning a node i for every variable X_i in the function and by creating a weighted arc (with weight $w_{i,j}$) between node i and node j for every term $w_{i,j} X_i X_j$. Similarly, a bias θ_i is given to unit i if the term $\theta_i X_i$ is in the function. For example, Figure 2-b shows the network that corresponds to the goodness function $E(X_1, ..., X_7) = 3X_2X_3 - X_1X_3 + 2X_3X_4 - 2X_4X_5 - 3X_3 - X_2 + 2X_1$. Each of the nodes is assigned a processing unit and the network collectively searches for an assignment that maximizes the goodness. The algorithm that is repeatedly executed in each unit/node is called the *protocol* or the *activation function*. A protocol is *uniform* if all the units execute it.

We give examples for the discrete Hopfield network [Hopfield 82] and the Boltzmann machine [Hinton, Sejnowski 86], which are two of the most popular models for connectionist energy minimization:

In the discrete Hopfield model, each unit computes its activation value using the formula

$$X_i = \begin{cases} 1 & \text{iff } \sum_j w_{i,j} X_j \geq -\theta_i, \\ 0 & \text{otherwise.} \end{cases}$$

For Boltzmann machines, the determination of the activation value is stochastic and the probability of setting the activation value of a unit to one is $P(X_i = 1) = 1/(1 + e^{-(\sum_j w_{i,j} X_j + \theta_i)/T})$, where T is the annealing temperature.

Both approaches may be integrated with our topology-based algorithm; in other words, nodes that cannot be identified as parts of a tree-like subnetwork use one of the standard algorithms.

The algorithm

Key idea

We assume that the model of communication is shared memory, multi-reader/single-writer, that scheduling is under a central demon, and that execution is fair. In a *shared memory, multi-reader/single-writer,* each unit has a shared register called the activation register. A unit may read the content of the registers of all its neighbors, but write only its own. *Central demon* means that the units are activated one at a time in an arbitrary order.[2] An execution is said to be *fair* if every unit is activated infinitely often.

The algorithm identifies parts of the network that have no cycles (tree-like subnetworks) and optimizes the free energy on these subnetworks. Once a tree is identified, it is optimized using an adaptation of a constraint optimization algorithm for cycle-free graphs presented in [Dechter et al. 90]. The algorithm belongs to the family of nonserial dynamic programming methods [Bertelé, Brioschi 72].

Let us assume first that the network is is an unrooted tree (cycle-free). Any such network may be directed into a rooted tree. The algorithm is based on the observation that given an activation value (0/1) for a node in a tree, the optimal assignments for all its adjacent nodes are independent of each other. In particular, the optimal assignment to the node's descendants are independent of the assignments for its ancestors. Therefore, each node i in the tree may compute two values: G_i^0 and G_i^1. G_i^1 is the maximal goodness contribution of the subtree rooted at i, including the connection to i's parent whose activation is *one*. Similarly, G_i^0 is the maximal goodness of the subtree, including the connection to i's parent whose activation value is *zero*. The acyclicity property will allow us to compute each node's G_i^1 and G_i^0 as a a simple function of its children's values, implemented as a propagation algorithm initiated by the leaves.

Knowing the activation value of its parent and the values G_j^0, G_j^1 of all its children, a node can compute the maximal goodness of its subtree. When the information reaches the root, it can assign a value (0/1) that maximizes the goodness of the whole network. The assignment information propagates now towards the leaves: knowing the activation value of its parent, a node can compute the preferred activation value for itself. At termination (at stable state), the tree is optimized.

The algorithm has three basic steps:

1) **Directing a tree.** Knowledge is propagated from leaves towards the center of the network, so that after a linear number of steps, every unit in the tree knows

[2] Standard algorithms need to assume the same condition in order to guarantee convergence to a *local* minimum (see [Hopfield 82]). This condition can be relaxed by restricting only that adjacent nodes may not be activated at the same time.

its parent and children.

