Report Number: WUCS-93-9

1993-01-01

Representing and Learning Propositional Logic in Symmetric Connectionist Networks

Gadi Pinkas

The chapter presents methods for efficiently representing logic formulas in connectionist networks that perform energy minimization. Algorithms are given for transforming any formula into a network in linear time and space and for learning representations of unknown formulas by observing examples of satisfying truth assignments. The relaxation process that underlies networks of energy minimization reveals an efficient hill climbing algorithm for satisfiability problems. Experimental results indicate that the parallel implementation of the algorithm with give extremely good average-case performance, even for large-scale, hard satisfiability problems (randomly generated).
... Read complete abstract on page 2.

Representing and Learning Propositional Logic in Symmetric Connectionist Networks

Gadi Pinkas

Complete Abstract:

The chapter presents methods for efficiently representing logic formulas in connectionist networks that perform energy minimization. Algorithms are given for transforming any formula into a network in linear time and space and for learning representations of unknown formulas by observing examples of satisfying truth assignments. The relaxation process that underlies networks of energy minimization reveals an efficient hill climbing algorithm for satisfiability problems. Experimental results indicate that the parallel implementation of the algorithm with give extremely good average-case performance, even for large-scale, hard satisfiability problems (randomly generated).

Representing and Learning Propositional Logic in Symmetric Connectionist Networks

Gadi Pinkas

WUCS-93-09

March 1993

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130-4899

LaTeX documents for the Machine Intelligence
and Pattern Recognition series
W.j. Maas

Representing and Learning Propositional Logic in Symmetric Connectionist Networks

Gadi Pinkas

Department of Computer Science and The Center for Optimization and Semantic Control, Box 1045, Washington University, St. Louis, 63130.

Abstract

The chapter presents methods for efficiently representing logic formulas in connectionist networks that perform energy minimization. Algorithms are given for transforming any formula into a network in linear time and space and for learning representations of unknown formulas by observing examples of satisfying truth assignments. The relaxation process that underlies networks of energy minimization reveals an efficient hill climbing algorithm for satisfiability problems. Experimental results indicate that the parallel implementation of the algorithm will give extremely good average-case performance, even for large-scale, hard satisfiability problems (randomly generated).

1. Introduction

An almost standard approach in AI going back to [MacCarthy 68], is to represent an agent's knowledge as a collection of formulas, which can be viewed as a knowledge base. An agent is then said to know a fact, if it is provable from the formulas in the knowledge base. The majority of existing formal AI reasoning systems are based on this logicist view; i.e., the use of logic formulas for representation and the use of a formal proof theory in order to reason about facts or beliefs not mentioned explicitly.

While scientists in traditional, symbolic AI were concentrating on development of powerful knowledge representation systems, connectionists were concentrating on powerful learning and adaptation mechanisms. Connectionism was criticized for lacking mechanisms like compositionality and systematicity, which are essential for high level cognitive tasks and are easy for symbolic approaches [Fodor, Pylyshyn 88]. It is clear that we would like to have systems that have sufficient expressive power, that perform quickly (the brain suggests massive parallelism), and that are capable

of learning and adjusting. As [Hinton 90] pointed out, the ultimate goal for both scientific approaches is to find efficient learning procedures for representationally powerful systems,...

Appreciating the benefits of both the connectionist paradigm and the logicist approach, this chapter tries to build the foundations for a bridge across the two. Dealing with foundations, it concentrates on a logical formalism that is simple and well understood - propositional logic. In this chapter, I shall consider only propositional knowledge; however, the approach can be extended to predicate calculus with nonmonotonic abilities [Pinkas 91c].

Among the different connectionist models, I choose to consider those with symmetric matrix of weights. This family of models includes Hopfield networks [Hopfield 82], [Hopfield 84], Boltzmann machines [Hinton, Sejnowski 86], harmony theory [Smolensky 86], mean field theory [Hinton 89], and other variations. The reasons for using *symmetric* connectionist networks (SCNs) are the following:

(i) Symmetric networks have a natural parallel realization. Implementations exist in analog, digital and optical hardware [Alspector et al. 88], [Abu Mustafa, Psaltis 87].
(ii) Symmetric networks can be characterized by energy functions. These functions make it easier to specify the networks' behavior [Feldman 85].
(iii) Symmetric networks have been used successfully to express and solve (approximate) hard problems [Hopfield, Tank 85].
(iv) Symmetric networks are capable of representing a large set of asymmetric networks [Pinkas 91b];[1] therefore they are quite powerful and we will not lose expressive power if we restrict ourselves to the symmetric case.[2]

My purpose is to show that: 1) Propositional logic can be represented efficiently in symmetric networks and SNs can be used for inference. 2) SNs can learn representations of unknown formulas by observing examples that satisfy these formulas. 3) The algorithm implemented by the network is efficient on average.

The chapter is organized as follows. Section 2 presents the energy paradigm. Section 3 shows that propositional logic can be represented compactly in SNs. Section 4 provides surprising experimental data. Section 5 describes a learning algorithm that enables a network to obtain repre-

[1] In fact, every non-oscillating network of binary threshold units is representable in SCNs.

[2] Sometimes an asymmetric form of a symmetric network will perform better; therefore, for efficiency, we may consider not to restrict ourselves to the symmetric case.

sentations of formulas by observing examples, and Section 6 discusses the results and related work. Proofs for the theorems appear in [Pinkas 92].

2. The energy paradigm

Finding minima for quadratic functions is the essence of symmetric connectionist models used for parallel constraint satisfaction [Hopfield 82], [Hinton, Sejnowski 86], [Smolensky 86]. These models are characterized by a recurrent network architecture, a symmetric matrix of weights (with zero diagonal) and a quadratic energy function that should be minimized. Each unit asynchronously computes the gradient of the function and adjusts its activation value, so that energy decreases gradually. The network eventually reaches equilibrium, settling on either a local or a global minimum. [Hopfield 82] demonstrated that certain complex optimization problems can be stated as constraints that are expressed in quadratic energy functions and be approximated using these kind of networks.

There is a direct mapping between these networks and the quadratic energy functions they minimize. Every quadratic energy function can be translated into a corresponding network and vice versa. Weighted arcs (i.e., pairwise connections) in the network correspond to weighted terms of two variables in the energy function (with opposite sign). Thresholds of units in the network correspond to single-variable terms in the function. Most of the time I shall not distinguish between the function and the network that minimizes it. An example of a network and its energy function is given in Figure 1.

2.1. High-order energy functions

To represent arbitrary logic formulas, a network will need the power of either high-order connections or hidden units. This section defines high-order networks, and shows how to convert them into standard (pair-wise) networks by introducing new hidden units.

High-order connectionist networks have sigma-pi units [Rumelhart et al. 86] with multiplicative connections. It is a common intuition that high-order networks can better express high-order problems [Sejnowski 86], and can compute functions that are not computable if only second-order connections are allowed [Williams 86]. In particular, *symmetric* networks can be easily extended to handle high-order connections. Naturally, such networks may be viewed as minimizing high-order energy functions [Sejnowski 86].

A k-order energy function is a function $E : \{0,1\}^n \rightarrow \mathcal{R}$ that can be expressed as sum of products, with product terms of up to k variables. A

Fig. 1. A symmetric network that represents the function $E = -2NT - 2ST - 2WT + 5T + 2RN - WN + W - R + S$.

k-order energy function is denoted by: $E^k(x_1, \ldots, x_n) =$

$$\sum_{1 \le i_1 < i_2 < \cdots < i_k \le n} w_{i_1,\ldots,i_k} X_{i_1} \cdots X_{i_k} + \cdots + \sum_{1 \le i \le n}^{1} w_i X_i$$

Quadratic energy functions (or second-order functions) are special cases of the high-order case:

$$\sum_{1 \le i < j \le n} w_{ij} X_i X_j + \sum_{i \le n} w_i X_i.$$

In the high-order model each node is assigned a sigma-pi unit that updates its activation value using:

$$net_i = \frac{dE}{dX_i} = \sum_{i_1 \cdots i \cdots i_k} -w_{i_1,\ldots,i,\ldots,i_k} \prod_{1 \le j \le k, i_j \ne i} X_{i_j}$$

$$a_i = F(net_i)$$

where $a_i = F(net_i)$ is the standard update rule that is unique to the model we wish to extend. In the Hopfield model for example, $F(net_i) = 1$ if $net_i > 0$ and $F(net_i) = 0$ otherwise. A high-order network (see Figure 2) is a hyper graph, where k-order terms are translated into hyper-arcs connecting k nodes. The arcs are not directed (the weight is the same for every node that is part of the arc) and the weight of an arc is determined by the weight of the corresponding term in the energy function (with an opposite sign).

Fig. 2. A cubic network that represents $E = -NSW + 2RN - WN + W - R + S$ using Sigma-Pi units and a cubic hyper-arc (Its is equivalent to the network of Figure 1 without hidden units).

As in the quadratic case, there is a translation back and forth between k-order energy functions and symmetric high-order networks with k-order sigma-pi units.