2) **Propagation of goodness values.** The values G_i^1, G_i^0 are propagated from leaves to the root. At termination, every node knows the maximal goodness of its subtree, and the appropriate activation value it should assign, given that of its parent. In particular, the root can now decide its own activation value so as to maximize the whole tree.

3) **Propagation of activation values.** Starting with the root, each node in turn determines its activation value. After O(depth of tree) steps, the units are in a stable state, which globally maximizes the goodness.

Each unit's *activation register* consists of the fields X_i (the activation value); G_i^0, G_i^1 (the maximal goodness values); and $P_i^1, .., P_i^j$ (a bit for each of the j neighbors of i that indicated which is i's parent).

Directing a tree

The goal of this algorithm is to inform every node of its role in the network and of its child-parent relationships. Nodes with a single neighbor identify themselves as leaves first and then identify their neighbor as a parent (point to it). A node whose neighbors all point towards it identifies itself as a root. A node whose neighbors all but one point towards it selects the one as a parent. Finally, a node that has at least two neighbors *not* pointing towards it identifies itself as being outside the tree.

Each unit uses one bit per neighbor to keep the pointing information: $P_i^j = 1$ indicates that node i sees its jth neighbor as its parent. By looking at P_j^i, node i knows whether j is pointing to it.

Identifying tree-like subnetworks in a general network may be done by the following algorithm:

Tree Directing (for unit i):

1. Initialization: If first time, then for all neighbors j, $P_i^j = 0$. /* Start with clear pointers (this step is not needed in trees) */
2. If there is only a single neighbor (j) and $P_j^i = 0$, then $P_i^j = 1$. /* A leaf selects its neighbor as parent if that neighbor doesn't point to it */
3. else, if one and only one neighbor (k) does not point to i ($P_k^i = 0$), then $P_i^k = 1$, and, for the rest of the neighbors, $P_i^j = 0$. /* k is the parent */
4. Else, for all neighbors j, $P_i^j = 0$. /* Node is either a root or outside the tree */

In Figure 1-(a), we see a cycle-free network after the tree-directing phase. The numbers on the edges represent the values of the P_i^j bits. In Figure 1-(b), a tree-like subnetwork is identified inside a cyclic network; note that node 5 is not a root, since not all its neighbors are pointing towards it.

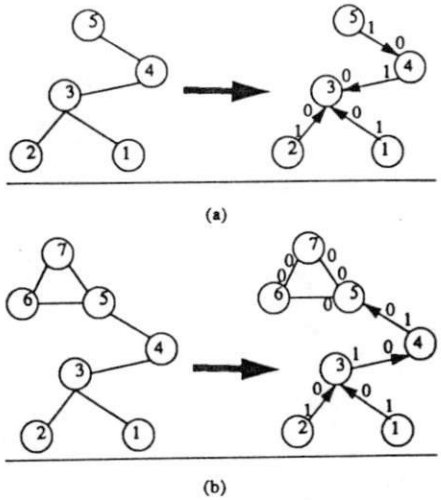

Figure 1: Directing a tree: (a) A tree, (b) A cyclic graph

Propagation of goodness values

In this phase, every node i computes its goodness values G_i^1, G_i^0 by propagating these two values from the leaves to the root (see figure 2).

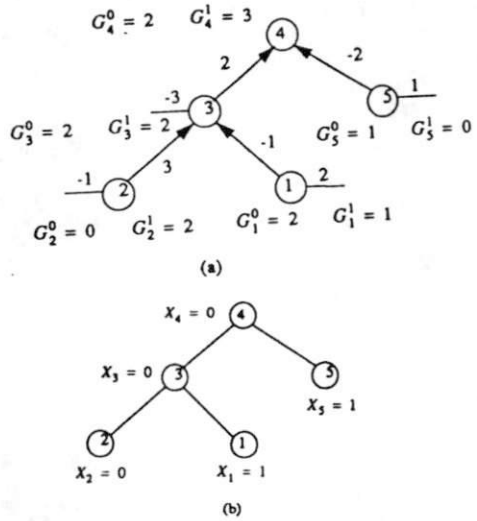

Figure 2: (a) Propagating goodness values, (b) Propagating activation values

Given a node X_i, its parent X_k, and its children $child(i)$ in the tree, it can be shown, based on the energy function (1), that the goodness values obey the following recurrence:

$$G_i^{X_k} = max\{ \sum_{j \in child(i)} G_j^{x_j} + w_{i,k} X_i X_k + \theta_i X_i \}.$$

Consequently, a nonleaf node i computes its goodness values using the goodness values of its children as follows: If $X_k = 0$, then i must decide between setting $X_i = 0$, obtaining a goodness of $\sum_j G_j^0$, or setting

$X_i = 1$, obtaining a goodness of $\sum_j G_j^1 + \theta_i$. This yields

$$G_i^0 = max\{\sum_{j \in child(i)} G_j^0, \sum_{j \in child(i)} G_j^1 + \theta_i\}.$$

Similarly, when $X_k = 1$, the choice between $X_i = 0$ and $X_i = 1$ yields

$$G_i^1 = max\{\sum_{j \in child(i)} G_j^0, \sum_{j \in child(i)} G_j^1 + w_{i,k} + \theta_i\}.$$

The initial goodness values for leaf nodes can be obtained from the above (no children). Thus, $G_i^0 = max\{0, \theta_i\}$, $G_i^1 = \{0, w_i k + \theta_i\}$.

For example: If unit 3 in Figure 2 is zero, then the maximal goodness contributed by node 1 is $G_1^0 = max_{X_1 \in \{0,1\}}\{2X_1\} = 2$ and it is obtained at $X_1 = 1$. Unit 2 (when $X_3 = 0$) contributes $G_2^0 = max_{X_2 \in \{0,1\}}\{-X_2\} = 0$, obtained at $X_2 = 0$, while $G_2^1 = max_{X_2 \in \{0,1\}}\{3X_2 - X_2\} = 2$ is obtained at $X_2 = 1$. As for nonleaf nodes, if $X_4 = 0$, then when $X_3 = 0$, the goodness contribution will be $\sum_k G_k^0 = 2 + 0 = 2$, while if $X_3 = 1$, the contribution will be $-3 + \sum_k G_k^1 = -3 + 1 + 2 = 0$. The maximal contribution $G_3^0 = 2$ is achieved at $X_3 = 0$.

Propagation of activation values

Once a node is assigned an activation value, all its children can activate themselves so as to maximize the goodness of the subtrees they control. When such a value is chosen for a node, its children can evaluate *their* activation values, and the process continues until the whole tree is assigned.

There are two kinds of nodes that may start the process: a root which will choose an activation value to optimize the entire tree, and a non-tree node which uses a standard activation function.

When a root X_i is identified, it chooses the value 0 if the maximal goodness is $\sum_j G_j^0$, while it chooses 1 if the maximal goodness is $\sum_j G_j^1 + \theta_i$. In summary, the root chooses its value according to

$$X_i = \begin{cases} 1 & \text{iff } \sum_j G_j^1 + \theta_i \geq \sum_j G_j^0, \\ 0 & \text{otherwise.} \end{cases}$$

In Figure 2, for example, $G_5^1 + G_3^1 + 0 = 2 < G_5^0 + G_3^0 = 3$ and therefore $X_4 = 0$.