The variables of an energy function can be arbitrarily divided into two sets: visible variables and hidden variables. The hidden variables correspond to the hidden units of the corresponding network, and the visible variables correspond to the visible units. An energy function with both

hidden and visible variables is denoted usually as a function $E(x,t)$, where x represents the visible variables and t represents the hidden variables.

An assignment of zeros and ones to the visible variables is called a visible state. The value of the hidden units is not usually of any interest to the external observer. The visible units, on the other hand, are used as inputs and outputs. The user clamps (or adds bias to) some units that act as inputs, and looks at the output units after the network has settled. The values of the visible units after an equilibrium is reached, is considered as the answer of the network. Later in this chapter, I'll interpret visible states as truth assignments: the visible variables are viewed as atomic propositions, 1 is interpreted as true and 0 is interpreted as false.

The set of minimizing vectors projected onto the visible variables is called the visible solutions of the minimization problem $(\mu(E) = \{x \mid (\exists t)E(x,t) = min_{y,z}\{E(y,z)\}\})$. Connectionist models like Boltzmann machines, harmony theory, mean field theory, as well as other variations, may be looked as searching for a *global minimum*[3] of the corresponding energy functions. Local minima or spurious memories may exist. In general however, local minima are considered to be undesirable phenomena, and cause a degradation in the performance of the network.

For every network with energy function $E(x,t)$ and with t hidden variables, define the *characteristic* function of the network to be:

$$Erank_E(x) = min_y\{E(x,y).\}$$

The $Erank_E$ function defines the energy of all visible states (the energy of a visible state is the energy level obtained when the visible units are clamped with the state's values, and the hidden units are free to settle so that a minimum is reached). This $Erank_E$ function characterizes the network's behavior: it is independent of the hidden units and it is also independent of the exact topology of the original network. I'll use the characteristic function to show equivalence between different networks.

2.2. Converting high-order networks into equivalent low-order networks

Two energy functions are *strongly equivalent*, if their corresponding characteristic ($Erank$) functions are equal up to a constant difference; i.e: $E_1 \approx E_2$ iff $Erank_{E_1} = Erank_{E_2} + c$. Networks that are strongly equivalent not only have the same set of global minima, but also have a very similar energy surface and induce the same ordering on the visible states; i.e., if s_1 and s_2 are visible states then same ordering means that $E_1(s_1) < E_1(s_2)$ iff $E_2(s_1) < E_2(s_2)$.

[3] Several global minima may exist, all with the same energy level.

The following theorem is useful for converting any high-order network into a strongly equivalent low-order one with additional hidden units.

THEOREM **2.1** – *Any k-order term* $(w\prod_{i=1}^{k} x_i)$, *with* NEGATIVE *coefficient* w, *can be replaced by the quadratic terms:* $\sum_{i=1}^{k} 2wX_iT - (2k-1)wT$ *generating a strongly equivalent energy function with one additional hidden variable* T.

– *Any k-order term* $(w\prod_{i=1}^{k} x_i)$, *with* POSITIVE *coefficient* w, *can be replaced by the terms:*
$w\prod_{i=1}^{k-1} x_i - (\sum_{i=1}^{k-1} 2wX_iT) + 2wX_kT + (2k-3)wT$, *generating a strongly equivalent energy function of order* $k-1$ *with one additional hidden variable* T.

EXAMPLE **2.1** The cubic function $E = -NSW + NS + RN - WN + W$ is strongly equivalent to $-2NT - 2ST - 2WT + 5T + NS + RN - WN + W$, (introducing the hidden variable T). The corresponding high-order network appears in Figure 2 while the equivalent quadratic one appears in Figure 1.

3. Propositional Logic and Energy Functions

3.1. Propositional Logic - A Review

DEFINITION **3.1** A *well formed formula* (a WFF) is an expression that combines atomic propositions (variables) and connectives $(\vee, \wedge, \neg, \rightarrow, (,))$. A WFF is defined recursively:
- if φ is an atomic proposition (a single variable), then φ is a WFF;
- if φ_1 and φ_2 are WFFs, then so are: $(\varphi_1 \vee \varphi_2), (\varphi_1 \wedge \varphi_2), (\varphi_1 \rightarrow \varphi_2), \neg\varphi_1$ and (φ_1);
- nothing else is a WFF.

For example, $((A \vee \neg B) \rightarrow C)$ is a WFF.

DEFINITION **3.2** The *characteristic function* of a formula φ in n variables is defined to be $H_\varphi : 2^n \rightarrow \{0,1\}$ such that:

- $H_{X_i}(X_1, \ldots, X_n) = X_i$
- $H_{\neg\varphi}(X_1, \ldots, X_n) = 1 - H_\varphi(X_1, \ldots, X_n)$
- $H_{(\varphi_1 \vee \varphi_2)}(X_1, \ldots, X_n) = H_{\varphi_1}(X_1, \ldots, X_n) + H_{\varphi_2}(X_1, \ldots, X_n) - H_{\varphi_1}(X_1, \ldots, X_n) \times H_{\varphi_2}(X_1, \ldots, X_n)$
- $H_{(\varphi_1 \wedge \varphi_2)}(X_1, \ldots, X_n) = H_{\varphi_1}(X_1, \ldots, X_n) \times H_{\varphi_2}(X_1, \ldots, X_n)$
- $H_{(\varphi_1 \rightarrow \varphi_2)}(X_1, \ldots, X_n) = H_{(\neg\varphi_1 \vee \varphi_2)}(X_1, \ldots, X_n)$

DEFINITION **3.3** A *model* (or truth assignment) is a vector of binary values that assigns 1 ("true") or 0 ("false") to each of the atomic propositions.

A model x *satisfies* a WFF φ, iff its characteristic function H_φ is evaluated to "one" given the vector x.

A model that satisfies a formula φ is a truth assignment that satisfies the constraints imposed by the formula. For example, given $A \vee \neg B$, the satisfying models will be those that assign 1 to A or 0 to B or both.

DEFINITION 3.4 The *satisfiability search* problem for a WFF φ is to find an x, if one exists, such that $H_{\varphi(x)} = 1$.

The decision problem of whether a formula has a satisfiable model is known to be NP-complete; the satisfiability search is also NP-complete [Garey,Johnson79].

DEFINITION 3.5 *Semantic entailment* in propositional logic is denoted by the relation $\models$. A formula ψ *entails* a formula φ ($\psi \models \varphi$) iff all the models that satisfy ψ also satisfy φ.

For example $((A \vee \neg B) \wedge B) \models A$.

3.2. Equivalence between Propositional Formulas

In order to convert a WFF *efficiently* into an energy function, we need an intermediate step. The WFF is transformed first into an equivalent form called Conjunction of Triples Form (CTF).

The atomic propositions of a WFF can be divided into visible and hidden propositions. Atomic propositions that are of interest for a certain application are called "visible variables" and are denoted by X. Additional atomic hidden propositions (denoted by T) may be added without changing the set of relevant models that satisfy the WFF.

DEFINITION 3.6 The set of models that satisfy φ projected onto the visible variables is then called the *visible satisfying models*; i.e., $\{x \mid (\exists t) H_\varphi(x,t) = 1\}$.

DEFINITION 3.7 Two WFFs are *equivalent* if the set of visible satisfying models of one is equal to the set of visible satisfying models of the other.

DEFINITION 3.8 A WFF φ is in *Conjunction of Triples Form* (CTF) if $\varphi = \bigwedge_{i=1}^{m} \varphi_i$ and every φ_i is a sub-formula of at most three variables.[4]

[4]CTF differs from the familiar Conjunctive Normal Form (3-CNF). The φ_i's are WFFs of up to 3 variables that may include any logical connectives and are not necessarily a disjunction of three literals as in 3-CNF. As a result, the CTF is more compact than its 3-CNF equivalent. For example, the CTF formula $(A \leftrightarrow (B \vee C))$ is translated into a 3-CNF with five clauses $(\neg A \vee B \vee C) \wedge (A \vee \neg B \vee C) \wedge (A \vee \neg B \neg C) \wedge (A \vee B \vee \neg C) \wedge (A \vee \neg B \vee \neg C)$.

If we can convert any WFF to a CTF of the same order of size, we will then be able to convert the CTF into a quadratic energy function that preserves the size. Thus, the motivation for CTF is only technical; it guarantees that the generated network is of the same order of size as the original formula.

Every WFF can be converted into an equivalent WFF in CTF by adding hidden variables. Intuitively, a new hidden variable is generated for every binary connective (e.g. $\vee, \rightarrow$) except for the top-most one, and this logical operation is "named" with the new variable by using the connective ($\leftrightarrow$).

The number of hidden variables and new connections needed is on the order of the number of binary connectives in the original formula. The size of the new formula (in CTF) is therefore linear in the size of the original formula.

EXAMPLE **3.1** Converting $\varphi = (\neg((\neg A) \wedge B) \rightarrow (\neg C \rightarrow D))$ into CTF:
From $(\neg((\neg A) \wedge B))$ we generate $(\neg(\neg A \wedge B) \leftrightarrow T_1)$ by adding a new hidden variable T_1.
From $(\neg C \rightarrow D)$ we generate $((\neg C \rightarrow D) \leftrightarrow T_2)$ by adding a new hidden variable T_2.
For the top-most connective $(\rightarrow)$, $(T_1 \rightarrow T_2)$ is generated.
The conjunction of these sub-formulas is therefore: $(\neg(\neg A \wedge B) \leftrightarrow T_1) \wedge ((\neg C \rightarrow D) \leftrightarrow T_2) \wedge (T_1 \rightarrow T_2)$. It is in CTF and is equivalent to φ.

The next subsection shows how to reduce a conjunction of triples into energy terms.

3.3. Constructing Energy Functions from WFFs

Let us associate the visible variables of an energy function with the visible atomic variables of a WFF.

DEFINITION **3.9** A WFF φ *describes* an energy function E if the set of visible satisfying models of φ is equal to the set of visible solutions of the minimization of E. Formally, φ describes E iff $\Gamma_\varphi = \{\boldsymbol{x} \mid (\exists \boldsymbol{t}) H_\varphi(\boldsymbol{x}, \boldsymbol{t}) = 1\}$
$= \{\boldsymbol{x} \mid (\exists \boldsymbol{t}) E(\boldsymbol{x}, \boldsymbol{t})) = MIN_{\boldsymbol{y}}\{E(\boldsymbol{y})\}\} = \Gamma_E$.

If φ describes E, then φ preserves the *weak* meaning of E by distinguishing the visible solutions of E as the visible satisfying models of φ.

Assume $\varphi = \bigwedge_{i=1}^{m} \varphi_i$. The φ_is are called *sub-formulas*. The energy function of a WFF φ is a function $E_\varphi : \{0,1\}^n \rightarrow \mathcal{N}$, that for every model $\boldsymbol{x}$, penalizes sub-formulas of the WFF that are not satisfied by $\boldsymbol{x}$. It computes the characteristic of the negation of every sub-formula φ_i (in the upper level

of the WFF's conjunctive structure):

$$E_{\varphi}(\boldsymbol{X}) = \sum_{i=1}^{m}(H_{\neg\varphi_i}(\boldsymbol{X})) = \sum_{i=1}^{m}(1 - H_{\varphi_i}(\boldsymbol{X}))$$

If $\boldsymbol{x}$ does not satisfy φ_i then $\boldsymbol{x}$ satisfies $\neg\varphi_i$ and therefore $H_{\neg\varphi_i}(\boldsymbol{x}) = 1$. Similarly, if $\boldsymbol{x}$ satisfies φ_i then $H_{\neg\varphi_i}(\boldsymbol{x}) = 0$. If all the sub-formulas are satisfied, E_{φ} has the value zero; otherwise, the function computes the number of sub-formulas that are unsatisfied.

LEMMA 3.1 *φ is satisfied by $\boldsymbol{x}$ iff E_{φ} is minimized by $\boldsymbol{x}$ and the global minimum is zero.*

We may conclude therefore, that every satisfiable WFF φ has a function E_{φ}, such that E_{φ} describes φ.

EXAMPLE 3.2

$$\begin{aligned} E_{((N\wedge S)\rightarrow W)\wedge(R\rightarrow\neg N)\wedge(N\vee\neg W)} &= H_{\neg((N\wedge S)\rightarrow W)} + H_{\neg(R\rightarrow(\neg N))} + H_{\neg(N\vee(\neg W))} \\ &= H_{N\wedge S\wedge(\neg W)} + H_{R\wedge N} + H_{(\neg N)\wedge W} \\ &= (NS(1-W)) + (RN) + ((1-N)W) \\ &= -NSW + NS + RN - WN + W \end{aligned}$$

The corresponding high-order network appears in Figure 3. Note, that the symbols of the atomic propositions of φ are overloaded and used also as the variables of E.[5]

THEOREM 3.1 IF φ IS SATISFIABLE THEN φ DESCRIBES E_{φ}.

COROLLARY 3.1 *If φ is a satisfiable conjunction of WFFs, each of maximum k variables, then φ describes a k-order (at most) energy function with no hidden variables.*

EXAMPLE 3.3

$$\begin{aligned} E_{A\wedge(B\vee(\neg C))} &= E_A + E_{(B\vee(\neg C))} \\ &= (1-A) + H_{((\neg B)\wedge C)} \\ &= (1-A) + (1-B)C = 1 - A + C - BC \end{aligned}$$

THEOREM 3.2 EVERY SATISFIABLE WFF DESCRIBES SOME QUADRATIC ENERGY FUNCTION.

The following algorithm transforms a WFF into a quadratic energy function, generating $O(length(\varphi))$ hidden variables:

(i) Convert φ into CTF φ' (Section 3.2).
(ii) Convert the CTF φ' into a cubic energy function $E_{\varphi'}$ and simplify it to a sum of products form (Section 3.3).

[5] This deliberately confuses the symbols in the WFF with the corresponding variables in the energy function.

Fig. 3. The network (cubic) that represents $((N \wedge S) \rightarrow W) \wedge (R \rightarrow \neg N) \wedge (N \vee \neg W)$.

(iii) Convert the cubic terms in $E_{\varphi'}$ into quadratic terms. Each of the triples generates only one new variable.

The algorithm generates a network whose size is linear in the number of binary connectives of the original WFF. The number of connections (fan-out) associated with a single hidden unit is bounded by a constant.

EXAMPLE **3.4** Converting

$$(((((T_1 \leftrightarrow A) \wedge (T_2 \leftrightarrow B)) \wedge (T_3 \leftrightarrow (T_1 \wedge T_2))) \wedge (T_4 \leftrightarrow (\neg C))) \wedge (T_3 \vee T_4))$$

Eliminating $\leftrightarrow$:

$$((\neg T_1) \vee A) \wedge (T_1 \vee (\neg A)) \wedge ((\neg T_2) \vee B) \wedge (T_2 \vee (\neg B)) \wedge ((\neg T_3 \vee T_1) \wedge ((\neg T_3) \vee T_2) \wedge ((T_3 \vee (\neg T_1) \vee (\neg T_2)) \wedge ((\neg T_4) \vee (\neg C)) \wedge (T_4 \vee C) \wedge (T_3 \vee T_4)$$

Generating the cubic (3-order) energy function:

$$\begin{gathered} T_1(1-A) + (1-T_1)A + T_2(1-B) + (1-T_2)B + T_3(1-T_1) + T_3(1-T_2) \\ +(1-T_3)T_1T_2 + CT_4 + (1-T_4)(1-C) + (1-T_3)(1-T_4) \\ = \\ T_1 - 2AT_1 + A + T_2 - 2BT_2 + B + T_3 - T_1T_3 - T_2T_3 \\ +T_1T_2 - T_1T_2T_3 + 2CT_4 + 2 - C - 2T_4 + T_3T_4 \end{gathered}$$

Converting into a quadratic function:

$$T_1 - 2AT_1 + A + T_2 - 2BT_2 + B + T_3 - T_1T_3 - T_2T_3 + T_1T_2$$
$$-2T_1T_5 - 2T_2T_5 - 2T_3T_5 + 5T_5 + 2CT_4 + 2 - C - 2T_4 + T_3T_4$$

COROLLARY **3.2** *Any boolean function h may be implemented in a quadratic energy minimization network of size that is proportional to the length of the boolean expression φ that is characterized by h.*

3.3.1. Complexity analysis

The algorithm described above transforms a WFF into a quadratic energy function in time linear in the length of the WFF: The conversion into conjunction of triples and the conversion into the cubic energy function are operations that parse the nested structure of the WFF in linear time. Simplifying a 3-variable subexpression takes a constant time, and conversion of all the cubic terms into quadratic terms is linear in their number (the number of terms is of the order of the number of binary connectives).

The number of hidden units that are generated is linear in the length of the original WFF: Conversion into triples generates new variables as the number of binary connectives in the WFF. The conversion of triples into quadratic terms generates hidden variables in the same order (only triples of three variables may generate a cubic term).

The connectivity (fan-out) of all these hidden variables is at most four for those generated by the conversion into triples and at most three for those generated by the conversion into a quadratic function. The visible variables may have a fan-out of $O(n)$.

4. Experimental Results

Experiments have been made on randomly generated satisfiability problems (3-SAT). The simulations have managed to find satisfying solutions to large-scale 3-SAT problems with remarkable speed. I have tried three types of algorithms inspired from Hopfield networks, Boltzmann machines and Mean-Field networks.[6]

4.1. Simulations

The Hopfield Version:
Do until MAXT tries or until a solution is found:
In each try:
- Assign random (zero/one) values to the units.

[6] William Chen assisted me during the experiments both with ideas and with the programming.

- Perform Hopfield cycles until either MAXC cycles have been executed, a solution is found or until MAXP continuous cycles have been performed without reducing the energy.