An internal node whose parent is k chooses an activation value that maximizes $\sum_j G_j^{x_i} + w_{i,k} X_i X_k + \theta_i X_i$. The choice therefore, is between $\sum_j G_j^0$ (when $X_i = 0$) and $\sum_j G_j^1 + w_{i,k} X_k + \theta_i$ (when $X_i = 1$), yielding:

$$X_i = \begin{cases} 1 & \text{iff } \sum_j G_j^1 + w_{i,k} X_k + \theta_i \geq \sum_j G_j^0 \\ 0 & \text{otherwise.} \end{cases}$$

As a special case, a leaf i, chooses $X_i = 1$ iff $w_{i,k} X_k \geq -\theta_i$, which is exactly the discrete Hopfield activation function for a node with a single neighbor. For example, in Figure 2, $X_5 = 1$ since $w_{4,5} X_4 = 0 > -\theta_5 = -1$, and $X_3 = 0$ since $G_1^1 + G_2^1 + 2X_4 + \theta_3 = 1 + 2 + 0 - 3 = 0 < G_2^0 + G_1^0 = 2$. Figure 2-(b) shows the activation values obtained by propagating them from the root to the leaves.

A complete activation function

Interleaving the three algorithms described earlier achieves the goal of identifying tree-like subnetworks and maximizes their goodness. In this subsection, we present the complete algorithm, combining the three phases while simplifying the computation. The algorithm is integrated with the discrete Hopfield activation function; however it can be integrated also with other activation function (eg. Boltzmann machine).[3]
Let i be the executing unit, j a non-parent neighbor of i, and k the parent of i:

Optimizing on Tree-like Subnetworks (unit i):

1. Initialization: If first time, then $(\forall j)\ P_i^j = 0$. /*Clear pointers (needed only for cyclic nets)*/
2. Tree directing: If there exists a single neighbor k, such that $P_k^i = 0$,
   then $P_i^k = 1$, and for all other neighbors j, $P_i^j = 0$;
   else, for all neighbors, $P_i^j = 0$.
3. Computing goodness values:
   $G_i^0 = max\{\sum_{j \in child(i)} G_j^0 P_j^i, \sum_{j \in child(i)} G_j^1 P_j^i + \theta_i\}$.
   $G_i^1 = max\{\sum_{j \in child(i)} G_j^0 P_j^i, \sum_{j \in child(i)} (G_j^1 P_j^i + w_{i,j} P_i^j) + \theta_i\}$.
4. Assigning activation values:
   If at least two neighbors are not pointing to i, then /*use standard activation function (Hopfield) */ $X_i = \begin{cases} 1 & \text{if } \sum_j w_{i,j} X_j \geq -\theta_i, \\ 0 & \text{otherwise;} \end{cases}$
   else, /* Node in a tree (including root and leaves) */
   $X_i = \begin{cases} 1 & \text{if } \sum_j ((G_j^1 - G_j^0) P_j^i + w_{i,j} X_j P_i^j) \geq -\theta_i, \\ 0 & \text{otherwise.} \end{cases}$

An example

The example illustrated in Figure 3 demonstrates a case where a local minimum of the standard algorithms is avoided. Standard algorithms may enter such local minimum and stay in a stable state that is clearly wrong.

The example is a variation on a harmony network [Smolensky 86, page 259] and an example from [McClelland et al. 86, page 22]. The task of the network is to identify words from low-level line segments. Certain patterns of line segments excite units that represent characters, and certain patterns of characters excite units that represent words. The line strokes used

[3] Note how similar the new activation function is to the original Hopfield function.

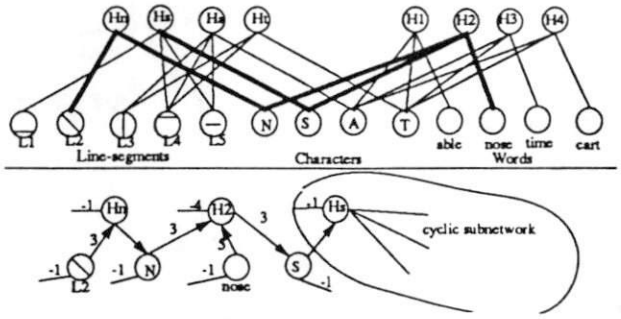

Figure 3: A Harmony network for recognizing words: Local minima along the subtrees are avoided

to draw the characters are the input units: L1,..., L5. The units "N," "S," "A," and "T" represent characters. The units "able," "nose," "time," and "cart" represent words, and Hn, Hs, Ha, Ht, H1,...,H4 are hidden units required by the Harmony model. For example, given the line segments of the character S, unit L4 is activated (input), and this causes units Hs and "S" to be activated. Since "NOSE" is the only word that contains the character "S," both H2 and the unit "nose" are also activated and the word "NOSE" is identified.