A Hopfield cycle is one that asynchronously updates all the units that need to be updated (each unit is updated only once) by randomly selecting a unit that has not been selected before in this cycle, and updating its value in the following way:
- If $net_i > 0$, the unit becomes one;
- If $net_i < 0$, the unit becomes zero;
- If $net_i = 0$, the unit is flipped.

The Boltzmann Version:
Do until MAXT tries or until a solution is found:
In each try:
- Assign random (zero/one) values to the units.
- Starting with temp=1 until temp=0:
  - Perform a Boltzmann cycle;
  - Reduce temp by 1/STEPS.

When the temperature is zero, Hopfield cycles are executed until either MAXC tries have been performed,[7] a solution has been found, or MAXP continuous cycles could not reduce the energy.
- If MAXT tries have not been executed and a solution hasn't been found, another annealing begins with STEPS=STEPS+DELTA (annealing slows).

A Boltzmann cycle is an asynchronous update of all the units (every unit is visited in random order but only once), flipping their value stochastically with a probability which is a function of net_i and the temperature (see [Hinton, Sejnowski 86].

The Mean-Field Version:
As in the Boltzmann version, the simulator tries MAXT annealings (each time the annealing is slower); however, the first annealing is done using Mean-Field theory (MFT) cycles (see [Peterson, Hartman 89]), while the rest of the annealings are done using Boltzmann cycles.[8]

A MFT cycle is an asynchronous update of all the units (as in Boltzmann cycle). The units are selected in random order , and each unit in its turn updates its own activation value using the deterministic activation function for MFT (see [Peterson, Hartman 89]).

4.2. The Experiments

Random 3-SAT formulas of n variables and m clauses were generated in the following way:

[7] The number of cycles includes those executed during the annealing.
[8] Trying more MFT cycles is not a good strategy because MFT is deterministic.

- Generate a random truth assignment; i.e., zero/one vector of n bits. The formula to be generated will be satisfied by this assignment.
- Starting with an empty formula, until m clauses are added:
  - – Randomly generate a 3-variable clause (selection of 3 out of n variables).
  - – If the clause is new and is satisfied by the assignment, then add the new clause to the formula.

In the experiments conducted, a ratio of 4.3 between the number of clauses and the number of variables (m/n) was kept. This ratio was found to generate "hard" satisfiability problems [Mitchell et al.92].[9] One hundred formulas were generated for each of 50, 70, 100, 120 and 200 variables, and only 50 formulas were generated for 300, 400 and 500 variables. The parameters used for the simulations appear in the table below.

n	m	MAXT	MAXC	MAXP	STEPS	DELTA
50	215	50	250	20	8	1
70	301	50	350	20	11	1
100	430	100	500	60	15	1
120	516	250	600	60	14	1
200	860	500	200	60	28	2
300	1275	2000	6000	120	35	5
400	1700	2500	8000	170	50	5
500	2150	3000	10000	200	77	5

During the final stages of experiments[10] two recent algorithms that perform a very similar local search for satisfiability and may be seen as variations of Hopfield networks became known to us [Selman et al. 92], [Gu 92]. In GSAT [Selman et al. 92], maximum MAXT tries are executed. In each try a random truth assignment is generated and variable "flips" are performed until either a solution is found or MAXT flips were performed. In each flip, only one of the variables is selected for flipping. The variable to be flipped is selected randomly among the variables which when flipped cause

[9] The way we generate the formulas is different from [Mitchell et al.92]. Our generator forces the formulas to be satisfiable, whereas in [Mitchell et al.92], random formulas are generated that are not forced to be satisfiable and later the Davis-Putnam algorithm [Davis, Putnam 60] (which is based on resolution) is used to eliminate the unsatisfiable formulas. Our approach seems to make the distribution generated easier than that of [Mitchell et al.92] (B. Selman, private communication). Experiments with the "unforced" version of the generator are in process but are not reported in this chapter.

[10] Our experimental design had a different idea for random generation of satisfiability problems. We changed our benchmark design to meet the ratio reported in [Mitchell et al.92].

the *largest* increase in satisfied clauses (largest absolute gradient). GSAT has been reported to perform significantly better than the Davis-Putnam algorithm which is one of the most popular algorithms for satisfiability [Davis, Putnam 60]. We have implemented GSAT for the purpose of comparing the approaches. In [Gu 92], all the variables which increase the number of satisfied clauses are flipped. The algorithm of [Gu 92] was not directly implemented by us because of its close similarity to the Hopfield version.[11] For the purpose of fair comparison with GSAT we revised our experiments and took the values for n, m and the MAXT and MAXC parameters were taken from the experiments reported in [Selman et al. 92]. The rest of the parameters (MAXP, STEPS and DELTA) were intuitively taken according to the size of the problem.[12] No fine tuning of these parameters was done.

The table below gives the average number of cycles in which each of the algorithms found a solution.

n	m	GSAT	Hopfield	Boltzmann	MFT
50	215	268.22	24.64	24.04	18.73
70	301	436.43	33.37	28.36	13.14
100	430	1095.35	69.43	83.89	55.59
120	516	1374.47	49.99	59.04	33.05
200	860	4817	85.92	88.39	46.78
300	1275	8771.8	105.2	101.68	55.92
400	1700	16247	154.32	129.14	106.26
500	2150	50664	297.82	233.54	152.06

The next table shows the percentage of experiments in which each of the algorithms managed to find a solution in the first trial:

[11] The difference is that in [Gu 92] nodes are visited in a predefined order, and the vector that is generated when the algorithm fails is not truly random.

[12] The STEPS parameter was taken to be 0.75 of the average number of cycles which Hopfield had in a successful try.

n	m	GSAT	Hopfield	Boltzmann	MFT
50	215	59%	69%	81%	94%
70	301	15%	61%	83%	94%
100	430	58%	68%	70%	93%
120	516	45%	65%	71%	87%
200	860	44%	66%	83%	96%
300	1275	54%	74%	90%	96%
400	1700	60%	80%	88%	94%
500	2150	22%	66%	86%	92%

The reader should note that the comparison with GSAT is based on parallel execution. A cycle (a GSAT flip or a single update of all the units) is assumed to run on parallel architecture and to take a constant time.[13]

The connectionist approaches are clearly leading. Not surprisingly, MFT has the best performance for first-hit. Based on preliminary studies, it is conjectured that the first hit scores improve as more annealing time is given to MFT.

5. Learning Representations of Formulas

So far we have seen that networks can be compiled from logic formulas. However, much of the appeal of connectionist models is their ability to learn from examples. This section shows that SNs can learn energy functions that represent unknown propositional formulas inductively and develop a representation that is equal to the the representations generated by compiling the formulas.

Assume the network tries to learn an unknown formula φ by looking at the set of the satisfying truth-assignments of φ. For simplicity, let us assume that the formula to be learned is a satisfiable WFF. The task of the network is to update its weights in such a way that at the end of the learning process the energy function is equal to the one obtained by translating φ into E_φ. Clearly, by doing so, the set of global minima of the energy function is equal to the set of satisfying models of φ (Γ_φ), which is the training set.

We may look at the process as learning a content-addressable memory. Given a set of vectors to be stored as memory, assuming that those vectors are the satisfying models of some unknown formula, we would like to construct a network such that the global minima of its energy function are

[13] Constant time for a cycle is certainly true for the connectionist approaches; however, a constant time for parallel GSAT cycles is not so obvious. I conjecture however, that a GSAT cycle can be computed in a constant average time with a suitable parallel architecture.

exactly equal to the vectors presented[14] with size linear in the length of the unknown formula.

The algorithm that will be described uses high-order units (sigma-pi) and connections that are hyper-arcs. The reader should remember, however, that it is always possible to convert the hyper-arcs into pairwise connections by adding hidden units.

DEFINITION **5.1** A k-CNF is a WFF that is formed as a conjunction of clauses, where each clause is a disjunction of up to k literals. A literal is either an atomic proposition or a negated ($\neg$) atomic proposition.
For example $(A \vee \neg B) \wedge (\neg A \vee \neg C \vee D)$ is a 3-CNF that is composed of two clauses: the first contains two literals and the second contains three.

I now present a new update rule for symmetric connections and a fast algorithm that learns a representation of an unknown k-CNF formula from the truth assignments that satisfy the formula. These truth assignments represent the possible realities that satisfy the unknown rule, and they are also called (in this context) *examples* or *presentations*.

After each presentation, the network is updated, and the corresponding energy function is guaranteed to have a set of global minima that is exactly equal to the set of presentations seen so far. Therefore, assuming we know the k value of the unknown k-CNF formula, the desired network is generated after a single scan over the training set.

Note that every formula can be expressed in k-CNF form; thus, the algorithm works in theory for every set of presentations and any unknown formula φ. It is exponential in k and therefore *not* practical when k is too large.[15]

5.1. A Learning Rule for High-Order Symmetric Connections

Let an instantiation of the visible units $X_1, ..., X_n$ be a vector $x = (x_1, ...x_n)$, such that $x_i \in \{0, 1\}$. A presentation is an instantiation of the visible units, introduced by clamping the visible units X_i with the values x_i. The learning rule soon to be described is responsible for the update of the weight of a single l-order hyper-arc. The rule is composed of two parts: the first part checks whether an arc should be updated as a result of the current presentation, while the second part updates the weight.

[14] The memories in such a network are content addressable: given a partial description of a stored vector clamped on some of the visible units, the network searches to complete the rest of the visible units with the stored vectors.

[15] Fortunately, expert domains are regulated by relatively short rules and therefore small k is sufficient for many practical domains.

Checking whether to update the arc:
The idea is that certain bit-patterns in the training set should cause an update of some weights, if they are seen for the first time.[16] A new bit-pattern of k bits in the training set causes the updating of arcs that connect units involved in this bit-pattern. A hyper-arc is updated only if all the units it connects participate in the new pattern, while the rest of the units of the pattern are not active; i.e., units of the pattern that do not appear in the arc should be instantiated as zeros.

Updating the weight:
Once it has been determined that the arc needs to be updated, the procedure to update a weight may be viewed as an extension of the Hebbian rule for high-order connections. If the number of activated units that participate in the hyper-arc is even, the weight is increased; otherwise, it is decreased. For the special case of a pairwise connection, we get the familiar Hebbian rule that increases the weight if the two units have the same activation value (both active or both inactive) and decreases the weight otherwise [Hebb 49].

The k-clause learning rule (formally):
Let $Arc = \{X_{i_1}, ..., X_{i_l}\}$ be an l-order undirected arc.
Given a presentation $\boldsymbol{x} = (x_1, ..., x_n)$ that instantiates the visible units to 0/1 values, the l-order arc Arc is updated iff there exists a k-bit pattern $P = (X_{i_1} = x_{i_1}, ...X_{i_l} = x_{i_l}, X_{j_1} = 0, ..., X_{j_{k-l}} = 0)$ that has never been seen in one of the earlier presentations i.e., the new pattern must include the units of Arc, and the rest of the units $(X_{j_1}, ..X_{j_{k-l}})$ must be zero-instantiated.
If this condition holds, then the weight of Arc is incremented $(+1)$ if the number of zero units in Arc is even (including the all ones case), and is decremented (-1) if the number of zeros is odd.

EXAMPLE 5.1 Given the presentation $ABC = 011$, a 2-clause rule causes the following updates:
- The weight of arc $\{AB\}$ is decremented $(\Delta_{AB} = -1)$, since the 2-bit pattern $(A = 0, B = 1)$ is new and the arc contains an odd number of zeros.
- The weight of $\{BC\}$ is incremented $(\Delta_{BC} = +1)$, since the 2-bit pattern $(B = 1, C = 1)$ is new to the arc and the arc contains no zeros (even).
- The bias of unit B (which is the singleton arc $\{B\}$) is incremented $(\Delta_B = +1)$, since the 2-bit pattern $A = 0, B = 1$ is new, the units of the pattern that are not in $\{B\}$ are zero-instantiated and the arc B in-

[16] This is one-shot learning: once a pattern is seen, it is captured completely and is not needed any longer; i.e., multiple occurrences of the same pattern do not provide us with more information. In contrast to Bayesian learning, the probability with which a bit-pattern occurs is irrelevant to the rule we want to learn.

cludes no zeros (even).
The bias of A is not updated since no 2-bit pattern contains a zero instantiated unit other than A.
In a similar way, the arc $\{AC\}$ is decremented, and the bias C is incremented.

5.2. Learning k-CNF

We have seen an update rule that changes weights of hyper-arcs; however, the question is how do we start the process and how can we guarantee that all examples are learned.

The idea is to start with the set of visible units disconnected, and construct and update connections using the update rule as more examples are presented. Each time an example appears, the weights change so that the set of global minima includes the new example.

Algorithm to learn a k-CNF formula:
Initialize all weights to zero.
For all the presentations in the training set
use the $k-$clause rule to update the weights of all the l-order arcs $(0 < l \leq k)$
that need to be updated.

THEOREM 5.1 IF THE PRESENTATIONS ARE TRUTH ASSIGNMENTS THAT SATISFY SOME UNKNOWN k-CNF FORMULA φ, THEN THE ALGORITHM GENERATES A NETWORK WHOSE GLOBAL MINIMA ARE EXACTLY THE SET OF PRESENTATIONS. THE NETWORK IS GENERATED AFTER A SINGLE PASS OVER THE PRESENTATIONS.

The network, however does not grow linearly with the presentations. Rather, it remains compact, and its size at the end of the process is not greater than the size of the unknown formula. The network is generated after one pass over the training set, and there may be $O(2^k)$ weight updates for each new bit pattern (worst case).

EXAMPLE 5.2 Learning the XOR formula $((A \oplus B) \leftrightarrow C)$ by looking at the four satisfying truth-assignments $ABC \in \{011, 101, 000, 110\}$.
We need a 3-clause rule since we cannot express the formula in less than 3-CNF. The patterns we look for are therefore 3-bit patterns (the presentations themselves).
Given the presentation $ABC = 011$:
$\triangle_{ABC} = -1$ (odd number of zeros); $\triangle_{BC} = +1$ (even number of zeros);
Given $ABC = 101$:
$\triangle_{ABC} = -1$ (odd); $\triangle_{AC} = +1$ (even) ;

Given $ABC = 000$:

$\Delta_{ABC} = -1; \Delta_{AC} = +1; \Delta_{BC} = +1; \Delta_{AB} = +1; \Delta_A = -1; \Delta_B = -1; \Delta_C = -1;$

Given $ABC = 110$:

$\Delta_{ABC} = -1; \Delta_{AB} = +1;$

The energy function obtained by summing the updates (after reversing signs) is $E = 4ABC - 2AC - 2BC - 2AB + A + B + C$ and is shown as a network in Figure 4(a). Its global minima are exactly the four presentations, and the high-order connection $(4ABC)$ can be replaced by second-order connections $(4AB - 8AH - 8BH + 8CH + 12H)$ by adding one hidden unit H (shown in Figure 4(b).

Fig. 4. The network for XOR $(A \oplus B) \leftrightarrow C$ that was constructed by the learning algorithm. (a) is the high-order network; (b) is the quadratic equivalent with one hidden unit H.

The algorithm as described so far needs to know k a priori. Can we modify the algorithm to work when the learner does not have prior knowledge of k? To solve this problem, the general learning theory approach is to start with low k (for example $k = 1$), to activate the k-clause learning rule for an entire cycle, and then to test whether the network "performs well."[17] If the network passes the test, the algorithm stops; otherwise, k is

[17] The definition of "Performing well" is designed for a specific task and usually uses

increased and another cycle of learning begins. Formally,

A general scheme for learning k-CNF when k is unknown:

1) $k = 1$; /* try monomials first */
2) Activate k-clause learning rule on all the examples of the training set;
3) TEST: if the network performs sufficiently well, stop;
4) $k = k + 1$;
5) Goto 2.

One approach to testing is discussed in [Haussler et al. 88] and is related to Valiant's notion of "probably approximately correct" (PAC) learning [Valiant 84]. In this model, positive and negative examples are given from some arbitrary distribution. The task is to find in polynomial time a network whose chance to have an error rate larger than ϵ (in classifying the same distribution), is less than δ, for arbitrary small ϵ and δ. We would also like to make sure (with probability greater than $1 - \delta$) that the algorithm will find k that is not larger than necessary, thus avoiding the construction of unduly large networks. The approach in [Haussler et al. 88] satisfies the above conditions and guarantees that only polynomial time is needed (polynomial in $n, 1/\delta, 1/\epsilon$, even if there are an exponential number of satisfying models). The PAC model assumes that both negative and positive examples are presented. The update rule discussed in this section uses only the positive examples; however, the test is based on checking whether the network correctly *classifies* enough of the examples.[18]

Another similar approach that is not PAC-motivated but similar in style involves a complete training set that contains only the satisfying assignments (only positive examples). The task is to generate a network from positive examples that performs well (with probability less than δ the network has a chance greater than ϵ to have the wrong minimum) and that is compact (the probability of using unnecessarily large k is also less than δ). The test in this case is a sequence of "samples" of the network behavior; each "sample" tests whether the network converges to a nonglobal minimum.

In both approaches, the test procedure allows up to m errors out of r checks.[19] If the number of errors is less than m then the test succeeds; otherwise, it fails and k is increased. A full discussion of these testing techniques is beyond the scope of this chapter. The reader is referred to

---

ϵ, δ approximation.

[18] An example is classified as positive if the energy rank associated with the example is minimized (globally). To test for a global minimum it is enough to check whether the energy is equal to the energy of one of the positive examples that have been learned already.

[19] m and r are computed from the ϵ-δ values using Chernoff bounds.

literature on learning and approximation theory.[20]

6. Discussion

The following sections summarize the results of the chapter and discuss the following related issues: 1) how to build an inference system that enables the user to query a propositional knowledge base; 2) how the knowledge in such a system may be incrementally updated; 3) related work.