The network has feedback cycles (symmetric weights) so that ambiguity among characters or line-segments may be resolved as a result of identifying a word. For example, assume that the line segments required to recognize the word "NOSE" appear, but the character "N" in the input is blurred and therefore the setting of unit L2 is ambiguous. Given the rest of the line segments (e.g., those of the character "S"), the network identifies the word "NOSE" and activates units "nose" and H2. This causes unit "N" to be activated and so are all of its line segments. Thus the ambiguity of L2 is resolved.

The network is indeed designed to have a global minimum when L2, Hn, "N," H2, and "nose" are all activated; however, standard connectionist algorithms may fall into a local minimum when all these units are zero, generating goodness of $5 - 4 = 1$. The correct setting (global minimum) is found by our tree-optimization protocol (with goodness: 3-1+3-1+3-1+5-1-4+3-1+5=13). The thick arcs in the upper network of Figure 3 mark the arcs of a tree-like subnetwork. This tree-like subnetwork is drawn with pointers and weights in the lower part of the figure. Node "S" is not part of the tree and its activation value is set to one because the line-segments of "S" are activated. Once "S" is set, the units along the tree are optimized (by setting them all to one) and the local minimum is avoided.

Limitations and extensions

We have shown a way to enhance the performance of connectionist energy minimization networks without loosing much of the simplicity of the standard approaches. Our simple algorithm is limited in two ways, however. First, the central demon (or atomicity of the protocol) is not a realistic restriction. We would like the network to work correctly also under a *distributed demon*, where any subset of units may be scheduled for execution at the same time. Second, we would like the algorithm to be *self-stabilizing*. It should converge to a legal, stable state given enough time, even after noisy fluctuations that cause the units to execute an arbitrary program state and the registers to have arbitrary content.

Scheduling demons

Two negative results (presented in [Collin et al. 91] following [Dijkstra 74]) regarding the feasibility of distributed constraint satisfaction, can be extended and proved for computing the global minimum of energy functions: (1) No uniform deterministic distributed algorithm exists that guarantees a stable global minimum under a distributed demon, even for simple chain-like trees, and (2) no uniform deterministic algorithm exists that guarantees a stable global minima under a *central demon* for *cyclic* networks, even for simple rings. For proofs see [Pinkas, Dechter 92].

These negative results should not discourage us, since they rely on obscure infinite sequences of executions which are unlikely to occur under a truly random demon. Our algorithm will converge to a global minimum under a distributed demon in each of the following cases: (1) If step 2 of the protocol in section is atomic; (2) if for every node i and every neighbor j, node i is executed without j infinitely often; (3) if one node is unique and acts as a root, that is, does not execute step 2 (an almost uniform protocol); and (4) if the network is cyclic.

Self-stabilization

A protocol is self-stabilizing if in any fair execution, starting from any input configuration and any program state, the system reaches a valid stable configuration.

The algorithm in Section is self-stabilizing for cycle-free networks (trees), and it remains self-stabilizing under distributed demon if every node executes without a neighbor infinitely often or if one node is acting as a root. The algorithm is not self-stabilizing for cyclic networks (see [Pinkas, Dechter 92]) due to its tree-directing sub-protocol. To solve this problem, we may use a variation of the self-stabilizing tree-directing protocol of [Collin et al. 91]. This algorithm remains self-stabilizing even in cyclic networks, although it is more complex and requires more space.