6.1. A Connectionist Inference System

We saw that networks may be viewed as satisfiability problem solvers for the propositional knowledge captured in their weights and topology. Can we use this property to perform inferences in propositional logic? Given a knowledge-base (KB) ψ, we would like to construct a network that is capable of answering whether a given query φ is entailed (from ψ).

By definition, φ is entailed from ψ iff every model that satisfies ψ also satisfies φ. Thus, the idea is to build a network that searches for a satisfying model of ψ which also satisfies $\neg\varphi$. If such a model is found, then $\neg\varphi$ is consistent with ψ, and therefore φ is not entailed. If we cannot find such a model, then we may conclude that all the satisfying models of ψ satisfy also φ, and therefore φ is entailed.

In order to guarantee that all global minima of our network satisfy ψ, larger penalties (weights) are used for the penalty terms constructed from ψ. Another constraint $T \leftrightarrow \varphi$ is added that causes T to indicate whether φ is satisfied or not; i.e., T holds iff φ holds. A bias is then added to T that causes the network to try and satisfy $\neg\varphi$, but in a way that does not generate new global minima. The bias for $\neg\varphi$ is applied by assigning a small positive threshold (negative bias) to T. The small bias causes the network to prefer models that satisfy ψ and also $\neg\varphi$. If, however, no such models exist, the preferred models (global minima) will satisfy ψ and also φ. The network therefore tries to find a model that satisfies $\psi \wedge (\neg\varphi)$ but settles on a model that satisfies $\psi \wedge \varphi$ if all models of ψ also satisfy φ. In the first case, when ψ and $\neg\varphi$ are satisfied, φ is violated and therefore T will be set to zero; in the second case, φ is satisfied and T is set to one. At a global minimum, therefore, the system provides the correct answer $T = 0$ iff φ is not entailed, and the answer $T = 1$ iff φ is entailed. Figure 5 demonstrates a network that decides whether $A \wedge (\neg A \vee B) \models B$. The energy terms for $\psi = A \wedge (\neg A \vee B) \wedge (T \leftrightarrow \neg B)$ are scaled (multiplied by a factor of 3) to make them larger than the bias for $\neg T$. The energy function that corresponds to

[20] The chapter does not carry any contributions related to these techniques. I only suggest their adaptation for the testing of step three of the general learning scheme.

Fig. 5. Network for deciding whether $A \wedge (\neg A \vee B) \models B$. The answer is provided in T.

$\psi \wedge (T \leftrightarrow \neg\varphi)$ is therefore $-3AB + 6TB - 3T - 3B$. The bias for $\neg T$ causes the term $+T$ to be added, and the result is $-3AB + 6TB - 2T - 3B$. For a somewhat more complex architecture that supports nonmonotonicity, see [Pinkas 91a].

6.2. Incremental Updating

An important engineering desideratum is the ability to update the knowledge base incrementally without recompiling the entire KB. The networks that are generated by either compiling formulas or inductive learning are capable of incremental updating. When a new fact or rule is added to the knowledge base, there is no need to re-compute all of the weights. We can find the energy function that describes the new fact and then take advantage of the fact that the penalty of a conjunction is the sum of the penalties. All that is needed is to add the new energy terms to the old terms. This way only the weights that are affected by the new fact will be updated (deleting a fact is done by subtracting the energy terms).

EXAMPLE **6.1** Assume that the original KB contains the rules:
$N \wedge S \rightarrow W$

$R \rightarrow (\neg N)$
$N \vee (\neg W)$
$S \rightarrow N$
$N \vee R$
The network $-NSW + 2RN - WN + W + S - R - N$ is shown in Figure 6. To add the WFF $N \rightarrow R$, we need to compute the penalty function for this WFF and add it to the previous energy function. In the case of $N \rightarrow R$ we add $N - NR$ to the previous function. The result is $-NSW + RN - WN + W + S - R$. If we now wish to delete $R \rightarrow (\neg N)$, we subtract RN and get the function $-NSW - WN + W + S - R$. The new knowledge base is represented by the network of Figure 6.

Fig. 6. The energy function $-NSW - WN + W + S - R$ captures the knowledge base after adding the WFF: $N \rightarrow R$ and deleting $R \rightarrow (\neg N)$.

6.3. Related Work

Derthick's $\mu KLONE$ takes an approach similar to the one discussed in this chapter [Derthick 88]. He uses numerical functions to express logical constraints and translates logical expressions to energy functions. However, his functions involve multiplication and exponentiation and are not particularly connectionist in their nature. The energy functions generated are

not polynomials, the individual constraints are not restricted to pairwise connections and the units have to implement functions that are much more general and powerful than those that are usually assumed in connectionist networks.

In [Grüninger 89], the author discusses the relationships between Hopfield networks, Reiter's nonmonotonic logic and prioritized circumscription. Grüninger manages to show that 3-SAT (3-CNF formulas) may be reduced into energy functions. His approach is different from the one described in this chapter. His reduction relies heavily on the assumption that the clauses are disjunctions of three literals and generates a separate network for each of the eight possible clauses. His approach generates more hidden units than are actually necessary, and his system (like Derthick's) *cannot learn* these representations.

Logical formulas may be viewed as specifications for constraints over binary domains. Constraint networks [Mackworth 77], [Dechter, Pearl 88] are graphs where each node i may obtain a value from a domain D_i. An arc (i, j) represents a set of tuples that is contained in $D_i \times D_j$ and specifies the consistent values that nodes i and j may obtain. The constraint satisfaction problem (CSP) is to find a value for each of the nodes that satisfies all the constraints. It is well known that arbitrary high-order constraints can be expressed in constraint networks if we allow hyper-arcs or hidden units with non-binary domains. The surprising result in [Dechter et al. 90] is that one *cannot* express arbitrary binary constraints in constraint networks of binary domains with pairwise arcs, even if we add hidden (binary) units; i.e., hidden units do not make constraint networks with binary domains more expressive. In contrast, the results of this chapter provide a way to represent arbitrary constraints in *connectionist* networks using only pairwise connections (and zero/one domains). The reason for such different results is that the pairwise constraints in the connectionist approach are weak and have weights, whereas the constraints in constraint networks are hard (must be satisfied). It is impossible to express boolean constraints in *hard* pairwise constraints even with hidden nodes, but it is possible to express them using weak (weighted) pairwise constraints.

As anticipated by NP-completeness, the problem of general constraint satisfaction may be reduced to energy minimization. In this chapter, an efficient reduction is given for binary domains. The method may be extended to encode arbitrary domains and arbitrary constraints by encoding the values of the domain as binary numbers and using one node for each bit. A more direct approach towards the reduction of *general* CSP is described in [Hertzberg, Güsgen 91].

6.4. Summary

I have shown how satisfiability instances may be linearly converted into problems of minimizing energy functions. The reduction is in the sense that for every WFF we can efficiently find a quadratic energy function, such that the global minima of the function are exactly equal to the satisfying models of the formula. The same energy function can be learned from examples using a new learning rule for high-order symmetric connections. The representations compiled or learned may be used to perform inferences with propositional logic and may be updated incrementally once knowledge is added (or deleted). Experimental results indicate that the local hill-climbing suggested in this chapter gives extremely good average case performance even for large-scale, hard satisfiability problems that are randomly generated. All the variations tried were surprisingly fast and manage to out-perform GSAT.[21] The best method seems to be a combination of MFT and Boltzmann.

Acknowledgment Thanks to Dan Kimura, Stan Kwasny, Ron Loui and Dave Touretzky for helpful discussions.

References

[Abu Mustafa, Psaltis 87] Y. S. Abu Mustafa, D. Psaltis, "Optical neural computers," *Scientific American 256*, pp. 88-95, March 1987.

[Alspector et al. 88] j. Alspector, R. B. Allen, V. Hu, S. Satyanarayana, "Stochastic learning networks and their electronic implementation," in D.Z. Anderson (ed.), *Neural Information Systems*, pp. 9-21, Denver, 1987.

[Davis, Putnam 60] M. Davis, H. Putnam, "A computing procedure for quantification theory," *Journal of the Association of Computer Machinery 7*, pp. 201-215, 1960.

[Dechter, Pearl 88] R. Dechter, J. Pearl, "Network-based heuristics for constraint-satisfaction problems," *Artificial Intelligence 34*, pp. 1-38, 1988.

[Dechter et al. 90] R. Dechter, A. Dechter, J. Pearl, "Optimization in constraint networks," in R. M. Oliver, J. Q. Smith, Influence Diagrams, Belief Nets and Decision Analysis, John Wiley and Sons, 1990.

[Dechter 90] R. Dechter, "On the design of networks with hidden variables," Technical Report, CIS-9007, Center for Intelligent Systems, Technion, Haifa, Israel, July 1990. Also in AAAI, 1990.

[Derthick 88] M. Derthick "Mundane reasoning by parallel constraint satisfaction," PhD thesis, CMU-CS-88-182 Carnegie Mellon University, Sept. 1988

[Derthick 90] M. Derthick "Mundane reasoning by setting on a plausible model," *Artificial Intelligence 46*, 1-2, pp. 107-158, 1990.

[21] Which was reported to outperform the Davis-Putnam algorithm.

[Feldman 85] J. A Feldman "Energy and the behavior of connectionist models," Technical Report, TR-155, Computer Science Department, University of Rochester, 1985.

[Fodor, Pylyshyn 88] J. A. Fodor, Z. W. Pylyshyn, "Connectionism and cognitive architecture: A critical analysis," *Cognition 28*, pp. 3-71, 1988.

[Garey,Johnson79] M. R. Garey, D. S. Johnson "Computers and Intractability - A Guide to the theory of NP Completeness," W.H. Freeman and Company, San Francisco, 1979

[Grüninger 89] M. J. Grüninger, "Model theory for parallel distributed processing," MS Thesis, Department of Computer Science, University of Toronto, 1989.

[Gu 92] J. Gu, "Efficient local search for very large satisfiability problem," *Sigart Bulletin 3*, 1, pp. 8-12, 1992.

[Haussler et al. 88] D. Haussler, M. Kearns, N. Littlestone, M. Warmuth, "Equivalence of models for polynomial learnability," *Information and Computation* (to appear), also in *The Proceedings of the Workshop on Computational Learning Theory*, pp. 42-55, 1988, also as Technical Report, UCSC-CRL-88-06, 1988.

[Hebb 49] D. O. Hebb, *The organization of behavior*, New York, Wiley, 1949.

[Hertzberg, Güsgen 91] J. Hertzberg, H. W. Güesgen, "Transforming constraint relaxation networks into Boltzmann machines," In *Proceedings of the German Workshop on Artificial Intelligence*, pp. 244-253, Springer, 1991.

[Hinton et al. 84] G. E Hinton, T. J. Sejnowski, D. H Ackley, "Boltzmann machines: Constraint satisfaction networks that learn," Technical Report, CMU-CS-84-119, Carnegie-Mellon University, Department of Computer Science, Pittsburgh, 1984.

[Hinton 89] G. E Hinton "Deterministic Boltzmann learning performs steepest descent in weight space," *Neural Computation 1*, 1, pp. 143-150, 1989.

[Hinton 90] G. E. Hinton, "Preface to the special issue on connectionist symbol processing," *Artificial Intelligence 46*, 1-2, pp. 1-4, 1990.

[Hinton, Sejnowski 86] G. E Hinton and T. J. Sejnowski, "Learning and relearning in Boltzman Machines," in J. L. McClelland and D. E. Rumelhart, *Parallel Distributed Processing: Explorations in The Microstructure of Cognition I*, pp. 282 - 317, MIT Press, 1986.

[Hopfield 82] 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.

[Hopfield 84] J. J. Hopfield "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the National Academy of Sciences 81*, pp. 3088-3092, 1984.

[Hopfield, Tank 85] J. J. Hopfield, D. W. Tank "Neural Computation of Decisions in Optimization Problems," *Biological Cybernetics 52*, pp. 144-152.

[MacCarthy 68] J. McCarthy, "Programs with commonsense," in M. Minsky (ed.), *Semantic Information Processing*, pp. 403-418, MIT Press, 1968.

[Mackworth 77] A. K. Mackworth, "Consistency in networks of relations," *Arti-*

ficial Intelligence 8, 77, pp. 99-118, 1977.

[Minton et al. 90] S. M. Minton, M. D. Johnson, A.B. Philips, P. Laird, " Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method," *Proceedings of the Eighth Conference on Artificial Intelligence*, pp. 17-24, 1990.

[Mitchell et al.92] D. Mitchell, B. Selman, H. Levesque, "Hard and easy distribution of SAT problems," *Proceedings of the thenth National Conference on Artificial Intelligence*, pp. 459-465, 1992.

[Peterson, Hartman 89] C. Peterson, E. Hartman, "Explorations of mean field theory learning algorithm," *Neural Networks 2*, 6, 1989.

[Pinkas 90] G. Pinkas, "Energy minimization and the satisfiability of propositional calculus," *Neural Computation 3*, 2, pp. 282-291, 1991. Also in Touretzky, D.S., Elman, J.L. Sejnowski, T.J. Hinton, G.E. (eds), *Proceedings of the 1990 Connectionist Models Summer School*, pp. 23-31, San Mateo, Morgan Kaufmann.

[Pinkas 91a] G. Pinkas, "Propositional non-monotonic reasoning and inconsistency in symmetric neural networks," *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pp. 525-530, Sydney, 1991.

[Pinkas 91b] G. Pinkas, "Converting binary threshold networks into symmetric networks," Technical Report, WUCS-91-31, Washington University, Computer Science Department, 1991.

[Pinkas 91c] G. Pinkas, "Constructing proofs in symmetric networks," in J. E. Moody, S. J. Hanson, R. P. Lipmann (eds.), to appear in *Advances in Information Processing Systems 4* (NIPS), pp. 217-224, 1992.

[Pinkas 92] G. Pinkas, "Logical inference in symmetric connectionist networks," Doctoral Thesis, Washington University 1992.

[Rumelhart et al. 86] D.E. Rumelhart, G.E Hinton, J.L. McClelland, "A general framework for parallel distributed processing," in J. L. McClelland and D. E. Rumelhart, *Parallel Distributed Processing: Explorations in The Microstructure of Cognition I*, MIT Press, 1986.

[Sejnowski 86] T. J. Sejnowski "Higher-Order Boltzman Machines," Neural Networks for Computing, *Proceedings of The American Institute of Physics 151*, Snowbird Utah, pp. 3984, 1986.

[Selman et al. 92] B. Selman, H. Levesque, D. Mitchell, "A new method for solving hard satisfiability problems," *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 440-446, San Jose, 1992.

[Shastri 88] L. Shastri, "Semantic networks: An evidential formulation and its connectionist realization," Pitman, London, 1988.

[Shastri, Ajjanagadde 90] L. Shastri, V. Ajjanagadde, "From simple associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings," Technical Report, MS-CIS-90-05, University of Pennsylvania, Philadelphia, 1990.

[Shoham 88] Y. Shoham, *Reasoning about Change*,

[Smolensky 86] P. Smolensky, "Information processing in dynamic systems: Foundations of harmony theory," in J.L.McClelland and D.E.Rumelhart,

Parallel Distributed Processing: Explorations in The Microstructure of Cognition I , MIT Press, 1986.

[Valiant 84] L. G. Valiant, "A theory of the learnable," *Communications of the ACM 27*, pp. 1134-1142, 1984.

[Williams 86] R. J. Williams, "The logic of activation functions," in J. L. McClelland and D. E. Rumelhart, *Parallel Distributed Processing: Explorations in The Microstructure of Cognition I* , MIT Press, 1986.