Summary

We have shown a uniform self-stabilizing connectionist activation function that is guaranteed to find a global minimum of tree-like symmetric networks in linear time. The algorithm optimizes tree-like *subnetworks* within general (cyclic) networks. The algorithm can be extended to be self-stabilizing for *all* cyclic networks, but more space will then be needed.

We stated two negative results: (1) Under a pure distributed demon, no *uniform* algorithm exists to optimize even simple chains, and (2) no uniform algorithm exists to optimize simple cyclic networks (rings) even under a central demon. We conjecture that these negative results are not of significant practical importance, since in truly random scheduling demons the probability of having such pathological executions approaches zero. Our algorithm remains correct under a distributed demon (without atomicity) if some weak assumptions are made.

References

D. H. Ballard, P. C. Gardner, M. A. Srinivas, "Graph problems and connectionist architectures," Department of Computer Science, University of Rochester, Technical Report 167, 1986.

U. Bertelé, F. Brioschi, *"Nonserial dynamic programming,"* Academic Press, 1972.

Z. Collin, R. Dechter, S. Katz, "On the feasibility of distributed constraint satisfaction," In IJCAI-91: *Proceedings of 12th International Conference on Artificial Intelligence*, Sydney, Australia, 1991.

R. Dechter, J. Pearl, "Network-based heuristics for constraint-satisfaction problems," *Artificial Intelligence 34*, pp. 1-38, 1988.

R. Dechter, A. Dechter, J. Pearl, "Optimization in constraint networks," in R.M. Oliver and J.Q. Smith (editors), *Influence diagrams, belief nets and decision analysis*, John Wiley, 1990.

E.W. Dijkstra, "Self-stabilizing systems in spite of distributed control," *Communications of the ACM 17*, pp. 643-644, 1974.

J.A Feldman, D.H. Ballard, "Connectionist models and their properties," *Cognitive Science* 6, 1982.

G.E Hinton, T.J. Sejnowski, "Learning and relearning in Boltzman machines," in J. L. McClelland, D. E. Rumelhart, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition I*, MIT Press, 1986.

J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences 79*, pp. 2554-2558, 1982.

J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the National Academy of Sciences* , pp. 3088-3092, 1984.

J.J. Hopfield, D.W. Tank, "Neural computation of decisions in optimization problems," *Biological Cybernetics 52*, pp. 144-152, 1985.

S. Kasif, S. Banerjee, A. Delcher, G. Sullivan, "Some results on the computational complexity of symmetric connectionist networks," Department of Computer Science, The John Hopkins University, Technical Report JHU/CS-89/10, 1989.

J. L. McClelland, D. E. Rumelhart, G.E Hinton, J.L. McClelland, "The appeal of PDP," in J. L. McClelland, D. E. Rumelhart, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition I*, MIT Press, 1986.

C. Papadimitriou, A. Shaffer, M. Yannakakis, "On the complexity of local search," in *ACM Symposium on the Theory of Computation*, pp. 438-445, 1990.

C. Peterson, E. Hartman, "Explorations of mean field theory learning algorithm," *Neural Networks 2*, no. 6, 1989.

G. Pinkas, "Energy minimization and the satisfiability of propositional calculus," *Neural Computation 3*, no. 2, 1991.

G. Pinkas, R. Dechter, "An improvement for Hopfield networks that avoids local minima along tree-like subnetworks in linear time," Department of Computer Science, Washington University, Technical Report WUCS-92-2, 1992.

D.E. Rumelhart, G.E Hinton, J.L. McClelland, "A general framework for parallel distributed processing," in J. L. McClelland, D. E. Rumelhart, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition I*, MIT Press, 1986.

P. Smolensky, "Information processing in dynamic systems: Foundations of harmony theory," in J. L. McClelland, D. E. Rumelhart, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition I*, MIT Press, 1986.