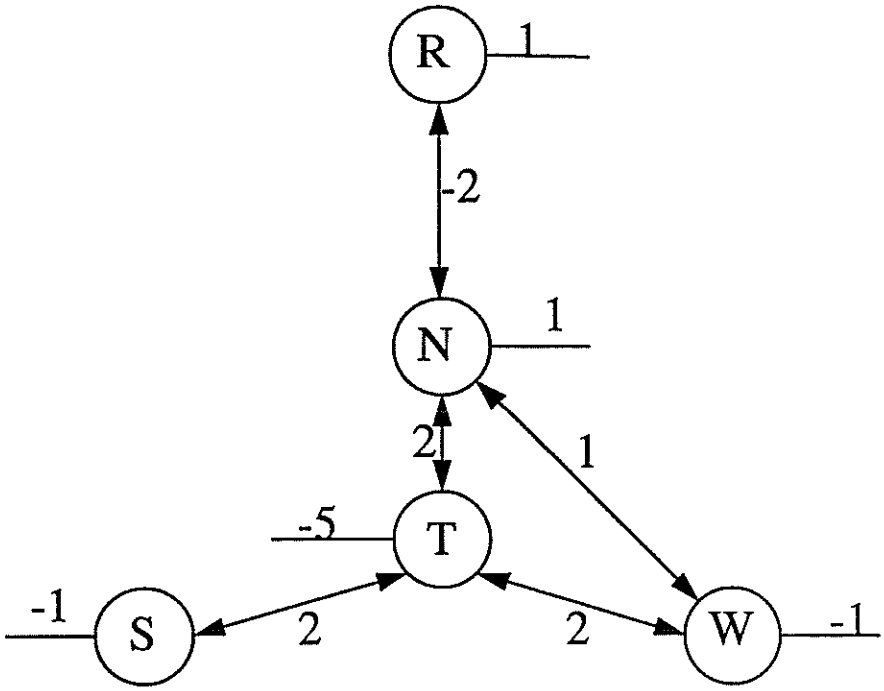


Figure 1

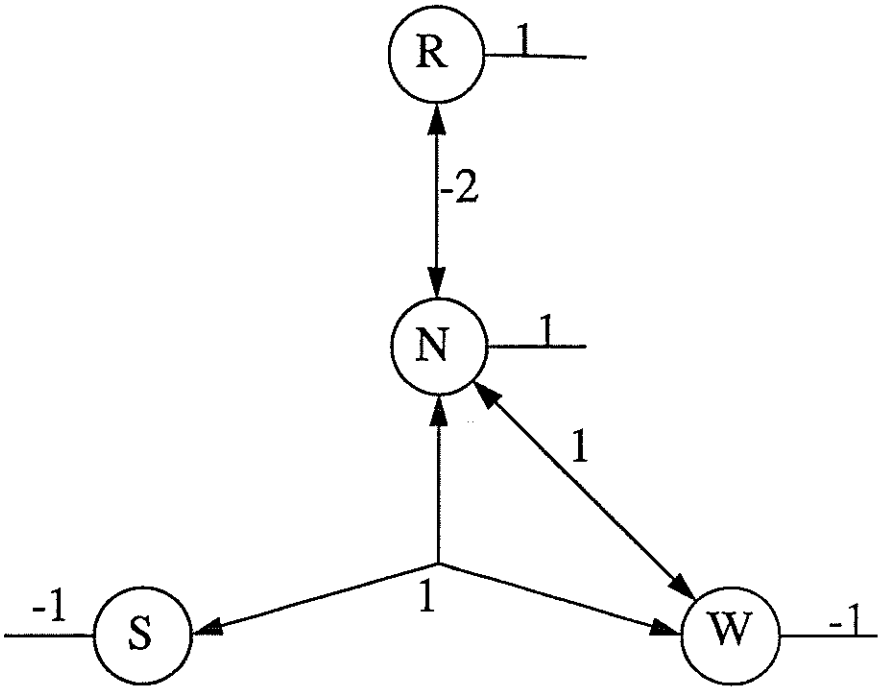


Figure [redacted] 2

[redacted]

2

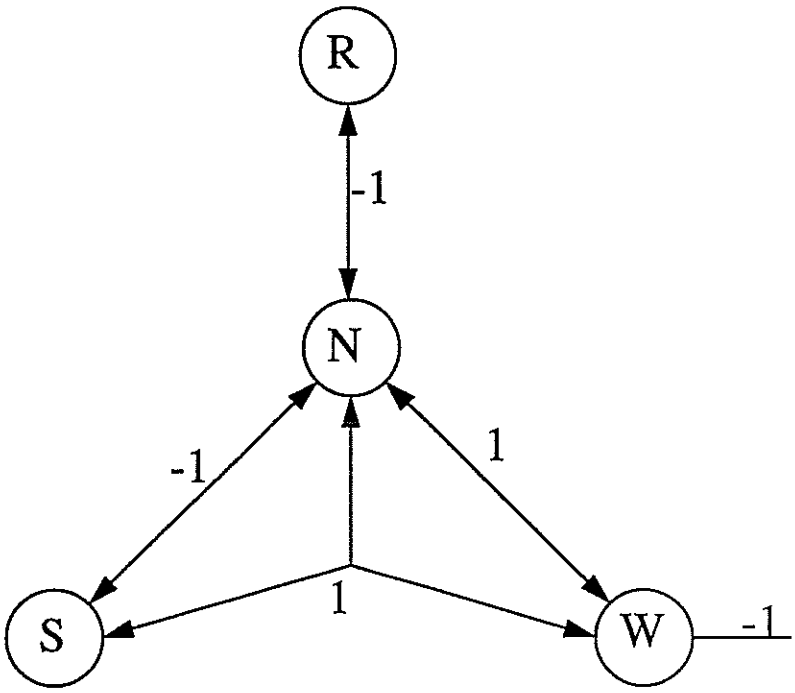
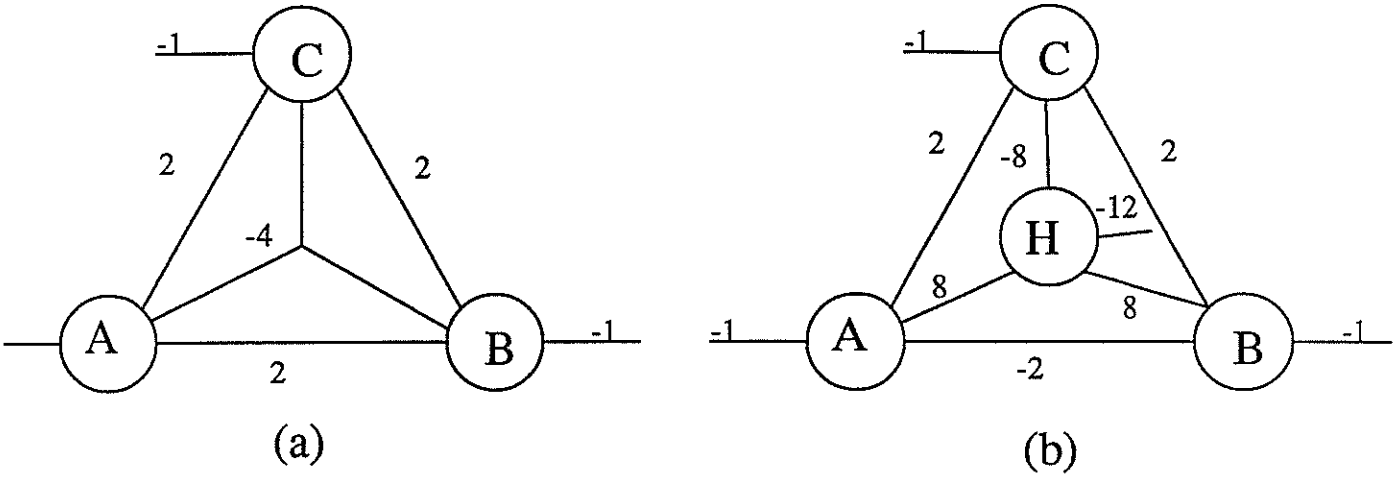


Figure 3



XOR

Figure 4

4

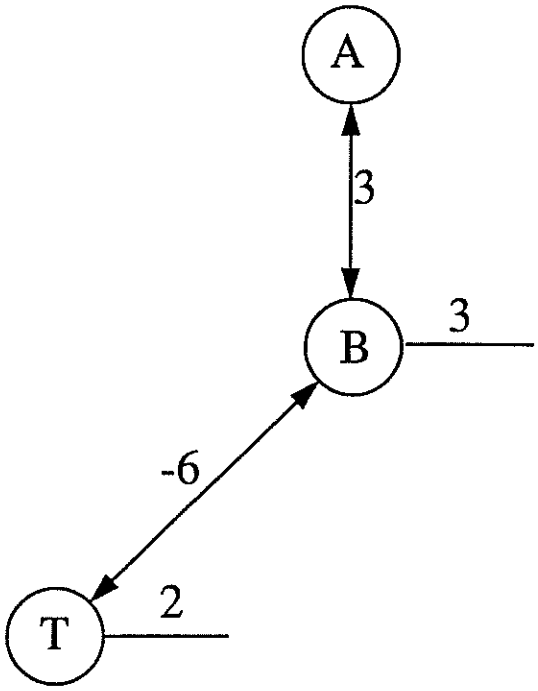


Figure [illegible]

fig4:engine

[illegible] 5

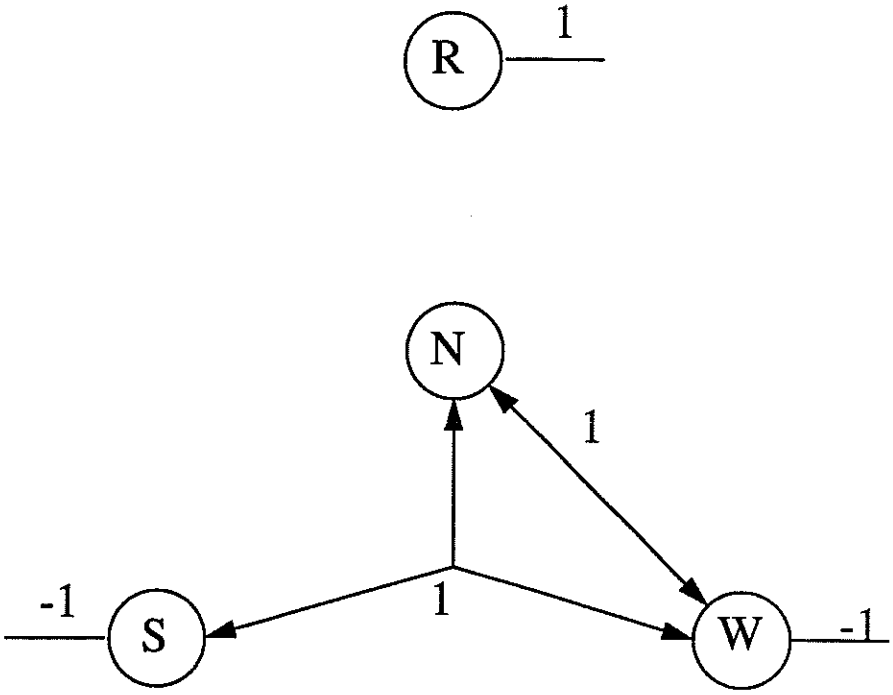


Figure 6

6